
Familiarization to Mainframe
Lesson 00:
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
People matter, results count.

Course Goals and Non Goals

- Course Goals
  - To learn about how to write good program by understanding concepts like
    - Readability
    - Maintainability
    - Modularity
    - Defensive programming
  - To learn about how to write pseudocode in design phase
  - To develop robust programs by performing Code Reviews and Unit Testing (test cases/results)
  - Understanding Software testing

- Course Non Goals
  - To learn any specific language features in this course. (Language features will be covered in subsequent modules.)

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

Intended Audience

- Programmers

Day Wise Schedule

- Day 1
  - Lesson 1: Introduction to program development with pseudocode
  - Lesson 2: Good Programming Practices
  - Introduction to Mainframe
- Day 2
  - Lesson 3: File Handling and refactoring
  - Working with Mainframe
- Day 3
  - Lesson 4: Exception Handling
  - Lesson 5: Software Reviews and Testing
  - Working with Mainframe
- Day 4
  - Working with Mainframe

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5

Table of Contents

 6

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8

Table of Contents

 9

References

- Expert MVS/XA JCL - Mani Carathanassis
- MVS/JCL - Doug Lowe

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

Next Step Courses (if applicable)

- MVS
- JCL
- COBOL
- VSAM
- CICS
- DB2

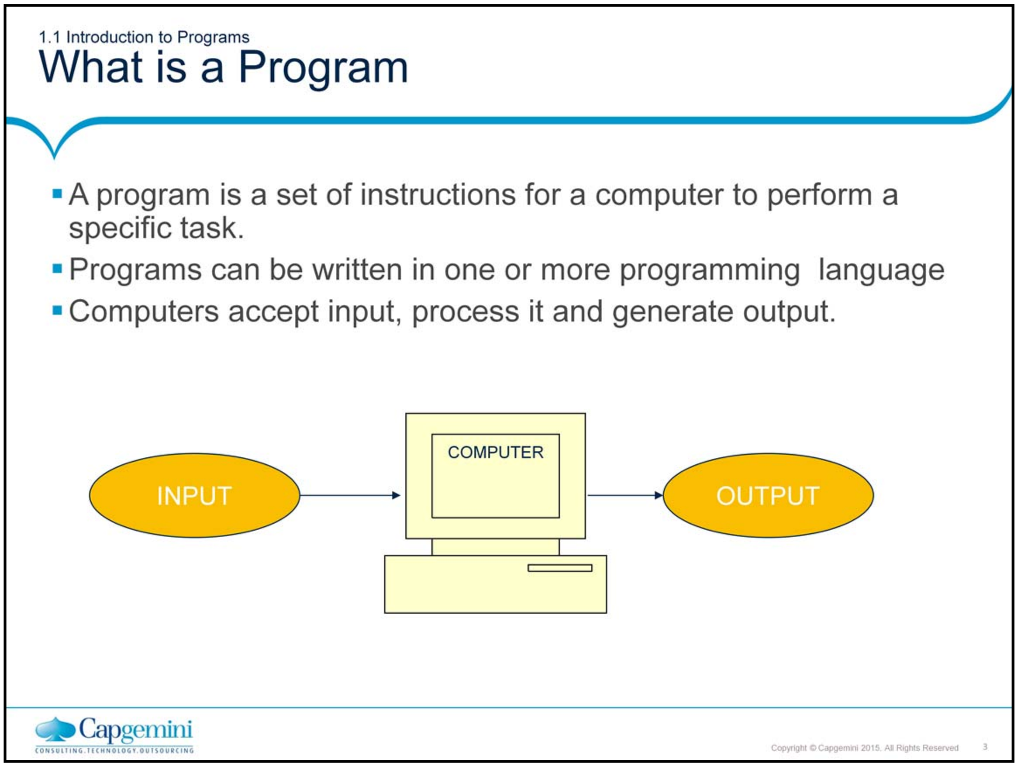
 11

Programming Foundation

Lesson 1: Introduction to Programming

Lesson Objectives

- To understand the following concepts
  - Introduction to programs
  - Types of projects
  - SDLC process of waterfall model
  - Introduction to Pseudocode
  - Usage of variables and operators
  - Introduction to control constructs



