Documentation

Computer Science Practice and Experience: Development Basics
CS1XC3

Professor: Kevin Browne
E-mail: brownek@mcmaster.ca

Documenting C code

- Documenting C code isn't fundamentally different from documenting any other type of code
  - So much of what we talk about today is really relevant across different languages

- But in the instructor's opinion...
  - Higher-level languages such as Haskell tend to be more ***self-documenting*** than lower-level languages like C
  - So with C in particular pay attention to documenting how lower-level features (e.g. pointers) implement higher-level designs (e.g. edges between trees)

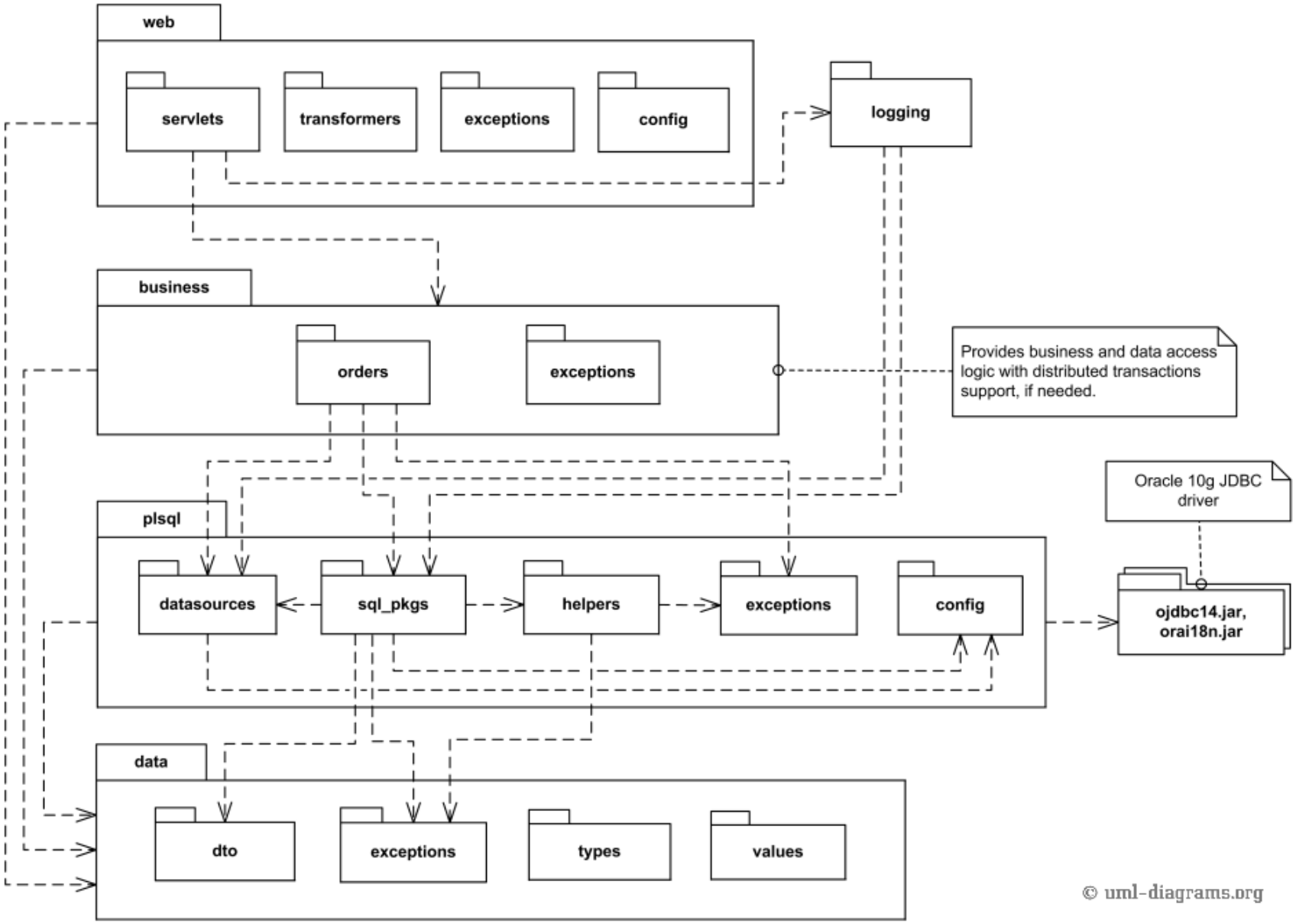
How can we document our code?

