UNIT 4

CODING

SOFTWARE ENGINEERING

Coding

The coding is the process of transforming the design of a system into a computer language format. This coding phase of software development is concerned with software translating design specification into the source code. It is necessary to write source code & internal documentation so that conformance of the code to its specification can be easily verified.

Coding is done by the coder or programmers who are independent people than the designer. The goal is not to reduce the effort and cost of the coding phase, but to cut to the cost of a later stage. The cost of testing and maintenance can be significantly reduced with efficient coding.

Goals of Coding

1.To translate the design of system into a computer language format: The coding is the process of transforming the design of a system into a computer language format, which can be executed by a computer and that perform tasks as specified by the design of operation during the design phase.

2.To reduce the cost of later phases: The cost of testing and maintenance can be significantly reduced with efficient coding.

3.Making the program more readable: Program should be easy to read and understand. It increases code understanding having readability and understandability as a clear objective of the coding activity can itself help in producing more maintainable software.

Characteristics of Programming Language

Readability: A good high-level language will allow programs to be written in some methods that resemble a quite-English description of the underlying functions. The coding may be done in an essentially self-documenting way.

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

Generality: Most high-level languages allow the writing of a vast collection of programs, thus relieving the programmer of the need to develop into an expert in many diverse languages.

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

Error checking: A programmer is likely to make many errors in the development of a computer program. Many high-level languages invoke a lot of bugs checking both at compile-time and run-time.

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

Quick translation: It should permit quick translation.

Efficiency: It should authorize the creation of an efficient object code.

Modularity: It is desirable that programs can be developed in the language as several separately compiled modules, with the appropriate structure for ensuring self-consistency among these modules.

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

Standards and Guidelines.

Good software development organizations maintain some well-defined and standard style of coding called coding standards.. They usually make their own coding standards and guidelines depending on what suits their organization best and based on the types of software they develop.It is very important for the programmers to maintain the coding standards other wise the code may be rejected during code review.·

Purpose of Having Coding Standards:·

- A coding standard gives a uniform appearance to the codes written by different engineers..
- It improves readability, and maintainability of the code and it reduces complexity also.
- It helps in code reuse and helps to detect error easily.
- It promotes sound programming practices and increases efficiency of the programmers

A coding standard lists several rules to be followed during coding, such as the way variables are to be named, the way the code is to be laid out, error return conventions, etc.

Coding Standards

General coding standards refers to how the developer writes code, so here we will discuss some essential standards regardless of the programming language being used.

The following are some representative coding standards:

1.Indentation: Proper and consistent indentation is essential in producing easy to read and maintainable programs.

Indentation should be used to:

1. Emphasize the body of a control structure such as a loop or a select statement.
2. Emphasize the body of a conditional statement
3. Emphasize a new scope block

2.Inline comments: Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.

3.Rules for limiting the use of global: These rules file what types of data can be declared global and what cannot.

4.**Structured Programming:** Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain, except as outlined line in the FORTRAN Standards and Guidelines.

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

6.**Error return conventions and exception handling system:** Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently.

Coding Guidelines

General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain. Most of the examples use the C language syntax, but the guidelines can be tested to all languages.

1. **Line Length:** It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

2. Spacing: The appropriate use of spaces within a line of code can improve readability.
Example:

Bad:
```
cost=price+(price*sales_tax)
fprintf(stdout ,"The total cost is %5.2f\n",cost);
```

Better:
```
cost=price+(price*sales_tax)
fprintf (stdout,"The total cost is %5.2f\n",cost);
```

3. 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.

