

Code Conventions and Documentation

Module Code: ELEE1147

Module Name: Programming for Engineers

Credits: 15

Module Leader: Seb Blair BEng(H) PGCAP MIET MIHEEM FHEA

Naming Conventions

- Lower case `lowercase` : `publicdomiansoftware`
 - elements and attributes
- Upper case `UPPERCASE` : `PUBLICDOMAINSOFTWARE`
 - Naming constants
- Camel Case `camelCase` : `publicDomainSoftware`
 - local variable names
- Pascal Case `PascalCase` : `PublicDomainSoftware`
- Snake Case `snake_case` : `public_domain_software`
 - C/C++ standard library names
- Screaming Snake Case `SCREAMING_SNAKE_CASE` : `PUBLIC_DOMAIN_SOFTWARE`
 - Naming Constants
- Kebab Case `kebab-case` : `public-domain-software`
 - class names, ids
- Screaming Kebab Case `SCREAMING-KEBAB-CASE` : `PUBLIC-DOMAIN-SOFTWARE`
 - Macros

VS C Convention

```
#include <stdio.h>

// Macros
#define MAX(a, b) ((a) > (b) ? (a) : (b))
#define MIN(a, b) ((a) < (b) ? (a) : (b))

// Global variables
int globalVariableOne;
int globalVariableTwo;

// Function prototypes
void InitializeGlobals();
int AddNumbers(int a, int b);

int main() {
    // Local variables
    int localVariable;

    // Initialize global variables
    InitializeGlobals();

    // Assign values to local variables
    localVariable = AddNumbers(globalVariableOne, globalVariableTwo);

    // Using macros
    printf("Max: %d\n", Max(globalVariableOne, localVariable));
    printf("Min: %d\n", Min(globalVariableTwo, localVariable));

    return 0;
}
...
```

```
...
// Function definitions
void InitializeGlobals() {
    globalVariableOne = 5;
    globalVariableTwo = 10;
}

int AddNumbers(int a, int b) {
    // This comment explains the function behavior
    return a + b;
}
```

Other conventions

GNU C:

- Naming: Typically follows the lowercase with underscores for variables and functions (e.g., `my_variable`, `my_function()`).
- Indentation: Uses spaces for indentation (often 2 or 4 spaces).
- Brace Style: Opening braces are usually on the same line as the statement, following the Kernighan and Ritchie style.

GCC (GNU Compiler Collection):

- Similar to the GNU C conventions.
- It may include additional guidelines for contributing to the GCC codebase.

Other conventions

LLVM:

- Naming: Uses camelCase for function names and lowercase with underscores for variable names (e.g., `myVariable`, `my_function()`).
- Indentation: Typically 2 spaces.
- Brace Style: Opening braces are on the same line.

Microsoft Visual Studio C++:

- Naming: Uses PascalCase for function and method names, and camelCase for variable names (e.g., `MyFunction()`, `myVariable`).
- Indentation: Typically 4 spaces.
- Brace Style: Opening braces are on the same line.

Other conventions

Google C++ Style Guide:

- Naming: Uses camelCase for variable names, and underscores for function names (e.g., `myVariable`, `my_function()`).
- Indentation: Typically 2 spaces.
- Brace Style: Opening braces are on the same line.

Mozilla C++ Coding Style:

- Naming: Uses camelCase for variable names and function parameters, and PascalCase for function names (e.g., `myVariable`, `MyFunction()`).
- Indentation: Typically 2 spaces.
- Brace Style: Opening braces are on the same line.

Other conventions

Linux Kernel Coding Style:

- Naming: Uses lowercase with underscores for variables and functions (e.g., `my_variable` , `my_function()`).
- Indentation: Typically 8 spaces.
- Brace Style: Opening braces are on the same line.

Qt Coding Style:

- Naming: Uses camelCase for variables and functions (e.g., `myVariable` , `myFunction()`).
- Indentation: Typically 4 spaces.
- Brace Style: Opening braces are on the same line.

Documentation, 'doc as you go...'

