C Coding Standard

EECS 678 – Operating Systems

Summary

- Coding standard is a set of good coding practices which help in writing legible code

- Can be broadly divided into two parts
  - Rules
    - Clauses of coding standard which should always be followed
  - Guidelines
    - Clauses of coding standard which should be adhered to unless a deviation can be justified

- In EECS-678, conformance with coding standard can earn you up-to 30% extra credit

Coding Standards for C-Language

- MISRA C is a widely accepted coding standard for C language which is generally used in safety critical systems
  - https://www.misra.org.uk

- Other standards also exist (which are openly available)
  - GNU Coding Standard
    - https://www.gnu.org/prep/standards/standards.html#Writing-C
  - Linux Kernel Coding Standard
    - https://www.kernel.org/doc/Documentation/process/coding-style.rst

EECS-678 Coding Standard - Rules

- Rule-1
  - Line width should not exceed 78 characters (columns)
- Rule-2
  - Code should be properly indented. Indentation should be done using tabs (instead of spaces)
- Rule-3
  - Tab space should be set to 8 characters (columns)

EECS-678 Coding Standard - Rules

- Example

Bad Code

```
int max (int x, int y)
{
  /* Assume y is greater of the x and y */
    int max_value = y;

  /* Update the maximum value if x is greater than y */
    if (x > y) {
    max_value = x;
}

return max_value;
}
~
```

Good Code

```
int max (int x, int y)
{
        /* Assume y is greater of the x and y */
        int max_value = y;

        /* Update the maximum value if x is greater than y */
        if (x > y) {
                max_value = x;
        }

        /* Return the maximum value to caller */
        return max_value;
}
```

EECS-678 Coding Standard - Rules

- Rule-4
  - Variables should be declared using meaningful names. Underscores should be used to separate name components

- Example
  - `int period_in_ms = 500;`

- Rule-5
  - Global data should be declared at the start of the source files

EECS-678 Coding Standard - Rules

- Rule-6
  - Macros and global constants should be declared using all capital letters

- Example
  - `#define MAX_FREQUENCY_HZ 1000`
  - `const int THREAD_COUNT = 4;`

- Rule-7
  - Inside functions and code blocks, variable declaration should be done at the beginning only

EECS-678 Coding Standard - Rules

- Rule-8
  - Header files should contain guard macros (to protect against circular inclusion)

- Example

```
#ifndef __MY_HEADER__
#define __MY_HEADER__
…
#endif /* __MY_HEADER__ */
```

EECS-678 Coding Standard - Rules

- Rule-9
  - Code must contain meaningful comments

- Example

Bad Code

```
int max_element (unsigned int *array, int length)
{
        unsigned int max_value = 0;
        for (i = 0; i < length; ++i) {
                if (array[i] > max_value) {
                        max_value = array[i];
                }
        }
        return max_value;
}
```

Good Code

```
int max_element (unsigned int *array, int length)
{
        /* Declare a variable to track the maximum value */
        unsigned int max_value = 0;

        /* Iterate over the array to update maximum value */
        for (i = 0; i < length; ++i) {
                /* Check if the current array element is greater than the
                   current max value */
                if (array[i] > max_value) {
                        /* New max value found */
                        max_value = array[i];
                }
        }

        return max_value;
}
```

EECS-678 Coding Standard - Guidelines

- Clause-1
  - Function definitions should contain descriptive headers which follow a uniform format

- Example

```
/*
 * max_element
 *
 * Description  : Function to determine the maximum value among the elements
 *                of an unsigned integer array
 * @array       : Pointer to the input array of unsigned integers
 * @length      : Integer value specifying the length of the array
 *
 * $int         : Return integer specifying the maximum value of array elements
 */
int max_element (unsigned int *array, int length)
{
        /* Declare a variable to track the maximum value */
        unsigned int max_value = 0;

        /* Iterate over the array to update maximum value */
        for (i = 0; i < length; ++i) {
                /* Check if the current array element is greater than the
                   current max value */
                if (array[i] > max_value) {
                        /* New max value found */
                        max_value = array[i];
                }
        }

        return max_value;
}
```

EECS-678 Coding Standard - Guidelines

- Clause-2
  - During code development, macro usage should be avoided as much as possible
    - Constant values should be declared using **enums** or global constant data types
    - Functions should be used instead of function like macros
- Clause-3
  - **Switch-Case** statement should be used in place of long of **if-else if** chains
- Caluse-4
  - Function prototypes should always be declared (either in header files or at the beginning of source files) before function definitions and usage

EECS-678 Coding Standard - Guidelines

- Caluse-5
  - In declaration of pointer variables, asterisk should be placed closer to the variable name (instead of type name)

- Example
  - `int    *x;`
  - `Int    max_element (int *array, int length) { … }`

EECS-678 Coding Standard - Guidelines

- Clause-6
  - Usage of **goto** and **continue** keywords should be avoided as much as possible
- Clause-7
  - Multiple return statements from a single function should be avoided
- Clause-8
  - All the **cases** in **switch-case** statements should be properly terminated. Fall through cases should not be deliberately used

EECS-678 Coding Standard - Guidelines

- Clause-9
  - Magic numbers should not be used – unless their usage is absolutely obvious
- Clause-10
  - In comparison statements, constant value should be placed on the left side of the comparison
- Example
  - `if ( 0 == x) { … }` `/* Desirable */`
  - `If ( x == 0) { … }` `/* Undesirable */`

EECS-678 Coding Standard - Guidelines

- Clause-11
  - Dynamically allocated memory must be freed. Code must be checked for memory leaks (using valgrind)

- Clause-12
  - All compiler reported warnings must be resolved

The Zen of Coding

“Beautiful is better than ugly.
Simple is better than complex.
Flat is better than nested.
Readability counts.
Although practicality beats purity.
Unless explicitly silenced.
Now is better than never.

Explicit is better than implicit.
Complex is better than complicated.
Sparse is better than dense.
Special cases aren't special enough to break the rules.
Errors should never pass silently.
In the face of ambiguity, refuse the temptation to guess.
*Although never is often better than *right* now.*

There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.” [1]

[1] Copied from the *Zen of Python by Tim Peters. [https://www.python.org/dev/peps/pep-0020/]*