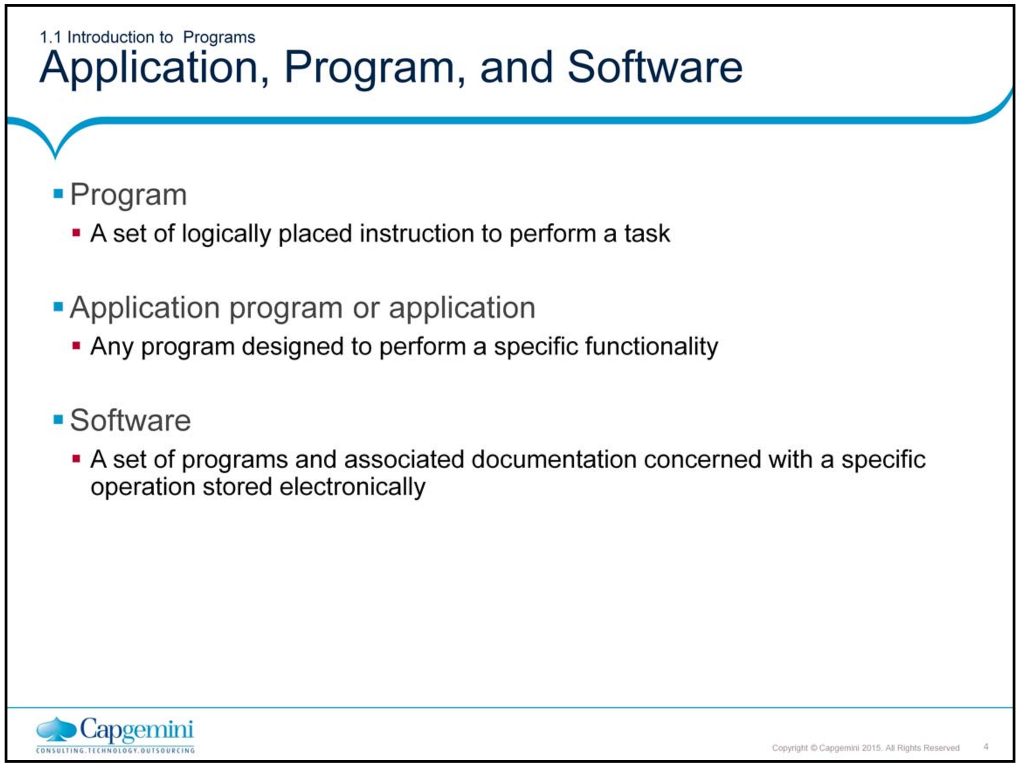
What is a Program?

Set of instructions for a computer to perform a specific task. Programs can be written in one or more programming language.

Example: Program to receive employee details as an input, then calculate and display the gross and net salary of an employee.

What is Programming Language?

- Used to feed instructions to the computer
- Can be categorized as Machine language, Assembly language, Compiled Languages, Interpreted Languages, Object Oriented Languages … etc
- Languages which are more simpler, easier are referred as High-Level Languages
- Low level languages provides little or no abstraction to the internal working of microprocessor



Program:

Set of ordered instructions that enable a computer to carry out a specific task. A collection of instructions that tell the computer what to do.

Ex: Program to find a prime number, Program to print employee pay slip etc..

Application :

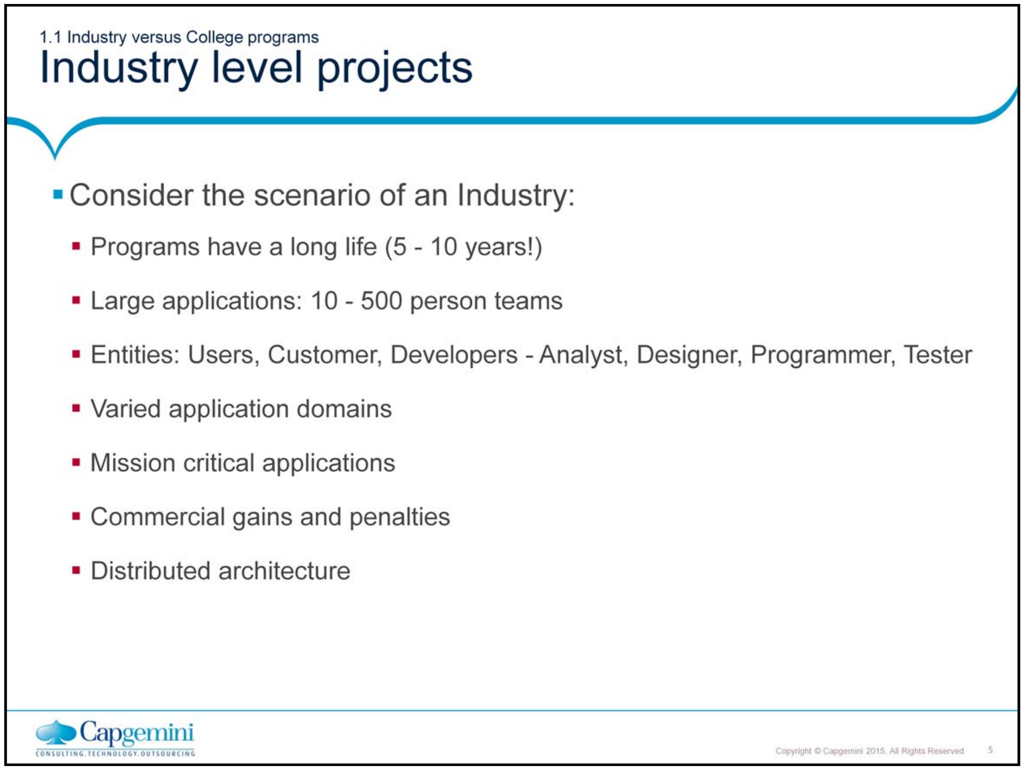
Any program designed to perform a specific function