Why Documentation

- **You**
 - put down the project and return to it much later
 - want people to use it and give you credit
- **Others**
 - would be encouraged to contribute
 - more easily use your code
- **Science / Engineering**
 - Advances
 - Open collaboration
 - Reproducibility and transparency



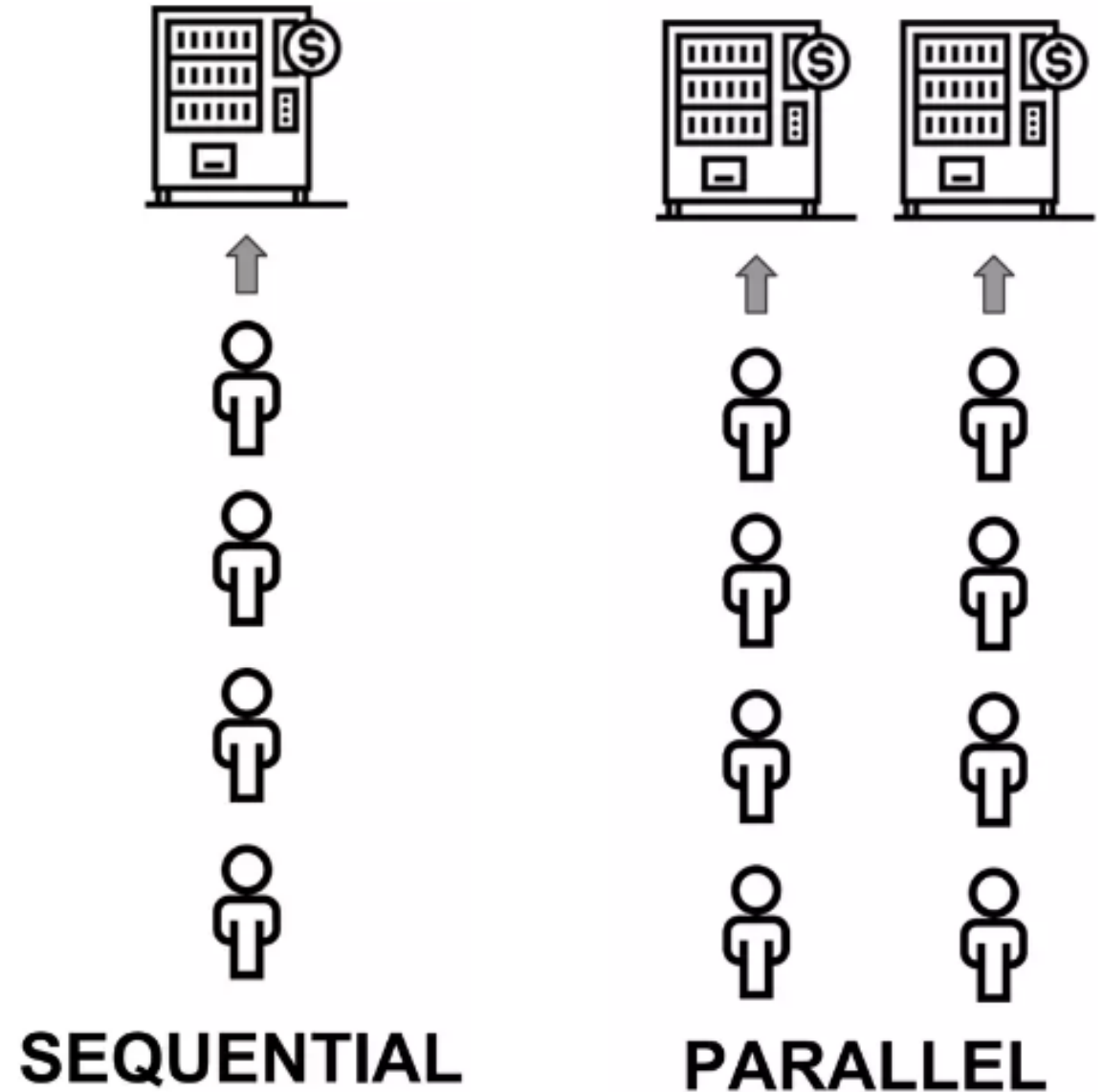
Tools for Documentantion

- **Python**
 - Sphinx, Doctest, Numpydoc
- **R**
 - R Markdown, Kite
- **C++**
 - BoostBook, QuickBook, GhostDoc
- **Java**
 - Javadoc
- **Ruby**
 - Docurium
- **Doxygen**
 - r, C, C#, PHP, Java, Python, and Fortran.



Divergence Dilemma

- As with all documentation code develops faster and is released, thus creates a divergence, as in code \leftrightarrow documentation become out of sync.



Literate Programming

- a computer program is given as an explanation of how it works in a natural language, such as English, interspersed (embedded) with snippets of macros and traditional source code, from which compilable source code can be generated.

The logo for doxygen, featuring the word "doxygen" in a blue, stylized font with a thick blue underline.

