CS 370
Variable Names

DR. MICHAEL J. REALE
FALL 2015
“**MUST** A NAME MEAN SOMETHING?” ALICE ASKED DOUBTFULLY.

“**OF COURSE IT MUST,**” HUMPTY DUMPTY SAID WITH A SHORT LAUGH:
“**MY NAME MEANS THE SHAPE I AM — AND A GOOD HANDSOME SHAPE IT IS, TOO. WITH A NAME LIKE YOURS, YOU MIGHT BE ANY SHAPE, ALMOST.**”

-- ALICE THROUGH THE LOOKING GLASS
Best Naming Guideline

- **Naming routines** → pick name that describes what the routine *does*
  - Describes the HOW

- **Naming variables** → pick name that describes what the variable *represents*
  - Describes the WHAT
  - Try to come up with description in words → often best name
  - Be as specific as possible
Avoid:

- Cryptic abbreviations
  - E.g., eitr
    - Turns out that stands for “Employee Income Tax Rate”...but how would anyone ELSE possibly know that?
  - Better name: employeeIncomeTaxRate

- Abuse of generic and/or very short names
  - Examples: x, i, x1, v, b
    - Exception: if referring to x coordinate, you can call a variable “x”
What and How

- Pick a name that describes WHAT is it, not HOW it is used/computed or some aspect of computing in general
  - *Example:*
    - bitFlag vs. printerReady
    - calcVal vs. sum
Name Length

- **Too short**
  - Not descriptive enough

- **Too long**
  - Hard to type
  - Obscures visual structure of program

- **Good average: 8-20 characters**
  - Does NOT mean you should FORCE your variables to have this length
  - Use as a potential warning sign
Short vs. Long Names

- Short name $\rightarrow$ implies:
  - Limited use, purpose, or scope
  - Scratch value
  - *Example*: i as an index counter in a loop
    - Implies garden-variety index counter
    - Has no purpose outside loop
What If You Repeat Names?

- In some cases, you may use the same name in two different contexts:
  - *Example:*
    - Employee class for User Interface subsystem
    - Employee class for Database subsystem

- If language allows it, partition global namespace
  - C++/C# \(\rightarrow\) use `namespace` keyword
  - Java \(\rightarrow\) use `packages`
    - `com.myproject.userinterface.employee`
    - `com.myproject.database.employee`

- Also prevents potential conflicts with other imported libraries
Computed-Value Qualifiers

- Often compute certain kinds of values:
  - Totals, averages, maximums, etc.
- You might use a qualifier in your variable name that implies this

**Good convention** → put qualifier on end

- **Examples**:
  - revenueTotal
  - expenseAverage
- **Reasons**:
  - Most important part of meaning up front → gets read first
  - Consistency with this avoids confusion of names
    - E.g., made `totalRevenue` and `revenueTotal`
  - Symmetry
    - revenueTotal, expenseTotal, revenueAverage, expenseAverage
  - Consistency also improves readability and ease of maintenance
- **Exception**: num → may be in front or back, depending on intended meaning
  - numCustomers → number of customers
  - customerNum → customer ID
Naming Specific Types of Data
Naming Loop Indices

- Loop indices
  - Can use conventional i, j, k, etc. if:
    - Short, simple loop (no nesting)
    - AND
    - Variable has meaning only inside loop
  - Choose more descriptive name if any of these are true:
    - Variable has meaning outside loop
    - Loop body is longer than a few lines
      - Can forget what i stands for
      - Worse, could forget you used i already!
    - Avoids confusion
      - Easy to mix up i and j as indices into 2D array
        - Example: score[teamIndex][eventIndex] better than score[i][j]
Naming Status Variables

- **Status variables**
  - Don’t use *flag*
    - Use something more descriptive
      - E.g., dataReady, characterType, etc.
  - Set values that are either:
    - Named constants
    - Macros/defines
    - Enumeration values
      - E.g., characterType = CONTROL_CHARACTER, not characterType = 0x80
  - *Litmus test*: Do you have to “figure out” your own code?
Naming Temporary Variables

Temporary variables

- Used for:
  - Intermediate results
  - Temporary placeholders
  - Housekeeping values

- Using names like tmp, x, etc. → may imply you don’t fully understand the problem

- In general, try to make variables “non-temporary” → identify their actual purpose

**Example:**

- discriminant = sqrt(b^2 – 4*a*c)
- ...instead of...
- temp = sqrt(b^2 – 4*a*c)
Naming Boolean Variables

