# Software standards

## Naming

- `const` and `#define` statements use all caps with underscores when spaces are nescessary.
- Anything part of a class must be in camel case.
- Outside of classes, use lowercase with underscores for naming

```
#define HELLO_WORLD 1
const int NUMBER = 72;

class helloWorldClass {
public:
  int helloWorld;

  // NOTE: passed vars in class' functions are lowercase,
  // since they're not strictly stored in the class
  int functionOne(int variable_one);
};

int function_two(int variable_two);
helloWorldClass hello_world;
```

## Whitespace

- Standard libraries are included and grouped before other includes
- Use spaces not tabs (4 as a standard).
- Commas must have spaces after them.
- No space between function and opening parenthesis.
- Space after semicolon if code continues on the same line.
- Conditional calls (eg else after if) on a new line.
- Do not nest question mark statements.
- Values for defines must be aligned in their code blocks/

```
// Showing the incldue rule
#include <stdio.h>
#include <stdint.h>

#include "someCutomLib.h"

// This shows the proper alignment of define macro values
#define HELLO_WORLD                                      22
#define HELLO_MY_BABY_HELLO_MY_HONEY_HELLO_MY_RAGTIME_GAL 420
```

```
// The block below shows examples for proper spacing of commas and semicolons
int a; bool b;
int a, b, c = 0;

// This is an example of function calls, builtin calls,
// as well as spacing for conditionals
if (some_test) {
  function_two(intNum);
}
else {
  hello_world.functionOne(intNum);
};
```

## Operators

- Brackets for the condition of the question mark operator
- Use of question mark operator stays on one line. If the line becomes too long, use if/else instead.
- Operators require spaces on either side.
- The block comment operators require their own lines.
- Comments require a new line before them, as well as spaces after comment operator.

```
/*
This shows the proper usage of a block comment
*/
if (0) {
  a = (1 + 2 / (1 * 72));
}
else (b = (2 == c) ? 1 : 2);

// This and the previos lines show proper usage of operator spacing,
// question mark usage and commenting
```

## Brackets

- No padding between parenthesis.
- Short functions (e.g. get and set), can have incline curly brackets. Padding between the def and the backets must be used.
- Builtins use inline curly brackets, function defs use newline curly brackets

```
int some_function_name(void) {return someVar}

int function(long num)
```

```
{
  // function definition here
}

function(number + 1) // No space betwuen parentheses

for (int i = 0; i < 10000000000; i++) {  // Inline curly bracket
}
```

## Classes, structures

- Each variable inside a struct must have its own line when using dot notation
- If using dot notation for structs, it can be inline
- Public and private and protected are on the same indentation as the class
- Classes and structures have curly brackets inline

```
class someClassHere {
public:
  //stuff
private:
  //stuff
protected:
  //stuff
};

// Showing correct curly brackets here
typedef struct new_struct {
  int i;
  int j;
  int k;
} new_struct;

// Structure example with dot notation
strct new_strct = {
    .i = 1;
    .j = 2;
    .k = 3;
};

// Struct example without dot notation
strct new_strct = {1, 2, 3};
```