Ex : Notepad, M S Paint etc

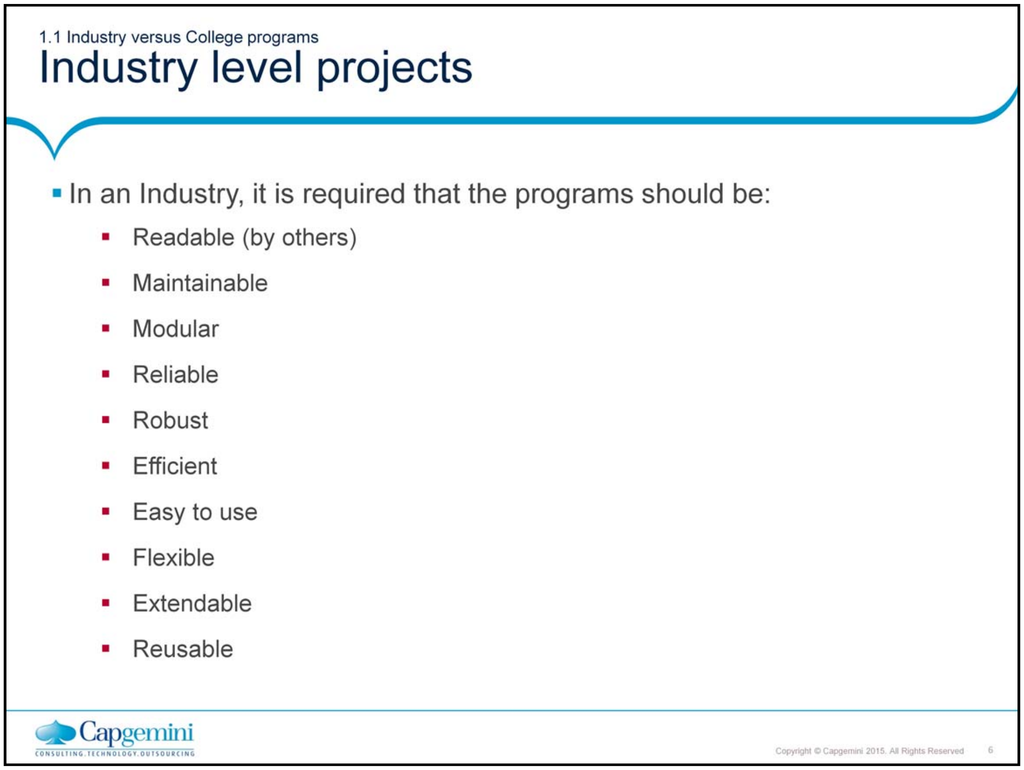
Software:

A set of programs and associated documentation concerned with a specific operation stored electronically

Ex : Microsoft Office, Oracle etc



- Consider the scenario of a College:
  - Throw away programs
    - These programs are not used by any one later
  - Same Users / Developers
  - Small assignments: 1-2 person teams
  - No commercial angle
  - Familiar application domain
  - Low criticality
  - Traditional single-machine architecture



Industry level programs should be

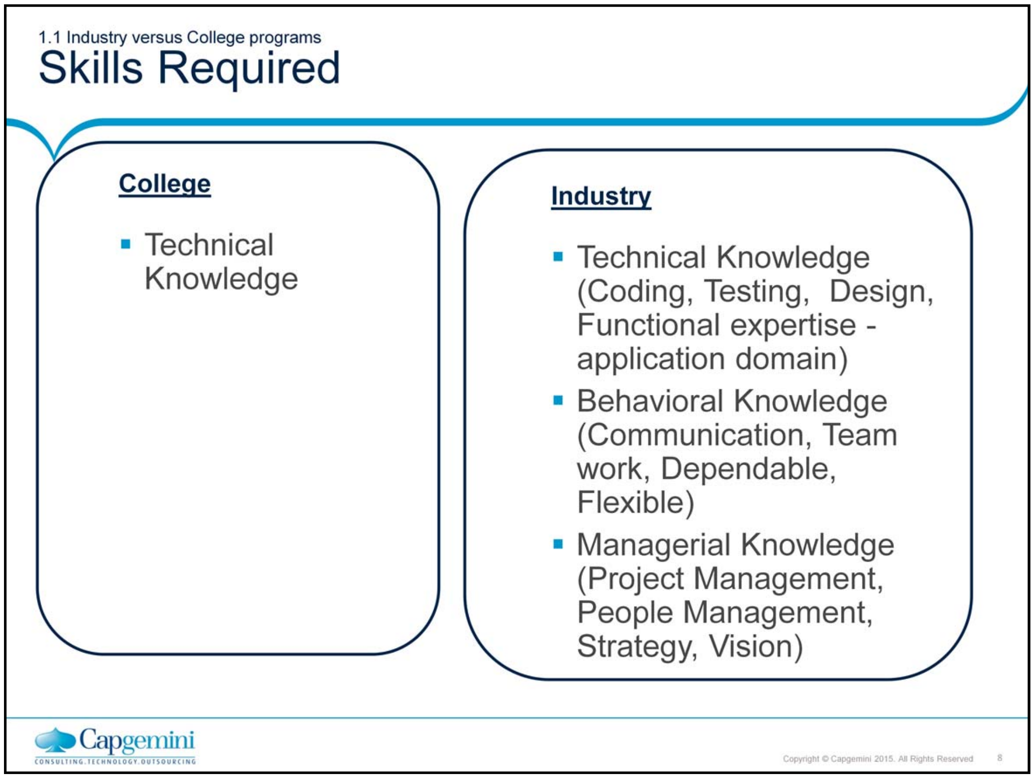
- Readable : Easy to read and understand the code at any time since lifetime of projects will be longer in terms of years.
- Maintainable : If the program is easy to understand and If it is easy to modify then the program is called as maintainable
- Modular : A small unit of code for a single purpose
- Reliable : Reliability describes about the ability of a system will work perfectly as stated without failure or error
- Robust : The ability of a system to continue operating despite abnormalities in input, calculations, etc.
- Efficient : A task which gets done in the specified time with desired quality
- Easy to use: Program which is easy to use by the end users
- Flexible
- Extendable : If the additional features of a program is possible to be included without any side effects, then the program is called as extendable.
- Reusable : If the task written in program is called multiple times, then the program is reusable.