- **Boolean variables**
  - Can use typical names if appropriate:
    - done, error, found, success, ok
  - Use names that imply true or false
    - done → is it or is it not done?
    - status → not as clear
    - statusOK → much better
  - The “Is” prefix technique
    - Prefix variable with “is”
    - **Pros:**
      - Makes variable a question: isDone? isError?
      - Makes bad variable names apparent: isStatus?
    - **Cons:**
      - Can make logical statements less readable
        - “if ( isFound )” vs. “if ( found )”
  - Use positive names
    - Avoid notFound, notdone, etc.
Naming Enumerated Types

- Enumerated Types
  - Capitalization?
    - "Enums = user-defined type → format like other user-defined types (e.g., classes)"
    - *Example*: Color_Red
    - OR
    - "Enums = constants → format like constants"
    - *Example*: COLOR_RED
  - Prefixes?
    - Can add prefix to identify what enum the constant is a part of
      - *Examples*: Color_Red, Color_Green, etc.
    - If you can name the enum → can avoid prefix
      - *Examples*: Color.Red, Color.Green, etc.
Naming Constants

- **Constants**
  - Name the abstract entity the constant represents
    - NOT the number the constant refers to!
  - *Examples:*
    - THIRTEEN $\rightarrow$ terrible, even if it does equal 13
    - BAKERS_DOZEN $\rightarrow$ again, poor name
    - DONUTS_MAX $\rightarrow$ good name
A Word On Conventions
When to Have Conventions

- **If:**
  - Your project has multiple people working on it
  - Your code will be reviewed/maintained by someone else
  - Your code is very large
  - Your code is something future you might revisit
  - You have a lot of project-specific terms that you want a standard for

- ...you should have a convention for naming things

- How formal this is depends on the size of the project
Why to Have Conventions

Why?

- Allows you to take stuff for granted
- Can understand unfamiliar variables
- Code is more consistent and easier to learn/understand
- Reduces name proliferation (e.g., pointsTotal vs. totalPoints)
- Compensates for language weaknesses
  - I.e., differentiate data in ways that are not supported by language
- Emphasizes relationships among related items
  - E.g., employeeName, employeePhone, etc.
Things to Decide

- You should (at least in your own mind) distinguish between:
  - Variable names vs. Routine names
    - First letter capitalization?
  - Classes vs. Objects
    - First letter capitalization?
      - Works in case-sensitive language
      - Too close to each other?
    - Prefix? ("t_" for types OR "a" for variables?)
      - Works, but arguably hideous
  - Constants vs. Everything else
    - All caps?

- For multiword names, use either:
  - Capitalization of every word
  - Separate words with underscores "_"
- Either is fine; just be consistent (don’t mix them)
Short Names
Why Do We Have Short Names?

- Why have we traditionally had short names?
  - **Olden Days of Computing**
    - Older languages (e.g., Fortan) had character limits on names (2-8 chars)
  - Mathematics

- In modern languages, no real need to shorten names
If you do decide to abbreviate a variable name:

- Be consistent
  - Num vs. No

- Create names you can pronounce ("Telephone test")
  - “xPos” rather than “xPstn”

- Avoid misleading abbreviations
  - “End of B” → “ENDB”, not “BEND”
Names To Avoid

HEAD KNIGHT: DON'T SAY THAT WORD.

ARTHUR: WHAT WORD?

HEAD KNIGHT: I CANNOT TELL, SUFFICE TO SAY IS ONE OF THE WORDS THE KNIGHTS OF NI CANNOT HEAR.

ARTHUR: HOW CAN WE NOT SAY THE WORD IF YOU DON'T TELL US WHAT IT IS?

KNIGHTS OF NI: AAAAAUGH!

HEAD KNIGHT: YOU SAID IT AGAIN!

-- MONTY PYTHON AND THE HOLY GRAIL
Things to Avoid

- Avoid:
  - Misleading names and abbreviations
    - “FALSE” = “Fig and Almond Season”
  - Names with similar meanings
    - “input” and “inputValue” both in same code
  - Variables with different meanings but similar names
    - BAD: “clientRecs” vs. “clientReps”
    - BETTER: “clientRecords” vs. “clientReports”
  - Names that sound similar
    - “wrap” vs. “rap”
Things to Avoid

- Avoid:
  - Numerals in names (unless really does have inherent meaning)
    - “file1” and “file2” → in general, should be array “file[]”
  - Misspelled words in names
    - “hilite” vs. “highlight”
    - Good idea to avoid commonly misspelled words as well
  - Differentiating variables strictly on capitalization
    - “frd” vs. “FRD” vs. “Frd”
  - Multiple language differences
    - Settle on “color” or “colour”
Avoid:

- Names of standard types, variables, or routines
  - Most compilers will stop you, but some languages will let you get away with it, so avoid it!

- Totally unrelated names
  - Favorite sports team, cartoon characters, profanity, etc.

- Names with hard to read characters
  - *Example:* “one” vs. “lower-case L” vs. “upper-case I”