Doxygen syntax

```
/**
 * @file calculator.c
 * @brief Simple calculator program with basic operations.
 */

#include <stdio.h>

/**
 * @brief Adds two numbers.
 * @param a The first operand.
 * @param b The second operand.
 * @return The sum of a and b.
 */
int add(int a, int b) {
    return a + b;
}

/**
 * @brief Subtracts two numbers.
 * @param a The first operand.
 * @param b The second operand.
 * @return The result of subtracting b from a.
 */
int subtract(int a, int b) {
    return a - b;
}
```

```
/**
 * @brief Main function to demonstrate calculator operations.
 * @return 0 if successful, otherwise an error code.
 */
int main() {
    int num1, num2;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    printf("Sum: %d\n", add(num1, num2));
    printf("Difference: %d\n", subtract(num1, num2));

    return 0;
}
```

Example output

Calculator Documentation 1.0

A simple calculator program with basic operations.

[Main Page](#) [Files](#)

calculator.c File Reference

Simple calculator program with basic operations. [More...](#)

```
#include <stdio.h>
```

Functions

int	add (int a, int b)	Adds two numbers.
int	subtract (int a, int b)	Subtracts two numbers.
int	main ()	Main function to demonstrate calculator operations.

Detailed Description

Simple calculator program with basic operations.

Function Documentation

◆ add()

```
int add ( int a,  
         int b  
         )
```

Adds two numbers.

Parameters

- a** The first operand.
- b** The second operand.

Returns

The sum of a and b.

◆ main()

```
int main ( )
```

Main function to demonstrate calculator operations.

Returns

0 if successful, otherwise an error code.

Doxygen Configuration file

```
PM> doxygen.exe doxygenConfigFile
```

```
# Doxyfile for calculator.c

DOXYFILE_ENCODING      = UTF-8
PROJECT_NAME           = "Calculator Documentation"
PROJECT_NUMBER         = 1.0
PROJECT_BRIEF          = "A simple calculator program with basic operations."

OUTPUT_DIRECTORY      = ./docs
CREATE_SUBDIRS         = NO

INPUT                  = calculator.c
RECURSIVE              = NO

EXTRACT_ALL            = YES
EXTRACT_PRIVATE        = YES
EXTRACT_STATIC         = YES
EXTRACT_LOCAL_CLASSES  = YES

GENERATE_LATEX          = NO
GENERATE_HTML          = YES
```

Python Documentation Generators

- Sphinx
 - Python, Linux Kernel and Project Jupyter
- MkDocs
 - a fast, simple and downright gorgeous static site generator that's geared towards building project documentation.
- Doxygen
- Pydoc
- Pydoctor

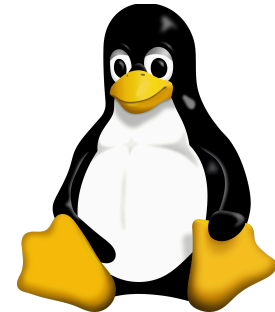
pydoc (builtin)

```
def add_numbers(a, b):  
    """  
    Adds two numbers together and returns the result.  
  
    Parameters:  
    a (int): The first number.  
    b (int): The second number.  
  
    Returns:  
    int: The sum of the two numbers.  
    """  
    return a + b
```

```
Git\tmp\python via v3.12.8  
> python -m pydoc add  
Help on module add:  
  
NAME  
    add  
  
FUNCTIONS  
    add_numbers(a, b)  
        Adds two numbers together and returns the result.  
  
        Parameters:  
        a (int): The first number.  
        b (int): The second number.  
  
        Returns:  
        int: The sum of the two numbers.
```


Sphinx

- GitHub link -> <https://github.com/sphinx-doc/sphinx>
- Webpage -> <https://www.sphinx-doc.org/en/master/>
- is a third-party tool
- Preferred tool for Python, Linux Kernel and Project Jupyter



Sphinx

add.py

```
def add_numbers(a, b):  
    """  
    Adds two numbers together and returns the result.  
  
    :param a: The first number.  
    :type a: int  
    :param b: The second number.  
    :type b: int  
  
    returns: The sum of the two numbers.  
    rtype: int  
    """  
    return a + b
```

conf.py

```
project = 'adding'  
copyright = '2025, Seb Blair'  
author = 'Seb Blair'  
release = '0.1'  
  
extensions = []  
  
templates_path = ['_templates']  
exclude_patterns = []  
  
html_theme = 'alabaster'  
html_static_path = ['_static']
```