JACOBS UNIVERSITY

Documentation

Credits:
Richard Clegg

Instructor: Peter Baumann

email: p.baumann@jacobs-university.de
tel: -3178
office: room 88, Research 1

*"Real programmers don't document.
If it was hard to write,
it should be hard to understand."*

Roadmap: Types of Documentation

- Internal documentation
  - *What: comments in your code*
  - Level of detail: local (particular statements, variables, …)
- External programmer documentation
  - *What: for other programmers who would work with your code*
  - Level of detail: global, implementation directed (module dependencies, interfaces, anything else of interest);
    where necessary: details (algorithms, data structures, restrictions, …)
- User documentation
  - *What: the manual for the poor fools who will be using your code*
  - Level of detail: global, usage directed

Internal (Inline) Documentation, or: *How to Write Good Comments*

- Does your comment help your reader understand the code?
- Are you writing a comment just because you know that "comments are good"?
- Is the comment something that the reader could easily work out for themselves?
- Don't be afraid to add a reference instead of a comment for tricky things
- See history.js

Some Common Bad Comments

```
i= i+1;  /* Add one to i */

for (i= 0; i < 1000; i++) { /* Tricky bit */
.
. Hundreds of lines of obscure uncommented code here
.
}
int x,y,q3,z4;  /* Define some variables */

int main()
/* Main routine */

while (i < 7) { /*This comment carries on and on */
```

How Much To Comment?

- Just because comments are good doesn't mean that you should comment every line
- Too many comments make your code hard to read
- Too few comments make your code hard to understand
- Comment only where you couldn't trivially understand what was going on by looking at the code for a minute or so

What Should I *Always* Comment?

- Every file to say what it contains
- Every function – what input does it take and what does it return
  - Preconditions
  - Postconditions (eg, error return values)
  - I like to comment prototypes too, slightly, to give a hint
- Every variable apart from "obvious" ones
  - **`i,j,k`** for loops, **`FILE *fptr`** ~~don't require a comment~~
  - but **`int total;`** might
- Every struct/typedef
  - unless it's *really* trivial

It does - not for the fptr,
but for the file purpose!
(see top)

Other Rules for Comments

- Comment if you do something "weird" that might fool other programmers
  - In particular: "tricks", optimizations
  - *Aka natural penalty: the more tricky, the more to comment…*
- If a comment is getting long consider referring to other text instead
  - external documentation
- Don't let comments interfere with how the code looks
  - e.g. make indentation hard to find
- Keep comments up to date!
  - Outdated comments are worse than no comment at all: misleading

How Comments Can Make Code Worse

```
while (j < ARRAYLEN) {
    printf ("J is %d\n", j);
    for (i= 0; i < MAXLEN; i++) {
/* These comments only */
        for (k= 0; k < KPOS; k++) {
/* Serve to break up */
            printf ("%d %d\n",i,k);
/* the program */
        }
/* And make the indentation */
    }
/* Very hard for the programmer to see */
    j++;
}
```

External (Programmer) Documentation

- Tells other programmers what your code does
- The aim is to allow another programmer to use & modify your code without having to read &understand every line
- Here just ONE way of doing it – everyone has their own rules
  - Most large companies have their own standards for doing this
- Global structure:
  - Stage 1: overview & purpose
  - Stage 2: the mechanics
  - Stage 3: the gory details: globals
  - Stage 4: the gory details: locals

External Documentation (Stage 1)

- What is your code supposed to do?
- How does your code work generally?
- What files does it read from or write to?
  - Purpose only, not internals
- What does it assume about program input?
- What algorithms does it use?

External Documentation (Stage 2)

- Describe the general flow of your program
  - no real need for a flowchart though
  - Diagrams can help
- Explain any complex algorithms which your program uses or refer to explanations elsewhere
  - e.g. "*I use vcomplexsort, see Knuth page 45 for details*"

External Documentation (Stage 3)

JACOBS UNIVERSITY

- If you use multi-file programming explain what each file contains
- Explain any struct which is used a lot in your program
- explain (and justify) any global variables you have chosen to use

External Documentation (Stage 4)

- Describe every "major" function in your program:
  what arguments must be passed, what is returned
  - It is up to you to decide what is a "major" function
  - …and really depends on the level of detail you wish to document to
- Consider which functions are doing "the real work"
  - they might not necessarily be the longest or most difficult to write ones

User Documentation

- This is documentation for the user of your program (aka "user manual“)
- *Entire books have been written on the subject!*
  - Sometimes it is written before your code is even ready to be tested
  - For highly structured and complex projects it is likely that you will have to adapt your code to match the user manual
  - It has to be written from the point of view of the end users of your program
  - Many, many more considerations and guidelines not covered here…

Tool Support

- C++:
  - Doxygen, doc++
- Java:
  - Javadoc
- General:
  - doc-to-help: generate online help + word documentation from same source

Finally: ECCS overview (1/2)

- European Cooperation for Space Standardization (http://www.ecss.nl/)
  - develop a coherent, single set of user-friendly standards for use in all European space activities
  - publicly available, result of consultation with space agencies in Europe and industry, designed to secure acceptance by users
  - requirement should address its need, rather than the means to fulfill it
- Four “Space” branches
  - Engineering (E), Project management (M), Product assurance (Q), Space sustainability (U)
  - Each with several disciplines

Finally: ECCS focus (2/2)

- Engineering branch has software discipline (E-40)
- Software general requirements (E-ST-40C)
  - concerns “product software”, i.e. software that is part of a space system product tree and developed as part of a space project
- Several documentation “folders”
  - Requirements Baseline, Design Definition, Design Justification, Technical Spec
- In Design Definition, there is “Software Design Document” (SDD)
- For each software “component”: Identifier, Type, Purpose, Function, Subordinates, Dependencies, Interfaces, Resources, References, Data