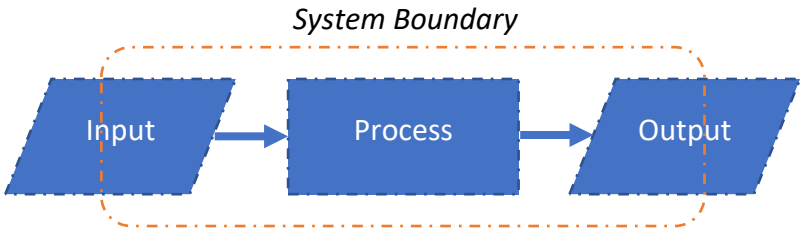
Commenting Program Source Code

Has what you said ever been mistaken for what you meant? Don't you wish the love of your life had understood what you meant *before* they heard what you said? (It can't be unsaid, so back into the dating pool for you!) Well, programmers are people, too.

Comments are what the program means, code is what the program does.



See the Programming input, process, output model.

The system boundary is the imaginary border between the program and its environment. Many C programs operate in the command line interface environment where the host operating system connects a C program's stdin to a keyboard and stdout to a screen where a user types input and reads output. Sometimes, like acceptance testing on matrix, stdin and stdout interface with software, not a human. Those are indeed different environments.

Comments explain what input crosses the system boundary from the outside world (user, file, another program) into the program, what processing the program does with input to create output, and what output is sent outside the system boundary.

A program cannot not know the world outside its system boundary. But programmers do. And programmers' comments explain what travels across the boundary. The program sees a variable as an integer. The programmer and the user see that number as a meaningful measure of value (42) or a unique identifier such as a student number – what the integer *means*.

Do describe program operation in terms of how it interacts with the user: the user is the input source and output target outside the system boundary. In what way does the program process / transform the input to satisfy the user's purpose and meet the user's objective?

You know you have good comments if you delete all the code and what's left still makes sense as a program. If the comment makes sense *without* the code, it is a good comment telling the story of the program.

Comment on the purpose: *Why* was this program, function, structure, or line of code written in the first place? What does it do in service of the needs, goals, or intent of the user? It may help to consider "What part of the story does this code support?"

Comment on the relationship this structure or group of lines has to other parts of the program. (E.g. from user input, calculate…)

Comment the program, not the code. Anyone reading a source file will be a programmer – they don't need individual lines of code explained! If they want to know *how* the code works, they will read the code. *The reader wants to know the relationship this line of code has to the rest of the program.*

What is the entrance / exit criteria for an iterative structure?
(e.g. describe how the loop/program knows when to stop)

Code often changes the state of the program. (e.g. any statement that does assignment =) Good variable names will explain what is happening. However, bad variable names beg for an in-line comment.

It may help to consider what would happen if the line was removed from the program.

Spelling counts. Clarity in communication and respect for the reader shows your attention to detail. Mist stakes imply acyurasee does nt madder to you – wy shud the reeder trist your coed? If the code is as well written as the comments, things are likely to be good, or very very bad.

Properly **formatted code** should be considered the first and last steps when commenting a program. In Visual Studio IDE, Ctrl+KD.

75% of IT coding budgets are spent on maintenance. During new programming development (the remaining 25% of coding budgets), comments are often extensive; they explain the intent of the code to be written, its purpose, the reasoning behind the logic to be implemented. Yes, the comments are written *before* the code. Development comments should satisfy the mythical DoWIM compiler: the "Do What I Mean" compiler ignores code and compiles comments.

Comments are for your future self at 4am when you've been called out of bed to fix a program crash. Comments are for programmers who will maintain the code in the future. Comments are read only by IT professionals who may very well know the language better than you. *You don't need to explain the code.* If they want to know how the code works, they will read the code. Most of the time, they want to know if this is the source file they should be working on, the purpose of the program in general, what service that a function provides, and in the case of individual structures (a loop, a series of nested IF – ELSE), what the structure accomplishes.

Comments are brief and informative. One or two phrases takes one or two seconds to read. A few lines of code can take minutes to analyse, compile in one's head, perform a virtual walkthrough, check the results, yet the why and how of the code may still be misunderstood.

Without good comments in a professional environment, the code is neither acceptable nor economically maintainable.

Have your source code ready when the DoWIM compiler gets out of beta.

Organisation of Comments

Custom .h header files deserve comments to remind us what the library/constant/etc. is, why it is used here, and why a programmer went to the trouble of creating a .h file instead of putting it all in the .c source file.

Program comments

– appear at the beginning of a source file.

On the #include *custom*.h statement, give a summary of the .h file so the reader knows why it is there…without having to open and look inside the .h file.