4. The length of any function should not exceed 10 source lines: A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.

5. Do not use goto statements: Use of goto statements makes a program unstructured and very tough to understand.

6. Inline Comments: Inline comments promote readability.

7. Error Messages: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

8.Avoid using a coding style that is too difficult to understand

9.Avoid using an identifier for multiple purposes

STRUCTURED CODING

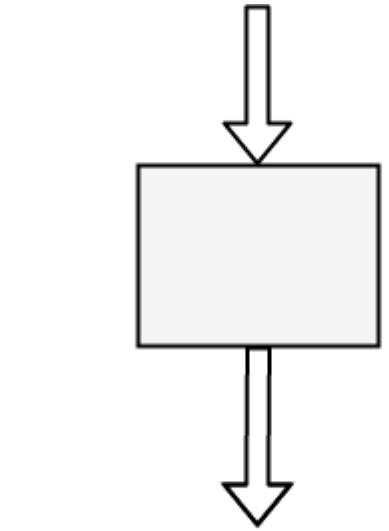
- In structured programming , the whole program is sub divided into small modules so that the program becomes easy to understand.
- The purpose of structured programming is to linearize control flow through a computer program so that the execution sequence follows the sequence in which the code is written. This linear flow of control can be managed.
- This enhances the readability, testability, and modifiability of the program.
- Structured programming allows the programmer to understand the program easily.(If a program consists of thousands of instructions and an error occurs then it is complicated to find that error in the whole program, but in structures programming, we can easily detect the error and then go to that location and correct it which saves a lot of time.)

These are the following rules in structured programming:

Structured Rule One: Code Block

If the entry conditions are correct, but the exit conditions are wrong, the error must be in the block. This is not true if the execution is allowed to jump into a block. The error might be anywhere in the program. Debugging under these circumstances is much harder.

Rule 1 of Structured Programming: A code block is structured, as shown in the figure. In flow-charting condition, a box with a single entry point and single exit point are structured. Structured programming is a method of making it evident that the program is correct.

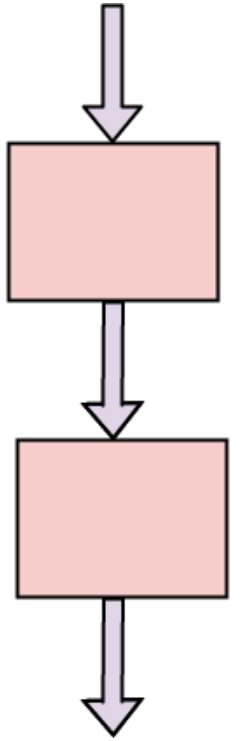


Rule1: Code block is structured

Structure Rule Two: Sequence

A sequence of blocks is correct if the exit conditions of each block match the entry conditions of the following block. Execution enters each block at the block's entry point and leaves through the block's exit point. The whole series can be regarded as a single block, with an entry point and an exit point.

Rule 2 of Structured Programming: Two or more code blocks in the sequence are structured, as shown in the figure.

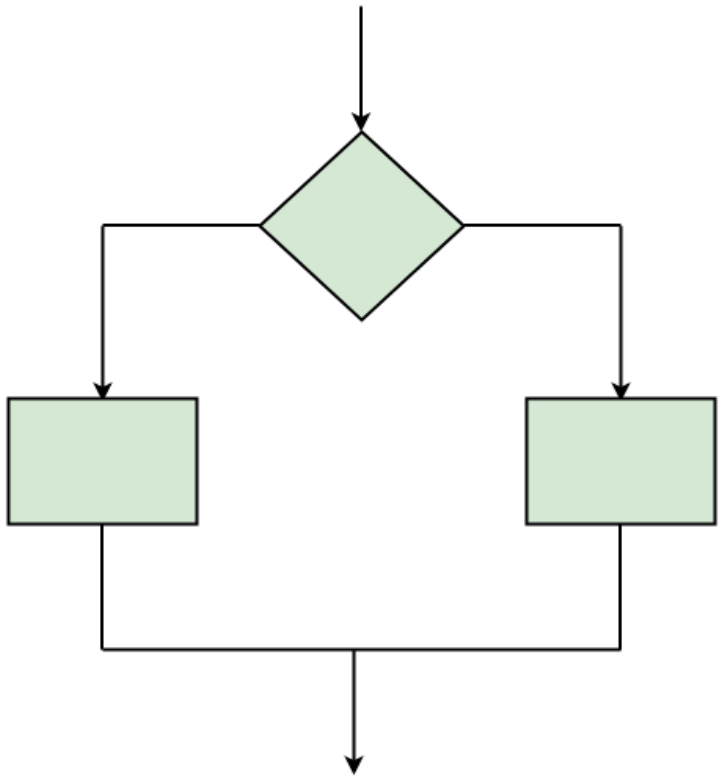


Rule2: A sequence of code blocks is structured

Structured Rule Three: Alternation

If-then-else is frequently called alternation (because there are alternative options). In structured programming, each choice is a code block. If alternation is organized as in the flowchart at right, then there is one entry point (at the top) and one exit point (at the bottom). The structure should be coded so that if the entry conditions are fulfilled, then the exit conditions are satisfied (just like a code block).

