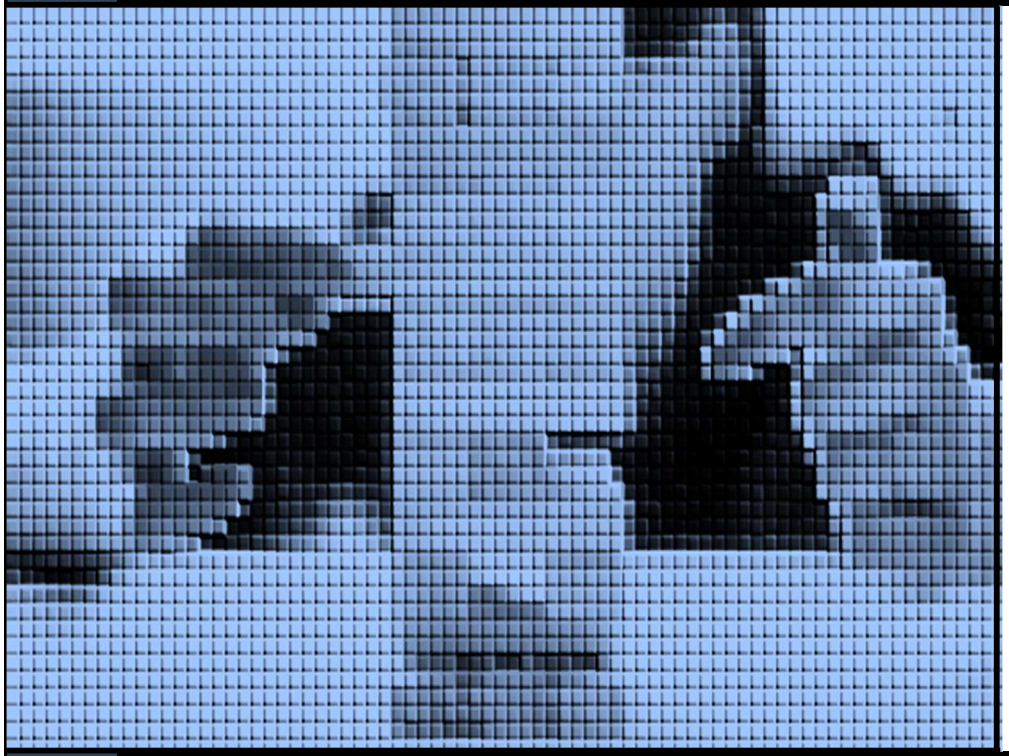
University of BRISTOL

Department of Computer Science

COMS10016: Imperative and Functional Programming



C Programming Style Guide

Tilo Burghardt

C Programming Style Guide

- The following slides provide a brief programming style guide for the COMS10016 unit.
- *You should adhere to it* whenever you write code, be that in the labs or when you write/submit coursework.
- Many of the style rules presented are universal, although some differ from other style guides.
- Most companies have their own in-house style guide and learning to adhere to a particular set of style rules when learning C is important.
- In any case, a style guide will help you write clear and readable programs in C.

PROGRAM DESIGN & READABILITY

- YOU SHOULD keep functions short, say around 20 lines maximum.
- YOU SHOULD implement unit testing for all functions that perform calculations.
- YOU SHOULD limit the length of a line of code, say less than 60 characters so that no scrolling is required for terminal editors etc.
- YOU SHOULD if programs become very large (e.g. 1000s of LoCs) use prototypes/definitions first, then `main`() followed by the function implementations. The reader then always knows to find `main`() near the top of the file. Alternatively, the `main`() procedure has to be last.
- USE DRY CODE (Don't Repeat Yourself). Never cut-and-paste large chunks of code and then make minor changes. Instead, make a function and pass arguments to implement variability.

DO NOT USE

- DO NOT USE global variables at (almost) any cost.
- DO NOT USE any of the keywords `continue`, `goto` or `break`. (The one exception is inside `switch`, where `break` is allowed because it is essential. Otherwise, try to rewrite code into functions, which can have multiple `return`s in them.)
- DO NOT USE `magic numbers' like `if (x<37)...` There should be no inexplicable numbers in your code. Instead, `#define` them or use `const` with a meaningful name.
- DO NOT USE stdout as the target for error outputs when exiting your program in an error state, make sure that you `fprintf` the error on stderr and you use `exit`.

DO USE

- DO USE all warnings in the compiler flags and make sure your code compiles warning-free.
- YOU MUST USE matching `fopen` and `fclose` calls, as well as matching `malloc` and `free` calls. This is essential to avoid memory and resource leaks.
- DO USE meaningful identifiers.
- DO USE consistent indentation. We recommend 2 spaces rather than tabs, opening curly braces on the line of the associated statement and closing them on a new line.
- DO USE `typedef`s, `enum`s and `struct`s for readability.
- DO USE the values that are returned by functions.

NAMING and COMMENTS

- USE succinct identifiers in camelCode starting with a lower case character for local variables, function names, and function arguments.
- USE succinct identifiers in camelCode (or CamelCode starting with an upper case character if you want to make them distinct) for naming user `typedef`s and `struct`s.
- USE succinct identifiers in CAPITALS or Capital for all globally `#define`d constants, global `const`s, and `enum`erated constants.
- USE comments // or /*…*/ before functions and at important points in the code. Make sure comments are brief, and non-trivial.

READABILITY & MISC

- Only write a *single statement per line*, unless you are using a simple `if else` statement where a single statement can follow the `if` or `else` statement.
- Use white space around operators, after commas, and between procedures.
- Keep expressions *small* so they are easy to understand, to check, and to test.
- Use redundant parentheses to make expressions *clearer*.
- *Avoid* using the underscore character _ at the start of identifiers and limit their size to 31 characters to prevent conflicts and help portability.