1 Rules

1.1 General

Use no accented letters or extended ASCII characters. Everything has to be written in plain english. No code-warnings are admitted.

1.2 Names

Use capitalization instead of underscores. Use the following Hungarian-like notation for prefixes:

t	struct	str	string
cl	class	ch	char
h	handle	b	boolean
i	integer	a	array
f	float	aa	array of array
d	double		

Give classes, variables, functions, etc. descriptive, differentiable names. Things that are related should have similar names, while things that are not should not. Keep the names up-to-date. If functionality changes, update the name.

Notice: if you call your variables a, b, c, then it will be impossible to search for instances of them using a simple text editor. The same for misspellings: by misspelling in some function and variable names, and spelling it correctly in others (such as SetPintleOpening SetPintalClosing) you avoid the use of effective text-search techniques.

1.3 Spacing

Here there is famous haiku on the consistency of spacing:

> Anybody who mixes tabs and spaces
> for indentation
> will spend an eternity burning in hell.

Most important rule: BE CONSISTENT. Other rules:

1. identation must be performed using only tabs;
2. tab width in the editors must be of 4 spaces;
3. always place spaces:
   (a) after any commas;
   (b) on both sides of any binary arithmetic or logical operators *other than multiplication or division*;
4. expand code over multiple lines and make matching parenthesis appear in the same column.

Thus write `f( a, b, c )` instead of `f(a,b,c)`.

1.4 Functions

1. write functions that fit on one screen;
2. keep block nesting to a minimum;
3. think of those that come after you;
4. use less code: the less code you have, the less there is to maintain and to fix. Get rid of unused functions / code;
5. when possible, write error-checking software.

1.4.1 Comments

Always provide

1. a few sentences before the procedure/function saying what it does;
2. a description of the values being passed into it;
3. a description of what it possibly returns;
4. inside the function, comments that split the code up into shorter tasks;
5. for chunks of code that seem thorny, a quick explanation of what is happening.

Comment like a smart person. Bad examples:

```
% Now we increase Number_of_sensors by one.
Number_of_sensors = Number_of_sensors + 1;

function Output = XXX(Inputs)
... zillions of code rows without comments ...
return Output
% so we are done
```

2 General tips

Hard-coding = EVIL. Moreove, citing Donald Knuth, "Premature optimization is the root of all evil.". Simple code is faster to write, faster to understand when you return to it later, and faster to debug. Write programs like stereotypical old grannies drive.

3 Bugs management

Implement and maintain a bugs-database with entries:

1. complete steps to reproduce the bug;
2. expected behavior;
3. observed (buggy) behavior;
4. who it's assigned to;
5. whether it has been fixed or not.

Note: highest priority is to eliminate bugs before writing any new code.