Rule 3 of Structured Programming: The alternation of two code blocks is structured, as shown in the figure.

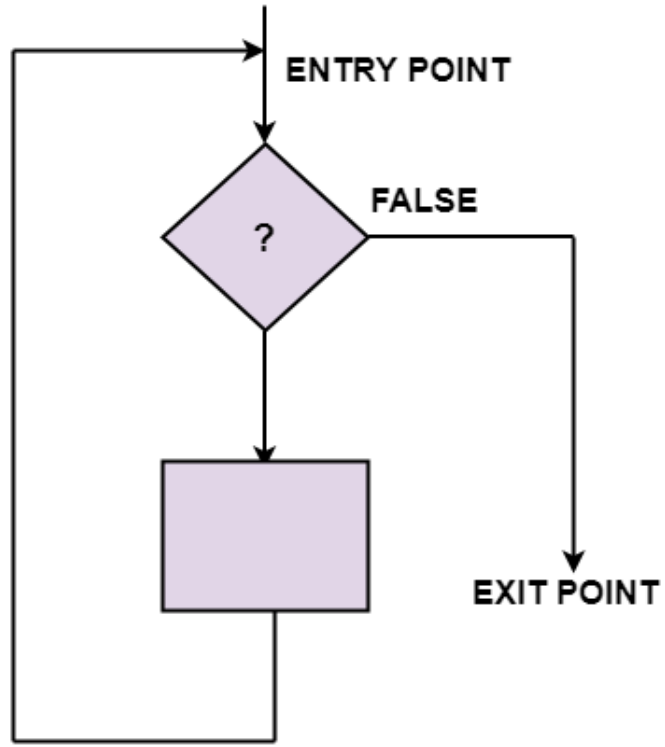


Rule 3: An alternation of code blocks is structured

Structured Rule 4: Iteration

Iteration (while-loop) is organized as at right. It also has one entry point and one exit point. The entry point has conditions that must be satisfied, and the exit point has requirements that will be fulfilled. There are no jumps into the form from external points of the code.

Rule 4 of Structured Programming: The iteration of a code block is structured, as shown in the figure.