1.1 Industry versus College programs

Industry level projects

- In an Industry, “coding standards” have to be maintained.
- Coding Standards help to read others code, and maintain consistency. They entail standards regarding:
  - naming conventions
  - indentation
  - commenting standards
  - use of global variables
  - modularity

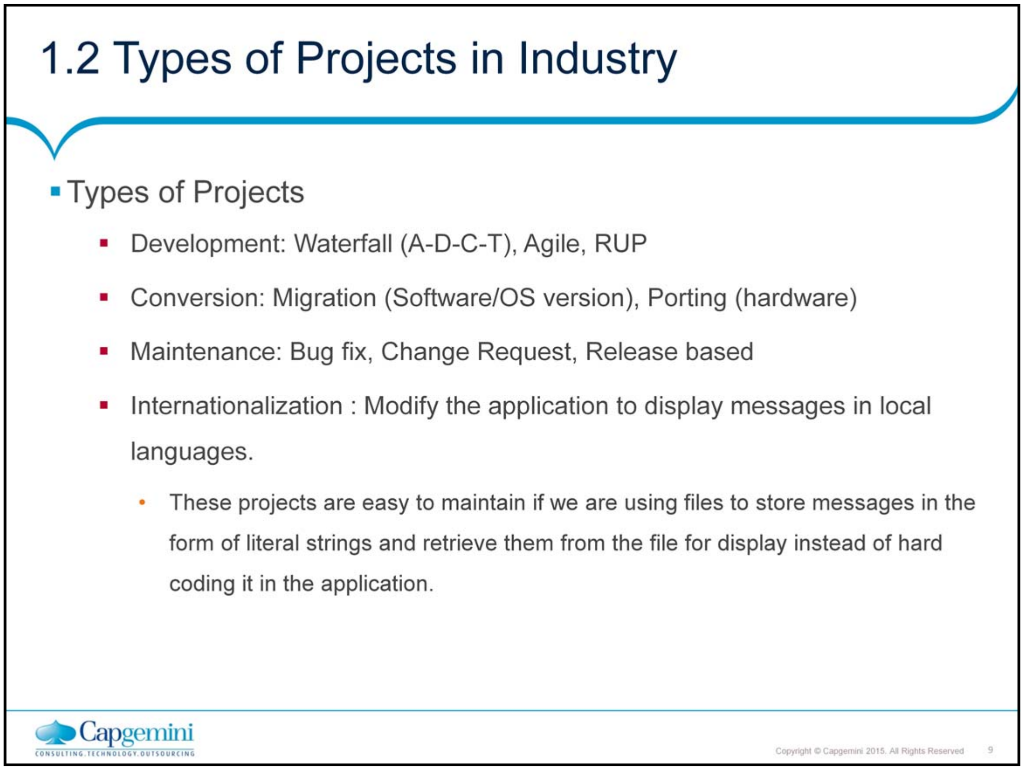
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



A person require much more technical knowledge for writing industry level applications as compared to writing programs in the college.
In college having good technical knowledge for a particular technology is enough to write program but in industry along with technical knowledge you require some other knowledge like

- What are good programming practices?
- How to write test cases?
- You also should have domain knowledge.
- As we need to work in a team in industry, you also need to improve your communication knowledge. You should be dependable and flexible.
- To reach at managerial level, you need to acquire people management skill, Project management skill over a period of time along with the technical skills