- We can put comments in our code to explain things
  - Comments are then right next to what they document

- We can create an external document in a word processor (e.g. Word) or other system (LaTeX, HTML) external to our code
  - We can then create things like navigation or "tables of contents" that allow us to overview functionalities
  - We can document multiple code files in one document
  - We can use formatting to make content more readable

How can we document our code?

- Using an external document to provide an overview of code is something used in **software architecture**
- Software architecture is about the high-level components of a software system and how they interact
- There is overlap between documenting code and documenting a software architecture



Source: https://www.uml-diagrams.org/examples/package-diagrams-example-layers.png

How can we document our code?

- Documenting a software architecture is about higher-level connections between components
  - Software architecture documentation is aimed at more stakeholders (testers, project leads, etc.)
  - e.g. How are different databases accessed, and by what components?

- Code documentation about lower-level implementation, documents how and why code implements a larger design
  - Code documentation audience is aimed at other developers
  - e.g. What are the parameters of a function and what is its return value?

Functions

Day Of Week As String

Returns the day of the week for example "Monday"

Parameters

`datetime` date time to be converted to a week day

Returns

Day of the week as a text string

See Also

Day Of Week As Short String
Month Of Year As String
Month Of Year As Short String

Source: https://community.jmp.com/t5/image/serverpage/image-id/9060iE6684636CD071039/image-size/medium?v=1.0&px=400

How can we document our code?

- Comments and an external document both have pros and cons
  - Comments document code in-place
  - External documents can summarize, format, allow navigation (e.g. hyperlinks), and illustrate connections
  - But producing both individually would involve reproducing content... then what if we need to make changes later? Do we change it in two places?

- Wouldn't it be nice if we could have the best of both worlds?
  - We can with document generator tools like Doxygen!

Document generators

- **Document generators** create software documentation from source code files
  - Documentation is typically in a format such as HTML, LaTeX, PDF, Word, etc.
  - Source code files are typically annotated with special comments intended to be processed by the document generator
  - The special comments indicate precisely what and how the document generator should capture certain features
  - Document generators can also recognize code features (functions, structs, files, etc.) and group documentation in a sensible way based on these features

Document generators

- Document generators allow us to document our code ***once***, with comments, in the code itself
  - BUT we also have the advantage of being able to generate a highly readable external document too!

- We'll talk about and use a document generator called doxygen today and next week

- But before we do that, let's talk a bit about commenting code in general...

Comments

- Exactly how to comment code is a bit subjective...
  - It's not like writing a C function that either works or does not work... but that doesn't mean there isn't "good" and "bad"
- Many people have strong opinions about "how"
  - Virtually none of which are supported by any evidence!
  - Including the idea that ***all*** comments are bad practice
- What we can do though is...
  - Always use a set of general do's and don'ts guidelines
  - Follow a set of specific guidelines when working as a team for consistency across a project

CODE COMMENTS BE LIKE

Source: https://memezila.com/wp-content/Code-comments-be-like-this-is-a-stop-sign-meme-1022.png

90% of all code comments:

Source: https://preview.redd.it/iuy9fxt300811.png?auto=webp&s=156f5ffebbea5b30189eb36012e5bc7e3369c381

Don't: repeat in a comment ***exactly*** what a line of code is doing!

```
// Creates a new BST node with the given key
bstNode* create_node(int new_key)
{
  bstNode *newNode = calloc(1, sizeof(bstNode)); // allocate space for 1 new node
  newNode->key = new_key; // set the key to new_key
  newNode->right_child = NULL; // set the right_child to NULL
  newNode->left_child = NULL; // set the left_child to NULL
  return newNode; // return the new newnode
}
```

This doesn't give the reader any new information, adds clutter.

Do: write for your audience

- As with all good writing, write for your audience!

- Comments are for other developers
  - And "other developers" could be ***you*** 3 months from now when you completely forget how your code works!
  - There is goal is to understand how your code works and why decisions were made, most likely to modify it or use it

- You can assume the developer...
  - Knows how the language works
  - Knows what each ***individual*** line of code ***means*** (but this is different from knowing how it fits into the larger control-flow)

And on that note…

- The comments that I add to code in this course are generally "reasonably good" to view as a guide

- They're lengthier than you would usually see in industry code (though, not "bad" either)
  - ...but that's partially because 1st year students trying to learn new things are ***my*** audience
  - Textbook and tutorial comments will do this too... perhaps documenting basic things that wouldn't normally be

- As a student, who is your audience?
  - Likely the marker, maybe other students. Comments that are a ***little bit*** lengthier is probably a good idea for you too!