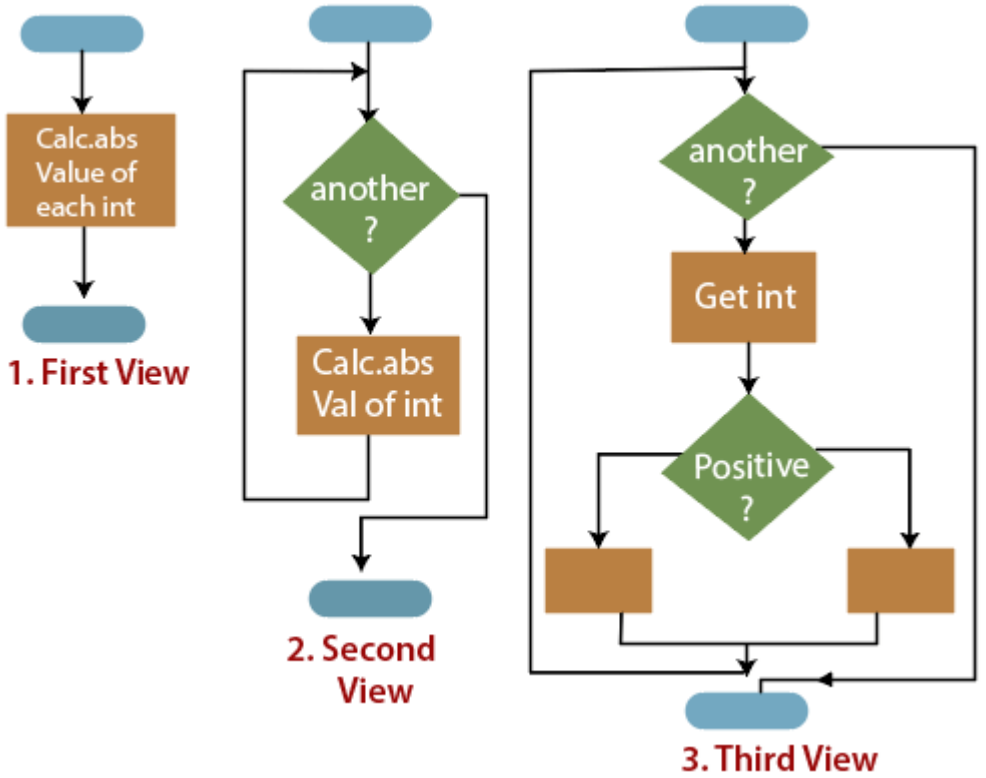


Rule 4: Iteration of code blocks is structured

Structured Rule 5: Nested Structures

In flowcharting conditions, any code block can be spread into any of the structures. If there is a portion of the flowchart that has a single entry point and a single exit point, it can be summarized as a single code block.

Rule 5 of Structured Programming: A structure (of any size) that has a single entry point and a single exit point is equivalent to a code block. For example, we are designing a program to go through a list of signed integers calculating the absolute value of each one. We may **(1)** first regard the program as one block, then **(2)** sketch in the iteration required, and finally **(3)** put in the details of the loop body, as shown in the figure.



Coding Styles

- Programming style refers to the technique used in writing the source code for a computer program.
- Most programming styles are designed to help programmers quickly read and understands the program as well as avoid making errors.
- The goal of good programming style is to provide understandable ,straightforward , elegant code.
- The programming style used in a various program may be derived from the coding standards of a company or computing organization , as well as the preferences of the actual programmer.

Documentation Guidelines

- Software documentation is a critical process in the overall software development process.
- At various stages of development multiple documents may be created for different users.
- In modular programming documentation becomes even more important because different modules of the software are developed by different teams. If anyone other than the development team wants to or needs to understand a module, good and detailed documentation will make the task easier.

. These are some guidelines for creating the documents

- Documentation should be from the point of view of the reader.
- Document should be unambiguous.
- There should be no repetition·
- Industry standards should be used.
- Documents should always be updated·
- Any outdated document should be phased out

Advantages of Documentation

These are some of the advantages of providing program documentation

- Keeps track of all parts of a software or program.
- Maintenance is easier·
- Programmers other than the developer can understand all aspects of software·
- Improves overall quality of the software·
- Assists in user training

A software can have many types of documents associated with it.
Some of the important ones are·

- **User manual** - It describes instructions and procedures for end users to use the different features of the software..
- **Operational manual** - It lists and describes all the operations being carried out and their inter-dependencies..
- **Design Document** - It gives an overview of the software and describes design elements in detail. It documents details like data flow diagrams , entity relationship diagrams, etc.
- **Requirements Document** - It has a list of all the requirements of the system as well as an analysis of viability of the requirements. It can have user cases, real life scenarios, etc.

- **Technical Documentation** - It is a documentation of actual programming components like algorithms, flowcharts, program codes, functional modules, etc.
- **Testing Document** - It records test plan, test cases, validation plan, verification plan, test results, etc. Testing is one phase of software development that needs intensive documentation.
- **List of Known Bugs** - Every software has bugs or errors that cannot be removed because either they were discovered very late or are harmless or will take more effort and time than necessary to rectify. These bugs are listed with program documentation so that they may be removed at a later date. Also they help the users, implementers and maintenance people if the bug is activated.

Modern Programming language features

1. Type checking

Type checking means checking that each operation receives proper number of arguments and are of proper data type.

A=B*j+d; * and - are basically int and float data types based operations and if any variable in this A=B*j+d; is of type other than int and float then compiler will generate type error.

Two ways of Type Checking:

1) Dynamic Type Checking:

It is done at runtime . It uses concept of type tag which is stored in each data objects that indicates the data type of the object.

Example:· An integer data object contains its 'type' and 'values' attribute.

* So Operation only be performed after type checking sequence in which type tag of each argument is checked. If the types are not correct then error will be generated.

2)Static Type Checking:

Static Type Checking is done at compile time.

Information needed at compile time is provided by declaration by language structures.

The information required includes:

1) for each operation: The number, order, and data type, of its arguments.
2) For each variables: Name and data type of data object.
   Example: A+B; in this type of A and B variables must not be changed.
3) for each constant: Name and data type and value .