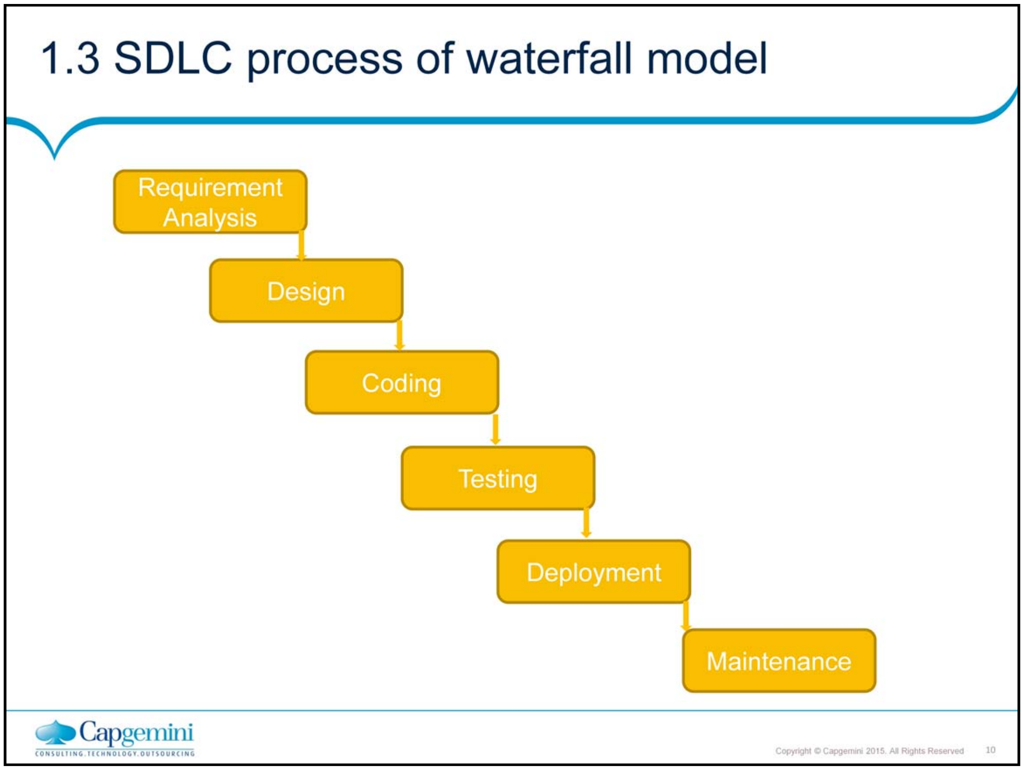
.



In an industry there are various types of projects like

- **Development Projects:** In these projects we may use different types of life cycles like Waterfall model, Agile, RUP (Rational Unified Process) you need to be familiar with these terms

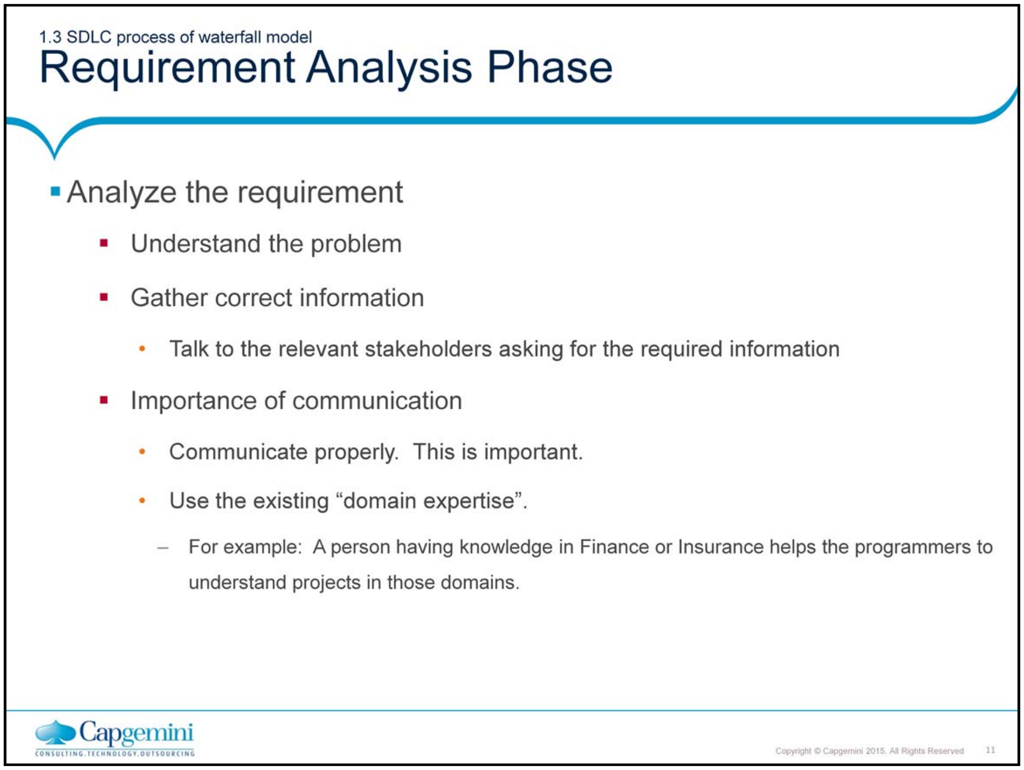
- **Conversion Projects**
  - Migration - It refers to projects like upgrading a software to different version of the OS or DBMS systems
  - Porting – If there is major change in the system like, Hardware Platform has changed.
- **Maintenance projects**
  - Bug fix – If there is an unwanted behavior exists in an existing system due to bug(Problem), then fix the bug by doing changes.
  - Change request – If there are any changes in the functionality are requested from customer such projects are called as changed request type of project.
  - Release base – Products for which periodically the new release of the product is made
- **Internationalization :** Normally most of the applications display messages to user in English but in some cases we may need to change the application to display messages in local language based on the location where we are using it. In such type of projects we need to change the code accordingly. If we have hard coded the messages in the applications then maintenance will be the tedious activity. Hence the good solution for it is store the messages in a file in the form of literal strings and use these files for displaying messages.



