
Week 4 Lecture 1: Best Practice

Review and Preview

- Arithmetic, Assignment, Relational and Logical Operators
- Type casting, sizeof
- The Midterm!

- Best Practice
  - Comments
  - Variable Names
- The Midterm!

K&R: Chapters 1-3
Sections 1.1 to 1.6
Sections 2.1 to 2.6
Sections 3.1 to 3.6

UNIX Command of the Day: date

- `date` allows you to display or set date and time.

- The -f flag means that the command uses a format string to parse the date rather than using the default [[[mm]dd]HH]MM[[cc]yy][.ss] format.

```
$ date "+%d %m %Y%n"
```

30 09 2019

Comments

- Why, Why not
- When, How

The Philosophy of Comments

- “Programs must be written for people to read, and only incidentally for machines to execute.”
  - *Structure and Interpretation of Computer Programs, 1985*
- “Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.”
  - *Literate Programming* by Donald Knuth

The Philosophy of Comments

- Code tells you **how** and comments tell you **why**.
- One opinion is that code should be enough of an explanation but it is doubtful that code can be completely **self-documenting**.
- Commenting decisions include:
  - When you should comment code
  - What a comment should look like

- https://blog.codinghorror.com/code-tells-you-how-comments-tell-you-why/

When You Should Comment Code

- When it is not clear from the code, **why** certain decisions were made.
- **Assumptions** are important to document.
- When the code is particularly **problematic**
  - security concerns
  - dependent on a particular version of a library

What a Comment Should Look Like

- **Header**
  - General information about development of the code
    - Not everyone likes this and if it is too large it distracts from the code.
  - Algorithm explanation
    - Small block of comments to explain an algorithm or part of an algorithm.

What a Comment Should Look Like

- **In-line**
  - Where needed - very brief comment.
  - Appropriate to explain non-obvious constants or to reference the origin of code.
  - But be careful with this type of comment - never put comments like:

```
num = 4;  /* set num to 4 */
```