Don't: leave mean comments

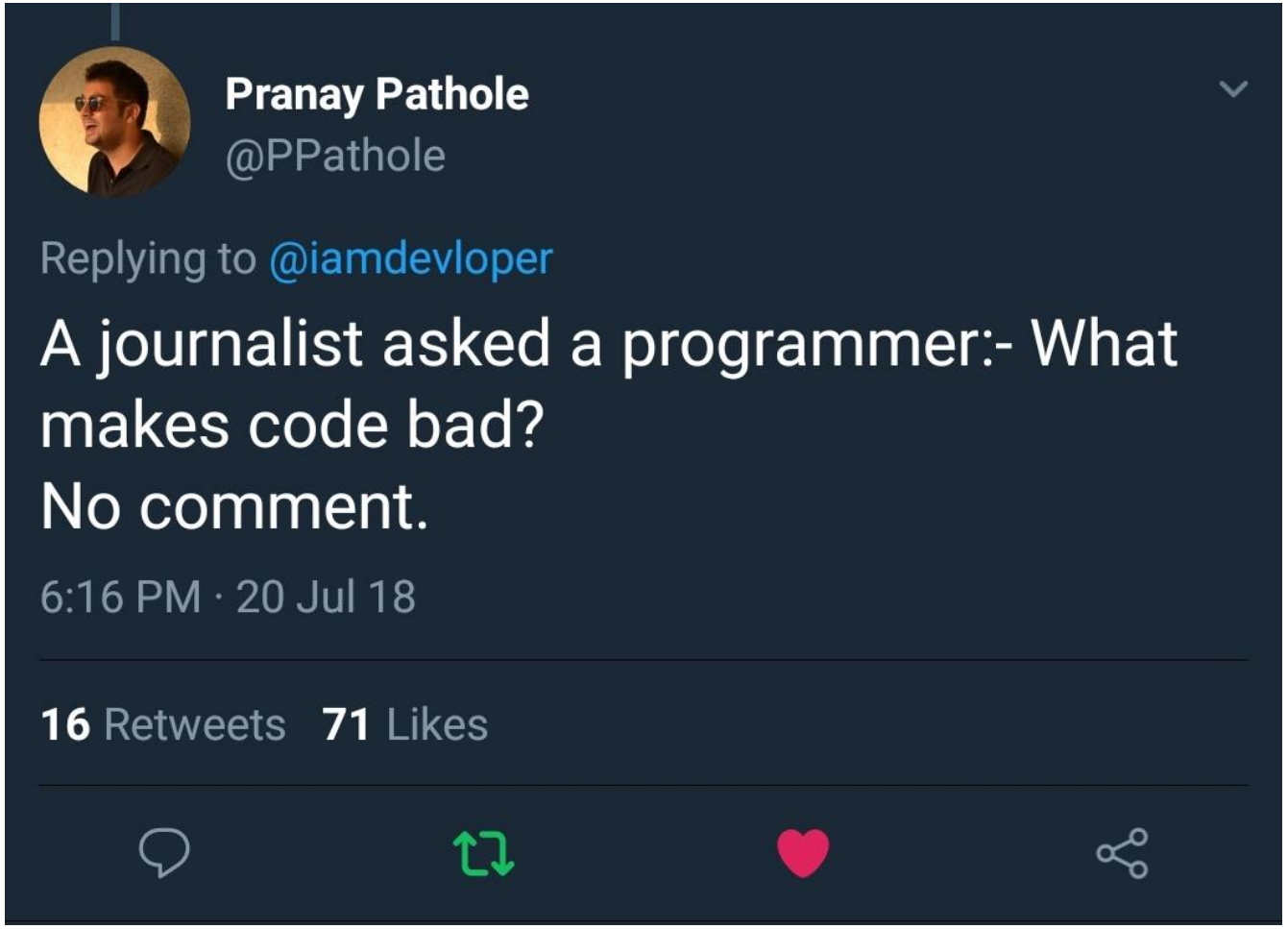
- ***Sigh***... yes... this is seriously a thing.