Eg: const int x=28;

const float x=2.087;. In this data type, the value and name is specified and in further if checked value assigned should match its data type.

2.Data abstraction

- In software engineering , abstraction is a technique for controlling the complexity of computer systems.
- Abstraction is the process of providing only the essentials and hiding the details.
- It works by establishing a level of simplicity on which a person interacts with the system, suppressing the more complex details below the current level.
- The programmer works with an idealized interface (usually well defined) and can add additional levels of functionality that would otherwise be too complex to handle which are hidden from the end user.
- Data abstraction allows handling pieces of data in meaningful ways

- Data abstraction separates the interface and implementation by enforcing a separation between the abstract properties of a data type and the details of its implementation·
- Most programming languages provide data abstraction at various levels like:

Data types: Users are allowed to use the data types without knowing the implementation

Abstract data types : It is a description of components of the data and the operations that are allowed on the data which are independent of the implementation . It explains only the set of values and operations on the data. It does not specify how data will be organised in memory and what algorithms are used for implementing the operations details .All primitive datatypes and UDTs are abstract datatypes.

Information hiding: Providing only the interface and hiding the implementation details in software development . Certain functions abstract the code and reveals the interface.

Modules : This mechanism allows partitioning the program into separate parts .It is a mechanism for an abstraction to be composed of a number of datatypes of which ,it is desired to give clients a limited view.

3.Exception handling

- Exception handling is the process of responding to the occurrence of exceptions - anomalous or exceptional conditions requiring special processing - during the execution of a program.
- Exception handling is a well-known mechanism for introducing forward error recovery in software systems.
- Many important object-oriented programming languages, such as Java, C++,and C# have incorporated this mechanism.
- In traditional software development, a large part of the code of a reliable software system is dedicated to detection and handling of exceptions. This redundant part of the code is usually the least understood, tested, and documented.

- Software developers incorporate exception handling strategy into system design from the start which is very difficult after implementation.
- An exception handler code constitutes the actions executed in response to a raised exception . Control is transferred to the exception handler when the corresponding exception condition is raised.
- The most popular styles of exception handling are 'try and catch’, 'throw’, except’, rescue’, finally' ,'ensure' etc

4. User defined datatype(UDT)

- Modern programming languages provide a variety of predefined datatypes like int ,float ,char etc.
- But not all languages provide all types. Some types may be pre defined and user can define other types in terms of existing types . Eg: Character string may be assigned by user as variable length arrays of characters.
- Fundamental reasons of providing user defined datatypes:

1)To allow specification of often needed types in terms of existing types

2)To allow mapping of concept from problem domain into implementation language

- A UDT is a datatype that is derived from an existing datatype.
- UDTs are used to extend built-in types already available and create customized datatypes.
- Once UDTs are defined ,they are used as elementary datatypes.
- UDTs can also be used as a template for creating datablocks (eg. Structure,class etc)·
- Programming language C supports the use of two UDTs type defintion(typedef) and enumeration(enum):

Eg: typedef int age;

```
age x,y;
enum colours{black,blue,red};
colours foreground,background;
```

5. Concurrency mechanism

- Concurrency mechanism involves the execution of two or more segments of a program concurrently independent of the order of execution.
- Program segments are called tasks or processes.· On a single processor machine , the execution independent program segments can be interleaved to achieve processing efficiency.
- Three fundamental approaches to concurrent programming are
    1) Shared variables
    2) Asynchronous message passing
    3) Synchronous message passing

Shared variables.

- The multiple processes have access to a common region of memory.
- The simplest form is the 'test and set' approach.
- When two tasks are to synchronize, the first task to reach its synchronization point will test and then set a shared memory cell to indicate that it is waiting for the second task.

Asynchronous message passing.

- It involves association of buffers with concurrent tasks.
- A sending task places information in the receiving task buffer.
- Items are removed from the buffer by the receiver as needed.. Removal will be on the first in first out basis

Synchronous message passing.

- The method is called rendezvous. Both symmetric and asymmetric rendezvous are possible
- In symmetric rendezvous first task waits for the next task.
- The output from the first task is the input to the second task.
- The tasks proceed concurrently.
- Each process is required to know the name of the other process involved in the rendezvous.
- Asymmetric rendezvous is similar to a procedure call.
- Information can be transferred between tasks using parameter lists and global variables.