/*
Author: *Name, email, ID, Date written, Course, Project*
[executable filename] : *[title of program]*
Purpose: *[what this program does, what problem does it solve?]*
*/

Function comments

The function's name() should serve as the title / purpose of the function. But there is only so much it can communicate to the reader.

/*
Purpose: *[what this function does, what problem does it solve?]*

Parameters: *Include only if needed to explain values passed (by copy|reference) when the function is called. Ideally, parameter names are self-explanatory.*
Returns: *Include and explain if not* `void`. *Nobody wants to read through your code to discover all the different values for* `int` *that the function returns or what those values mean.*
Modifies: *Include and explain if values in global variables or variables with pointers (pass by reference) are changed.* This is a critical comment for future maintenance. *Everyone does not notice that a variable modified in the body of the function is* not *in the parameter list (global variable), nor do they scour the function's parameter list for an *asterisk (pointer). Information hiding is assumed in functions: the function is expected to be a "black box" which does not change the state of anything outside the function. If that is not the case, the function has "side effects" and that must be clearly documented.*
*/

Inline code comments

Your comment must say something different than explaining the code itself.
`c = a + b;` // that c is the sum of a and b is obvious. That is what the code says; it does not need to be explained to a C programmer. What does that cryptic code mean?

Ideally, variable names should be self-explanatory. When they are not, comments are required.

`c = a + b;` // c stores total of assignment and test marks respectively.

Inline comments are placed on the same line as the code, tabbed to stand apart from the code. This format makes it easy to see the code and the comments, separately (vertically) or together (horizontally).

```
cryptic = C + code;          // explain this line's purpose in the program
cryptically = C + moreCode;  // explain this line's purpose in the program
```

Longer code comments

Sometimes comments need more space than would fit inline, so a comment is entered on its own line(s).

```
cryptically = C + moreCode;
/* Does this comment relate to the line above or below? We cannot know without analysing both lines of code which defeats the purpose of commenting. */
moreCryptically = C + cryptically;
```

Always put comments ABOVE the function() | {structure} | code.
Use vertical spacing to group comment and code lines together.

```
// this comment explains the purpose of the next line of code
cryptically = C + moreCode;

// this comment explains the purpose of the next line of code
moreCryptically = C + cryptically;
```

Never reference a line number in source comments. Just one newline inserted or deleted at the top of the source file and your comments are all broken.

Structures

Commenting of a { structure } states its purpose, so the reader does not have to analyse multiple lines of code to see how they work together to accomplish something.

// *[what the structure accomplishes]*
e.g.
// compute factorial

// prompt user until value within range 1 – 100 is input

Code Samples

```
printf("Type a few words separated by space(q - to quit):\n");
gets(words); // no one should be using gets() We'll fix it Someday.
while (strcmp(words, "q") != 0)
{
    word = strtok(words, " ");
    w_counter = 1;
    while (word)
    {
        printf("Word #%d is \'%s\'\n", w_counter++, word);
        word = strtok(NULL, " ");
    }
    printf("Type a few words separated by space(q - to quit):\n");
    gets(words); // fix this too!
}
```

- the first **`while`** continues until the 'words' variable is equal to "q". What is it for?
- there is another **`while`** {structure} ... what does it do?
    - How does it know when to stop looping? 'word' does not seem like a self-explanatory boolean.
    - **`// keep looping until pointer is NULL`**
      This just says what the code says. It should explain what work the while loop accomplishes -- think in terms of a specification/instruction an analyst would write to tell a programmer what code to create.
- **`}`** This is the end of scope for *which* structure? If the code above it scrolls off the screen and all the programmer sees is **`}`**, they either must remember the previous screen (not likely), or page up to see the preceding code (too much bother), or you could enter a comment (which they will appreciate).
    - This technique is even more important at the end of a series nested structures, e.g.
      `}` // I know what you're thinking: "Is this the end of the IF structure or one of the nested loops?"
      `}` // Well, to tell you the truth, in all this excitement, I've kinda lost track myself.
      `}` // But being this is C, the most powerful programming language in the world, which could blow up your weekend, you've got to ask yourself one question: 'Do I feel lucky?' Well, do ya?

Unusually written code must be commented. A structure such as `while (TRUE) { … }` is unconventional. It is obviously intended as an infinite loop. What is its exit condition? There must be a `break` within its scope. But where?

```
while (TRUE) // a comment explains why there is no exit condition here
{
...
if ( ... ) continue; // explain why the program repeats the loop structure.
// ------- ********  ← make it visually obvious that the loop iterates from here
...
if ( ... ) break;    // explain why the program exits the loop structure.
// ------- *****     ← make it visually obvious that the loop exits here
...
}
```

See https://npp-user-manual.org/ for a good comment about documentation.