- Developers sometimes leave mean or unprofessional comments to be funny or blow off steam, but they don't help anyone

```
// Keeps deleting the first match in the linked list because apparently
// efficiency doesn't matter to the #$%^ing clown that wrote this code.
do
{
  current = delete_first_match(current, delete_value, &deleted);
  if (deleted) *num_deleted  = *num_deleted + 1;
} while(deleted);
```

Do: write self-documenting code

- Using clear variable and function names can make code more self-documenting
  - Reduces the need for comments
- If you're calling a function to delete_matches, it's reasonably clear you're deleting matches
- If you're using a variable called head, it's reasonable clear it's referring to the head of the data structure



Source: https://i.redd.it/x5452hhoq4b11.jpg

Do: use comments liberally

- In general leave a comment at the top of every file, function, and typedef/struct explaining what it does
  - What does each struct member do?
  - What does the function accept as parameters, return?
  - What is the purpose of the code in the file?

- Leaving a short comment at the top of most control structures is also a good idea
  - Explain what it's doing in the function's overall algorithm
  - Exception would be if it's trivial or if doing this would add too much clutter… maybe one comment at the top of a group of related control structures or a short function can cover things

Working at a big company be like

Source: https://pics.me.me/useful-comments-in-code-comments-in-code-working-at-a-37817560.png

Don't: comment every single line

- This is a different problem from explaining exactly what a line is doing
- If you're finding that you need to comment every single line to explain what it is doing, that's a bad sign... either your code can be improved or that's you're using too many comments
- Maybe functions and variable names need to be more obvious, maybe your algorithm has an issue

Do: explain how and why

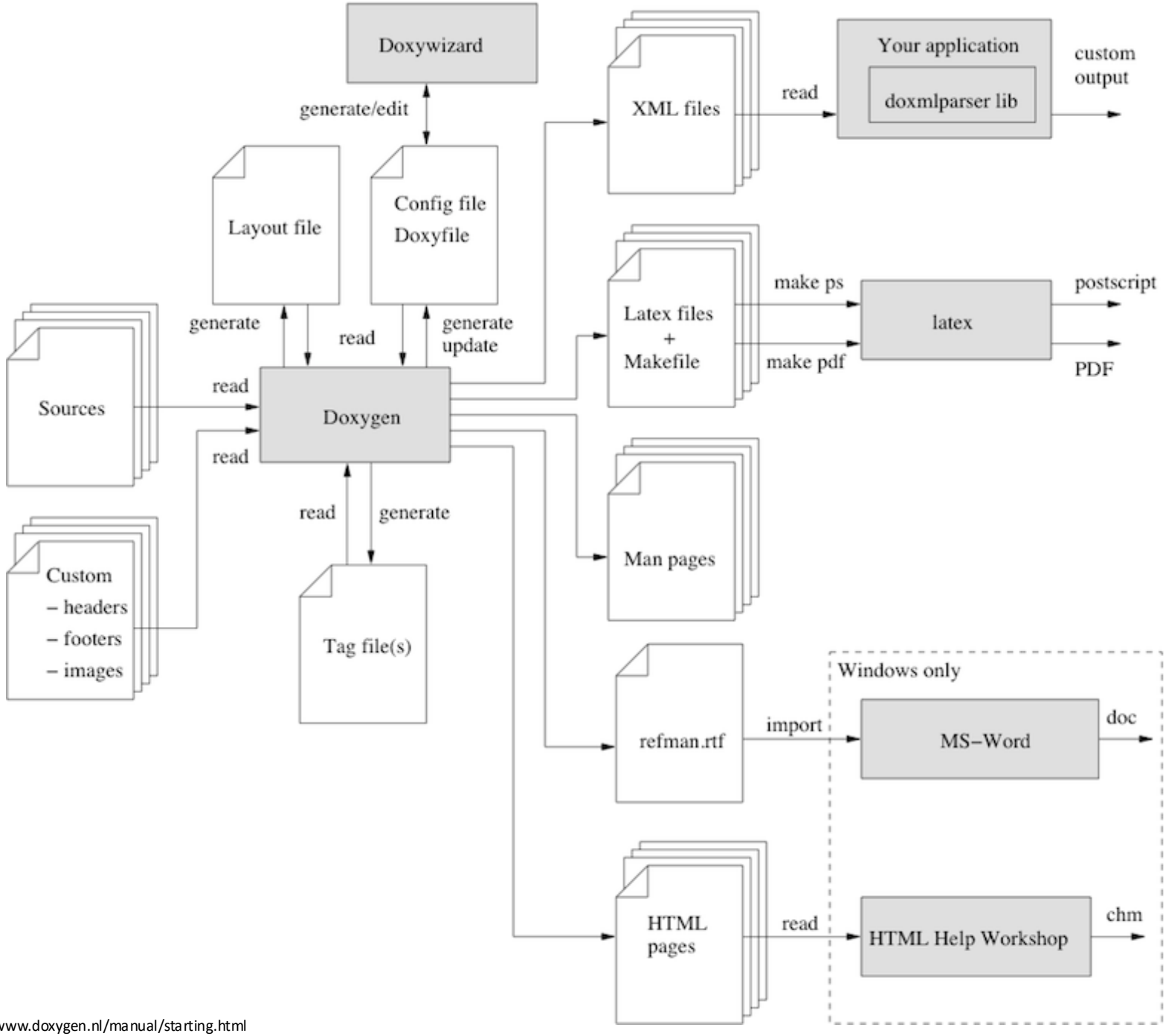
- Explain how code accomplishes what it does
  - What is the algorithm?
  - How does each block of code relate to the algorithm?
  - If an algorithm is well known (e.g. preorder traversal), citing the name may be enough to explain it
- Explain why implementation decisions were made
  - e.g. why is a certain buffer size used?
- All of this is even more important if the decisions are less obvious