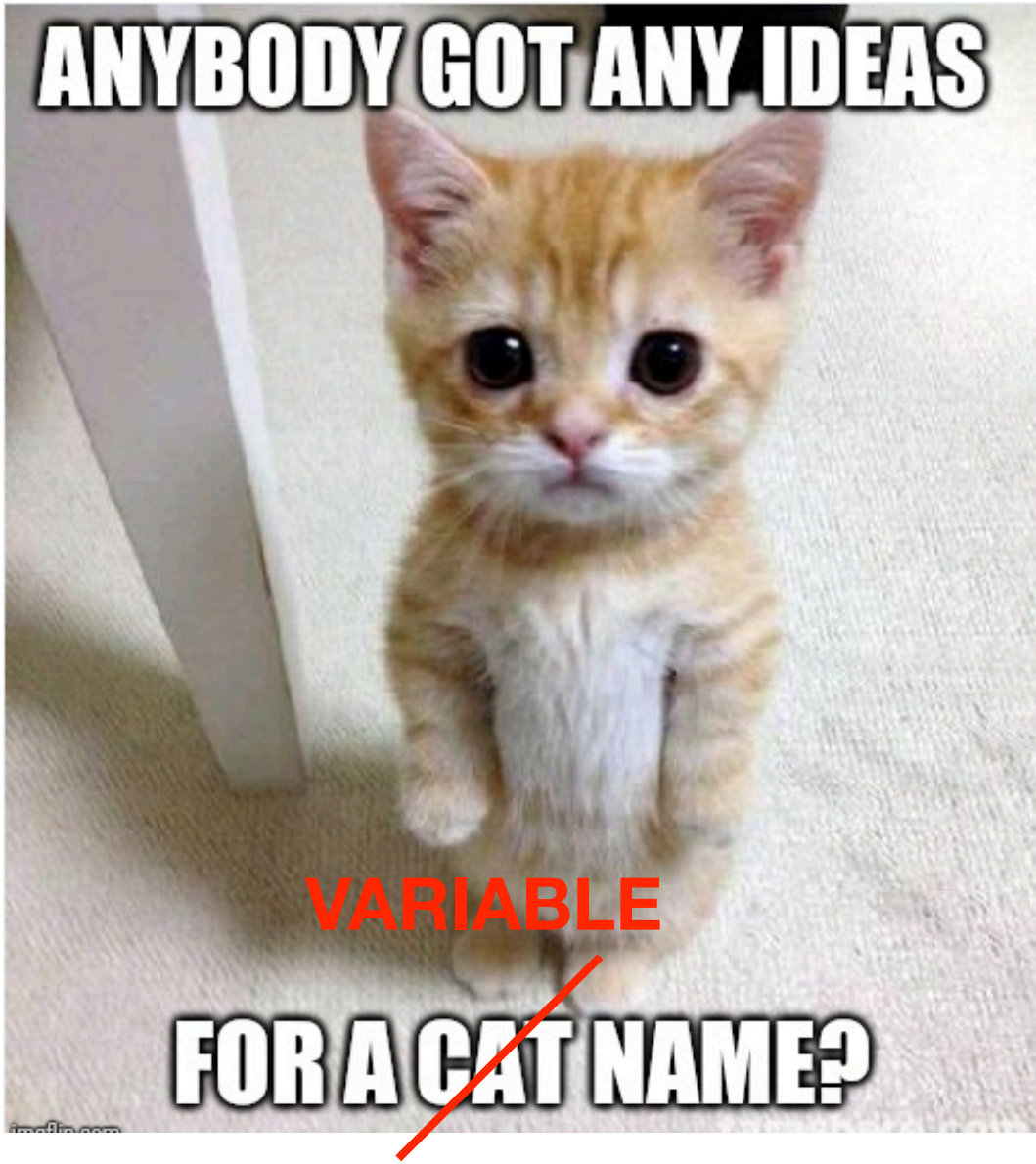
Rules of Thumb for Comments

- Keep most comments brief.
- Keep them to the point (and a single point at that).
- **Three Bears Rule**
  - Not too few, not too many,
    just the right number of comments.

- Write comments that you would be proud to read out loud to your peers and your boss - no swearing, no rudeness, no slander.

Variable Names

What's in a name?
Why is this important?
Are there guidelines?



What's in a Name?

- Variable names are part of the **documentation** of a program.
- Names should reflect what is being **stored**, e.g. are you storing the number of characters in a string, are you storing a collection of temperatures, etc.
- Variable names should help you tell the "***story***" of the program.
- Good names can document the **type** and **purpose** of the variable.

Characteristics of a Name

- **Length** - short versus long

```
int i;

int loopCounter;
```

- **Case** - do you include upper case characters?

```
int numDigits;
```

https://devopedia.org/naming-conventions

Characteristics of a Name

- **Non-alphabetic Characters** - do you use punctuation or numbers?

```
int num_digits, x-axis_2D;
```

- Intention and target reader (describe **content** not **storage**)

```
char *clientList[]; /* Business usage */

char *arrayNames[]; /* Describes type */
```

https://devopedia.org/naming-conventions

Characteristics of a Name

- **Easy to read and understand** - can you easily figure out what a name “means”?

```
int numWidgets;

int nWid;
```

- **Easy to tell apart** - is l1l2 the same as I1I2?
  - Depends on the font!
  - `l1I2` (lower case `l`, upper case `I`, number `1`)
  - `I1l2` (upper case `I`, lower case `l`, number `1`)

https://devopedia.org/naming-conventions

Short versus Long

- How long should a variable name be?
- Short names are quick to type but can be cryptic - what does `a1w2` really mean?
    - But if you are using a variable as a loop counter or array index then a short name such as `i` can be appropriate.
- But long names mean that you have to type more and that can lead to errors or even worse you might initially spell the variable name incorrectly (according to the dictionary) and then you have to keep using the **incorrect spelling** and that can lead to confusion for the next programmer.

Notations

- Notations are rules that tell you how to construct variable names.
- The idea is to consistently use a single notation throughout your program.
- There are many different types of notations: Camel Case, Pascal Case, Hungarian, Snake Case, Kebab Case.

