# Kinds of Comments (worst to best)

1. Repeat of code
2. Explanation of code
3. Marker in code
4. Summary of code
5. Description of code’s intent

\* “Information that can not possibly be expressed by the code itself”

===========================================================================

## Repeat of Code

Tells what the code does in words ◦ Basically useless. This is what I call “comment pollution”.

|  |
| --- |
| // assign 100 to aint a = 100; |

##

## Explanation of Code

If code is so complicated or confusing that it needs explanation then it probably needs rewriting

|  |
| --- |
| memcpy(tstr,f\_rec,8);i=0;while ((tstr[i] != 0x20) && (i<8)) i++; tstr[i]=0; |

|  |
| --- |
| // copy the first 8 chars of the directory entry data memcpy(tstr,f\_rec,8);// scan the 8 chars above for a space, don’t go over 8 i=0;while ((tstr[i] != 0x20) && (i<8))  i++;// terminate with a 0 tstr[i]=0; |

1. Marker in Code

|  |
| --- |
| // TODO: make changes here//MJW012111: Matt, I commented out this line and replaced it with the line that follows |

## Summary of Code

Simple summary of code into one or two sentences

|  |
| --- |
| // Function -// read the file indicated by the current dirEntry and rebuild // it on the local disk by successively reading and writing // clusters that make up the file on the disk being recovered |

## Description of Code’s Intent (Most useful)

|  |
| --- |
| // extract the root of the filename from the directory entrymemcpy(tstr,f\_rec,8);i=0;while ((tstr[i] != 0x20) && (i<8)) i++; tstr[i]=0; // \* still, this should be rewritten |

## \*Information that can not...

● Copyright notices

● Confidentiality

● Comment blocks assigned by instructor

● Citations of sources

### **Example:**

//----------------------------------------------------------

// Copyright 2007, eV Interacitve, Inc.

// Modification of this code is not permitted without express written

// consent of Matthew Harmon [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*@ev-interactive.com](https://wiki.illinois.edu/wiki/display/cs242/Commenting)

//----------------------------------------------------------

###

### **Example:**

//----------------------------------------------------------

// Lambert Conformal Conic Projects

// Formula on page for the sphere, pg. 106

// Map Projects – A Working Manual

// U.S. Geological Survey Professional Paper 1395

//----------------------------------------------------------

##

## Excuses for Bad or No Comments

“It Takes Too Much Time”

* It takes more time later and is harder to debug
* Commenting after the fact is more difficult
* My experience and that of EVERY professional that I have spoken to says otherwise

##

## Style: Comment Blocks

### **Example 1**

|  |
| --- |
| /\*-----------------------------------------------------\*//\* here’s a difficult style of \*//\* comment block to maintain \*//\* You’ll spend too much time \*//\* keeping the right hand column lined up \*//\*-----------------------------------------------------\*/ |

(keep in mind that your code might not be viewed with a color-coded syntax highlighting system)

### **Example 2**

|  |
| --- |
| //---------------------------------------------------------// this style is easier because you don’t// have to align the right hand column// you can just copy and paste the dashes// to start and end the block//--------------------------------------------------------- |

##

## Commenting Techniques

Write comments at the level of the code’s intent

Focus on the code itself so that comments enhance good code

Focus paragraph comments on why rather than how

1. Endline comments
	1. Tend to be cryptic
	2. Hard to maintain
	3. OK for data declarations
	4. OK to denote bug fixes
	5. OK for marking ends of blocks
	6. Some people use this method to pair up curly braces
		1. e.g. – “ } // end while”
	7. Probably not very useful
2. Commenting individual lines
	1. Complicated line of code (which you should just rewrite)
	2. Bug repair
	3. Overuse creates “speedbumps”
3. Commenting Paragraphs (blocks)

|  |
| --- |
| //==============================================// get next cluster fat\_index=0;//==============================================if (cluster >= (FAT\_ENTRIES/2)) { cluster = cluster-(FAT\_ENTRIES/2); fat\_index++; }cluster = \*(F\_tbl[fat\_index]+cluster); |

The better style would be:

|  |
| --- |
| cluster = getNextCluster(F\_tbl, cluster); |

###

## Comment the Why, not the How

Example from “recover.c”

|  |
| --- |
| if (track >= 1024) printf("%i WARNING (cylinder) in %s\n",status, filename);else write\_count = write(outFile,io\_buff, bytes\_2\_write); |

Preferable:

// this BIOS can not read tracks greater than 1024 without special drivers

// print warning but keep recovering the file anyway

|  |
| --- |
| if (track >= LAST\_LEGAL\_TRACK) printf("%i WARNING (cylinder) in %s\n",status,filename);else write\_count = write(outFile,io\_buff,bytes\_2\_write);//\*notice the literal or magic number “1024” replaced with “LAST\_LEGAL\_TRACK” |

## Commenting Routines

* Say what the routine WON’T do, mention permissible input values
* Document global effects (if any)
* Side effects (are dangerous)
* Create or destroy anything?
* Document source of algorithms
* Avoid enormous comment blocks
◦ I like some comments before every routine for visual delineation at the very least

##

##

## Balance

* useful amount of commenting
	+ If it takes too much time to wade through comments then there are too many comments
	+ If there’s as much or more comments than code then there are too many comments
	+ Rule of thumb: 1 line of comment for 10 lines of code
	+ Do not count lines and then add comments every 10th line!
* Avoid “comment pollution”
	+ inane commenting
	*inane - adjective*
		1. silly; stupid.
* Avoid "speed bumps" - comments with no redeeming value, they do not improve comprehension
* Metrics or “rules of thumb” such as “comments per 10 lines of code” are not particularly useful but they are intended to give some idea as to what is considered reasonable by the “ruling body”