Doxygen

- Doxygen is a popular document generator
  - Available at: https://www.doxygen.nl/index.html
  - Available on Unix-like systems; Windows binaries exist
  - Installed for us on the pascal server!
- Doxygen works with C, C++ and many other languages
- Doxygen works by checking .c and .h files for special comments and commands that indicate how and what to comment

Doxygen

- Doxygen produces LaTeX and HTML outputs
- HTML output is a navigable, searchable website documenting your code organized by categories
- Can configure it to produce other outputs such as Word processor formats (RTF, Word docs)
- Can produce man pages too!
  - Remember: **man command**



Source: https://www.doxygen.nl/manual/starting.html

Using Doxygen

- To use doxygen you need to first generate a doxygen configuration file ("Doxyfile")
  - Done with: **doxygen -g**

```
[brownek@pascal ~]$ doxygen -g

Configuration file `Doxyfile' created.

Now edit the configuration file and enter

  doxygen Doxyfile

to generate the documentation for your project
```

Doxygen configuration file

- Doxygen configuration file is massive, allows you to configure all kinds of things
  - **Important:** PROJECT_NAME should be set
  - Can also configure things like what directories to check for .c and .h files, what outputs to produce
  - By default, reads any .c and .h files in current directory

```
# The PROJECT_NAME tag is a single word (or a sequence of words surrounded by
# double-quotes, unless you are using Doxywizard) that should identify the
# project for which the documentation is generated. This name is used in the
# title of most generated pages and in a few other places.
# The default value is: My Project.

PROJECT_NAME           = "My Project"
```

Running Doxygen

- Run Doxygen with the command: **doxygen Doxyfile**
  - Produces html and latex folders
    - HTML folder contains website documentation, LaTeX folder contains latex file documentation
  - Note this is something we could have makefile do!

```
[brownek@pascal ~]$ doxygen Doxyfile
Searching for include files...
Searching for example files...
Searching for images...
Searching for dot files...
Searching for msc files...
```

LaTeX

- **LaTeX** is the standard way to produce technical documents for computer scientists
  - Plain text documents annotated with commands like **\begin{tabular} … \end{tabular}** (for producing tables) get compiled into formal documents in formats like PDF
  - A bit similar to plaintext HTML documents being read by a web browser to produce web page layouts/structures

- Students taking CS1DM3 learn LaTeX, but we can't assume you've seen it yet and so won't use it
  - Introduction to LaTeX for those that are curious... https://www.youtube.com/watch?v=FXdXqqqIMdk

Documenting code with Doxygen

- We can write doxygen comments like this, with two stars at the top of the comment:

```
/**

*/
```

- This signals to doxygen that it should be looking at this comment to include in the documentation!

Documenting code with Doxygen

- When we place these comments above functions, Doxygen will document the function with that comment
  - Same for structs/typedefs

- Doxygen looks for specific commands that we can specify to produce documentation
  - Commands start with @
  - @param command documents function parameters
  - e.g. **@param length the length of the linked list**

Doxygen

- There are many standard comment formats that different document generators use
  - Doxygen supports just about all of them!
- For example, instead of @command for commands, we can use a \command
  - It really doesn't matter what you use so long as you stick with a format, they provide this for convience
- See: https://www.doxygen.nl/manual/docblocks.html

Let's try using Doxygen!

- We'll play around with Doxygen a bit now to get a sense of how it works in general!
- You'll use it more in-labs next week to get to know more specific commands
- The next assignment will involve using doxygen to document code