CODING STANDARDS:

Limited use of globals:
These rules tell about which types of data that can be declared global and the data that can't be.

Standard headers for different modules:
For better understanding and maintenance of the code, the header of different modules should follow some standard format and information.

- Name of the module
- Date of module creation
- Author of the module
- Modification history
- Synopsis of the module about what the module does
- Different functions supported in the module along with their input output parameters
- Global variables accessed or modified by the module

Naming conventions for local variables, global variables, constants and functions:
Some of the naming conventions are given below:

- Meaningful and understandable variables name helps anyone to understand the reason of using it.
- Local variables should be named using camel case lettering starting with small letter (e.g. **localData**) whereas Global variables names should start with a capital letter (e.g. **GlobalData**). Constant names should be formed using capital letters only (e.g. **CONSDATA**).
- It is better to avoid the use of digits in variable names.
- The names of the function should be written in camel case starting with small letters.
- The name of the function must describe the reason of using the function clearly and briefly.

Indentation:
Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly. Some of the spacing conventions are given below:

- There must be a space after giving a comma between two function arguments.
- Each nested block should be properly indented and spaced.

- Proper Indentation should be there at the beginning and at the end of each block in the program.
- All braces should start from a new line and the code following the end of braces also start from a new line.

Error return values and exception handling conventions:
All functions that encountering an error condition should either return a 0 or 1 for simplifying the debugging.

Avoid using a coding style that is too difficult to understand:
Code should be easily understandable. The complex code makes maintenance and debugging difficult and expensive.

Avoid using an identifier for multiple purposes:
Each variable should be given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.

Code should be well documented:
The code should be properly commented for understanding easily. Comments regarding the statements increase the understandability of the code.

Length of functions should not be very large:
Lengthy functions are very difficult to understand. That’s why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.

Try not to use GOTO statement:
GOTO statement makes the program unstructured, thus it reduces the understandability of the program and also debugging becomes difficult.