Camel Case and Pascal Case

- **Camel Case**
  - Names are phrases where each word or abbreviation in the middle of the phrase begins with a capital letter, with no intervening spaces or punctuation.
    - The first word is not capitalized, e.g. `camelCase`
  - Capitalizing all of the words is referred to as **Pascal case** (e.g. `PascalCase`) although sometimes capitalizing each word is called `CamelCase`.

https://en.wikipedia.org/wiki/Camel_case

Hungarian Notation

- Names start with a lowercase **prefix** to indicate intention.
- The main body of the name is in **PascalCase**.
- There are two types of Hungarian notation:
  - (a) **Systems Hungarian**: the prefix indicates data type

```
intNumWidgets;
```

  - (b) **Apps Hungarian**: the prefix indicates logical type

```
char rowPosition;
```

https://en.wikipedia.org/wiki/Hungarian_notation

Other Notations

- **Snake Case**
  - Words within phrases are separated with an underscore, e.g. `first_name`
- **Kebab Case**
  - Words within phrases are separated with a hyphen, e.g. `first-name`

Conventions in C

- Identifiers representing macros (remember `#define`) are, by convention, written in uppercase with underscores separating words in a phrase.

- Names containing double underscores or beginning with an underscore and a capital letter are reserved for system functions (compiler, standard library) and should not be used by you.

    `__reserved` or `_Reserved`

Naming Rules of Thumb

- **Arrays** - since they represent multiple items give them names that are plurals,

  e.g. `int dressOrders[100];`

- **Booleans** - only have 2 values so indicate this with prefixes such as `is`, `has`, `can`,

  e.g. `isOpen`, `hasLength`.

- **Numbers** - use meaningful descriptive prefixes such as max, min, total,

  e.g. `int maxLength; int totalCount;`

https://hackernoon.com/the-art-of-naming-variables-52f44de00aad

What to do about naming variables?

- Consistency - pick one notation and stick with it.
- Read the code out loud to see if the names help (and you can pronounce the names).
- Spelling is important.
- Think about errors.
- Think about categories of variables - how can you create more meaning by grouping variables based on function.
- Write the rules down!

The Midterm!

How to study for the midterm

Midterm Preview

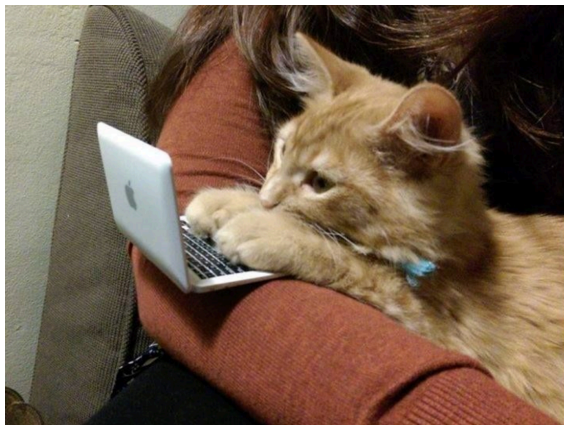
- **When**: Thursday, October 10 during class time (10:15am to 11:15am *or* 4:15pm to 5:15pm)
- **Where**: Roz 103 (lecture room)
- **Length**: 60 minutes
- **Type**: Closed book

Midterm Preview: Types of Content

- **General knowledge**
  - definitions, UNIX commands
- **Reading code**
  - Given a code snippet, what does it do
- **Writing code**
  - Given a task, write the code that implements that task

General Knowledge Questions: How to Study

- Go through the lecture notes and text and find all terms that have definitions.
- **Introduction** - what is an operating system, kernel, shell, process, program, virtual machine (VM)?
- **Week1Lecture1** - what is an example of a preprocessor directive? What does #include do?
- UNIX Command of the Day



Reading Code Questions: How to Study

- Go through all code posted to CourseLink.
- Go through all the code in the assigned textbook sections.
- Go through all code that you have written in the labs and for Assignment 1.

Writing Code Questions: How to Study

- Go through lectures - find all programming concepts and make up code that illustrates those concepts:
  - looping (for and while)
  - if else branching
  - arrays
  - command line parameters