Coding

Coding

The objective of the coding phase is to transform the design of a system into code in a high level language and then to unit test this code. The programmers adhere to standard and well defined style of coding which they call their coding standard. The main advantages of adhering to a standard style of coding are as follows:

- A coding standard gives uniform appearances to the code written by different engineers.
- It facilitates code of understanding.
- Promotes good programming practices.

For implementing our design into a code, we require a good high level language. A programming language should have the following features:

Characteristics of a Programming Language

Readability: A good high-level language will allow programs to be written in some ways that resemble a quite-English description of the underlying algorithms. If care is taken, the coding may be done in a way that is essentially self-documenting.

Portability: High-level languages, being essentially machine independent, should be able to develop portable software.

Generality: Most high-level languages allow the writing of a wide variety of programs, thus relieving the programmer of the need to become expert in many diverse languages.

Brevity: Language should have the ability to implement the algorithm with less amount of code. Programs expressed in high-level languages are often considerably shorter than their low-level equivalents.

Error checking: Being human, a programmer is likely to make many mistakes in the development of a computer program. Many high-level languages enforce a great deal of error checking both at compile-time and at run-time.

Cost: The ultimate cost of a programming language is a function of many of its characteristics.

Characteristics of a Programming Language

Familiar notation: A language should have familiar notation, so it can be understood by most of the programmers.

Quick translation: It should admit quick translation.

Efficiency: It should permit the generation of efficient object code.

Modularity: It is desirable that programs can be developed in the language as a collection of separately compiled modules, with appropriate mechanisms for ensuring self-consistency between these modules.

Widely available: Language should be widely available and it should be possible to provide translators for all the major machines and for all the major operating systems.

Coding Standards and Guidelines

Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop. The following are some representative coding standards.

1. Rules for limiting the use of global: These rules list what types of data can be declared global and what cannot.
2. Contents of the headers preceding codes for different modules: The information contained in the headers of different modules should be standard for an organization. The exact format in which the header information is organized in the header can also be specified.

The following are some standard header data:

• Name of the module. • Date on which the module was created. • Author's name. • Modification history. • Synopsis of the module. • Different functions supported, along with their input/output parameters. • Global variables accessed/modified by the module.

Coding Standards and Guidelines

3. Naming conventions for global variables, local variables, and constant identifiers: A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