Waterfall Model: The waterfall model is a sequential software development model in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design, coding, testing (validation), deployment, and maintenance.

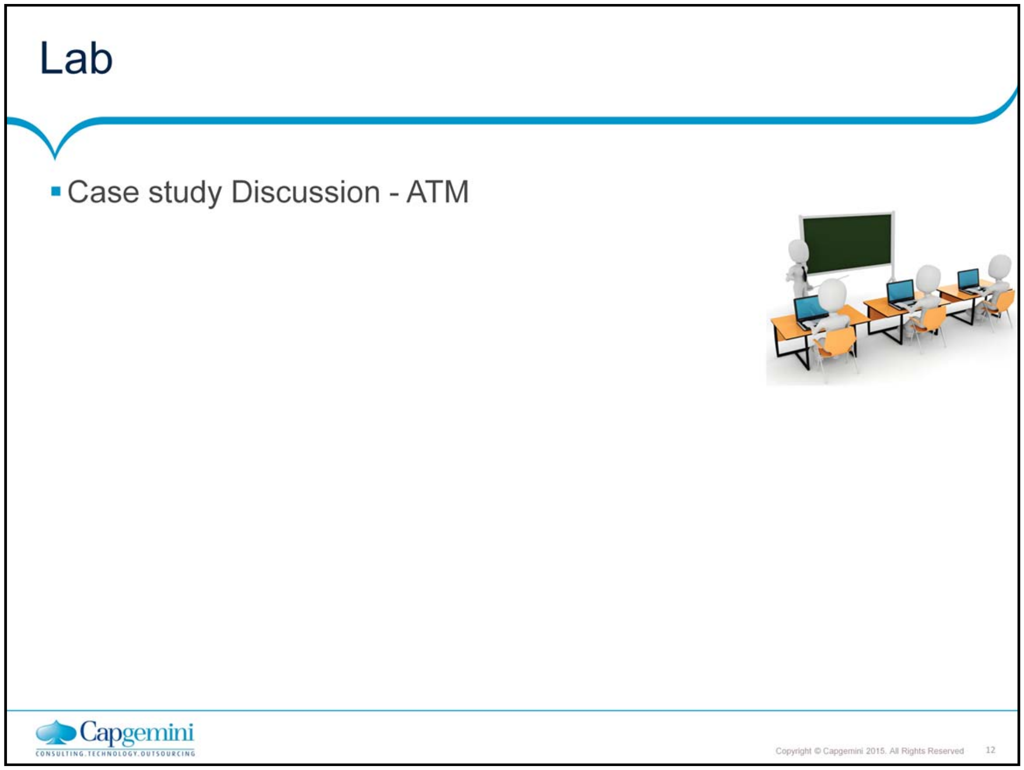
SDLC process of Waterfall Model:

- **Requirement Analysis :** Identify all the requirements. After requirement gathering, analyze the requirements for identifying their validity and the possibility of incorporating the requirements in the system to be developed.
- **Design:** It is a process of creating a detailed specification for a software module . It involves algorithmic design and other implementation specific approaches for a s/w component such as modularity , control hierarchy,, data structures etc. Designers/Technical leads ,senior developers , architects are involved in this phase
- **Coding :** Main objective of this phase is to translate the software design into code , each component identified in design is implemented as a program module following coding guidelines
- **Testing :** Process of checking what's been developed against the requirement.
- **Deployment :** Process of bringing the system into production environment
- **Maintenance :** The maintenance phase involves making changes to hardware, software, and documentation to support its operational effectiveness.



Analyze the requirement:

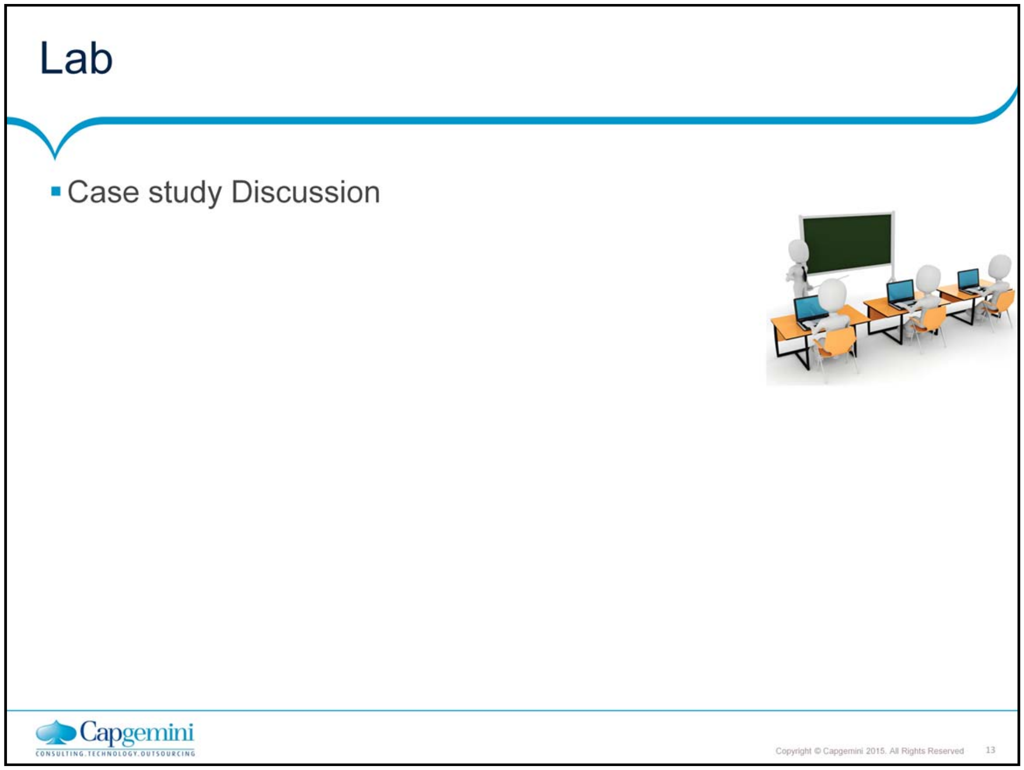
- **Understand the Problem:**
    - Interpret the problem in your own words
    - Determine the outputs required
    - Identify the inputs required to obtain the desired output
    - List out the clarifications required
    - List the assumptions made
    - List the constraints / limitations
- **Gather Correct Information:** Gather the required information from the concern stakeholders by communicating with them.
- **Importance of proper communication:** User talks in language of business, and Programmer talks in the language of technology. Since there is a big gap between these languages, understanding the requirements properly is very important. This is possible through proper communication. The communication exercise is as follows:
    - Consider income tax calculation logic is written by you. If you want to check whether the Income Tax calculation written in the program, is correct as per the current Tax laws? Where can you get this information?
    - Draft an email to the relevant stakeholders, requesting for the required information.



Case Study: The ATM application aims at performing ATM transactions and balance enquiry of an existing account holder in a user friendly way

Following is a list of functionalities of the system. Wherever, the description of functionality is not adequate; you can make appropriate assumptions and proceed.

- The user is requested for the card number and his personal pin number for authentication purpose.
- After authenticating the user, the application requests the user to choose any one of the following options:
    - BALANCE ENQUIRY
    - CASH WITHDRAWL
    - MINI STATEMENT
    - QUIT.
- When the user chooses one of the above options, say ‘1’, the balance of the user is retrieved and displayed. The application further requests the user whether he/she wants the report to be generated and responds accordingly.
- When the user chooses ‘2’, transaction is performed based on the request of the user with the help of the transaction file. Thus after the transaction is complete the user’s account is updated.



Case Study – ATM (Contd..)

- When user wants to generate a record for his/her last 5 DAYS transaction, mini statement is opted where details retrieved from Transaction History File.

Updating balance is done when recharge is successful in Master and Transaction File. Thus, the application requests the user for further processing and responds based on the input from the user.

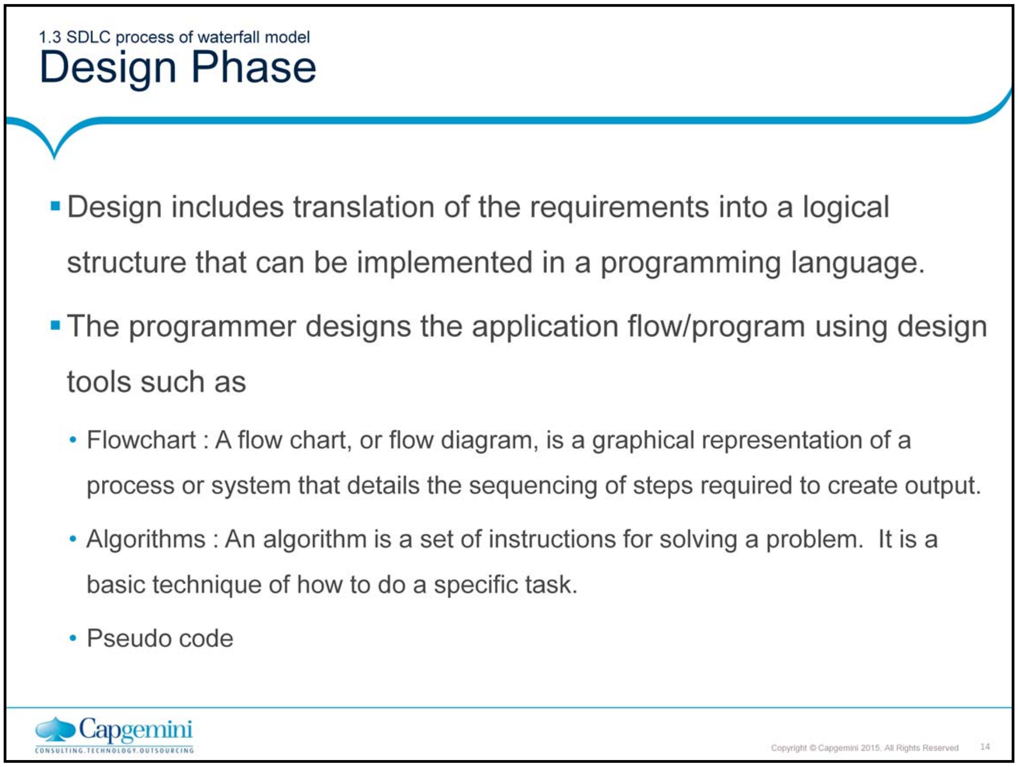
Case Study 1: In a firm there are 10 salespeople and incentive is paid on the portion of sales that exceeds two thirds of the average sales of the group. List the salesperson receiving incentives along with their incentive amount. Incentive amount is 20% the amount of sales that exceeds the two thirds of the average.