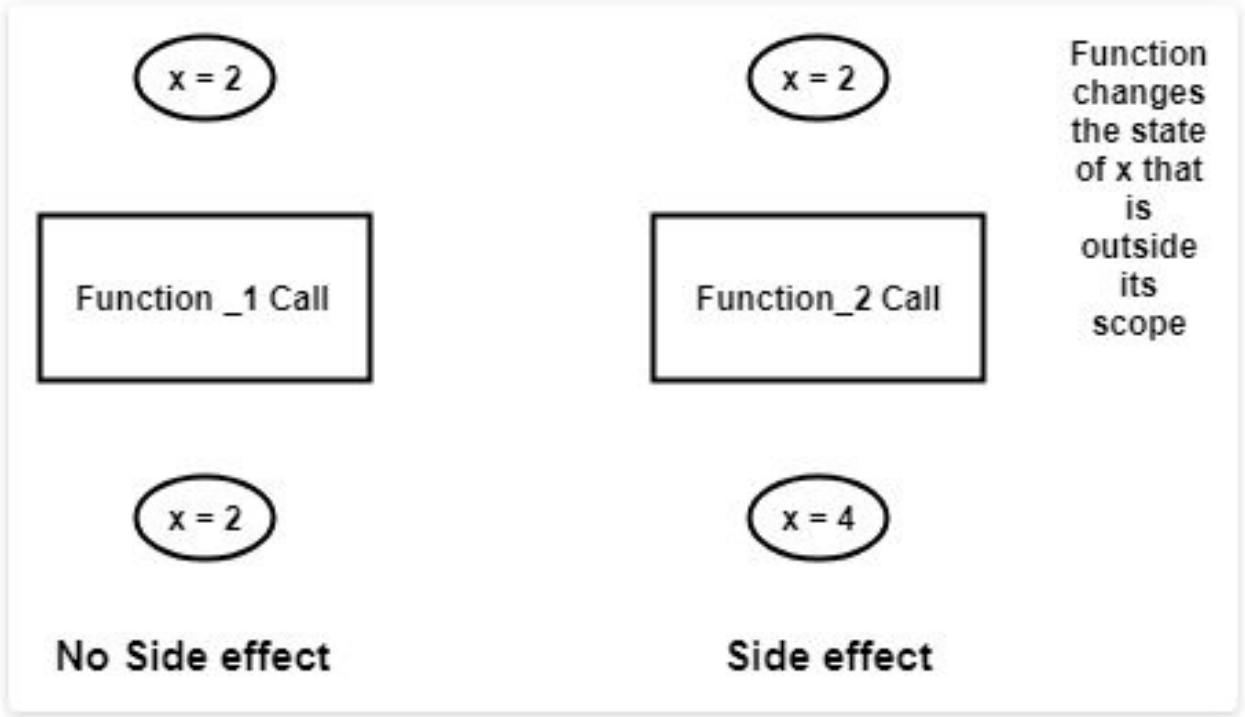
4. Error return conventions and exception handling mechanisms: The way error conditions are reported by different functions in a program are handled should be standard within an organization. For example, different functions while encountering an error condition should either return a 0 or 1 consistently.

Coding Standards and Guidelines

The following are some representative coding guidelines recommended by many software development organizations.

1. Do not use a coding style that is too clever or too difficult to understand: Code should be easy to understand. Many inexperienced engineers actually take pride in writing cryptic and incomprehensible code. Clever coding can obscure meaning of the code and hamper understanding. It also makes maintenance difficult.
2. Avoid obscure side effects: The side effects of a function call include modification of parameters passed by reference, modification of global variables, and I/O operations. An obscure side effect is one that is not obvious from a casual examination of the code. Obscure side effects make it difficult to understand a piece of code. For example, if a global variable is changed obscurely in a called module or some file I/O is performed which is difficult to infer from the function's name and header information, it becomes difficult for anybody trying to understand the code.

Coding Standards and Guidelines



Modifying a non-local variable:

```
1  x=3
2  def square(x):
3      x=x*x
4      return x
5  def square_side_effect():
6      global x
7      x=x*x
8      return x
9  print(x)                    # 3
10 print(square(x))            # 9
11 print(x)                    # 3
12 print(square_side_effect())     # 9
13 print(x)                    # 9
```

Figures: Examples of obscure side effects

Coding Standards and Guidelines

3. Do not use an identifier for multiple purposes: Programmers often use the same identifier to denote several temporary entities. For example, some programmers use a temporary loop variable for computing and a storing the final result. The rationale that is usually given by these programmers for such multiple uses of variables is memory efficiency, e.g. three variables use up three memory locations, whereas the same variable used in three different ways uses just one memory location.

However, there are several things wrong with this approach and hence should be avoided. Some of the problems caused by use of variables for multiple purposes as follows:

- Each variable should be given a descriptive name indicating its purpose. This is not possible if an identifier is used for multiple purposes. Use of a variable for multiple purposes can lead to confusion and make it difficult for somebody trying to read and understand the code.
- Use of variables for multiple purposes usually makes future enhancements more difficult.

Coding Standards and Guidelines

4. The code should be well-documented: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

5. The length of any function should not exceed 10 source lines: A function that is very lengthy is usually very difficult to understand as it probably carries out many different functions. For the same reason, lengthy functions are likely to have disproportionately larger number of bugs.

6. Avoid goto statements: Use of goto statements makes a program unstructured and very difficult to understand.