Case Study 2: All candidates have to take three tests. Selection for the interview round is done based on the scores of all the three tests. The individual score in each test has to be greater that 75 and the average score across the three tests should be a minimum of 80. An interview call letter is to be sent to candidates who have been selected and a reject letter is to be sent to the others. The interview stage involves two rounds:

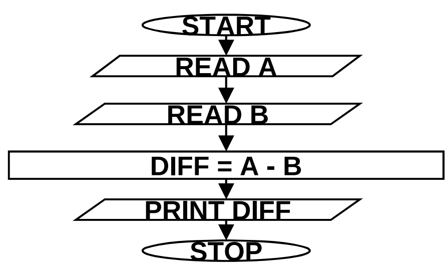
- *Round 1:* qualifying criterion: rating of greater than five on a scale of 1 to 10
- *Round 2:* qualifying criterion: rating of greater than seven on a scale of 1 to 10Candidates go through both rounds of interview.

Selected candidates are sent offer letters and the rest are sent reject letters.

Represent the logic for finding the number of candidates who have been sent interview call letters, who have been sent reject letters and who have been sent offer letters.



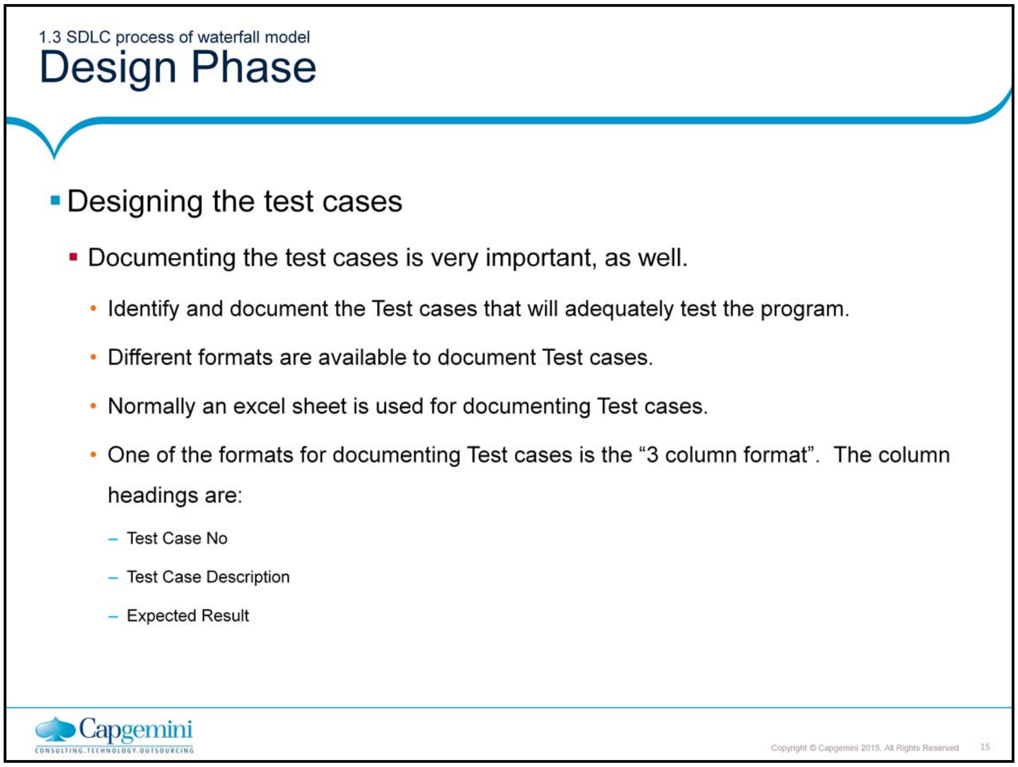
Flowchart: A flow chart, or flow diagram, is a graphical representation of a process or system that details the sequencing of steps required to create output. A typical flow chart uses a set of basic symbols to represent various functions, and shows the sequence and *interconnections* of functions with lines and arrows. Ex: Flow chart to calculate difference b/w two numbers:



Algorithm: An algorithm is a set of instructions for solving a problem. It is a basic technique of how to do a specific task. It takes input, processes it according to a set of instructions, and generates output. An algorithm must provide correct output for every possible input condition. An algorithm must have a definite end point so that when the input has been processed and the desired output achieved, the process stops.

Example: Algorithm to calculate the difference between two numbers

Accept two numbers as num1 and num2 Find the difference between num1 and num2 PRINT DIFFERENCE

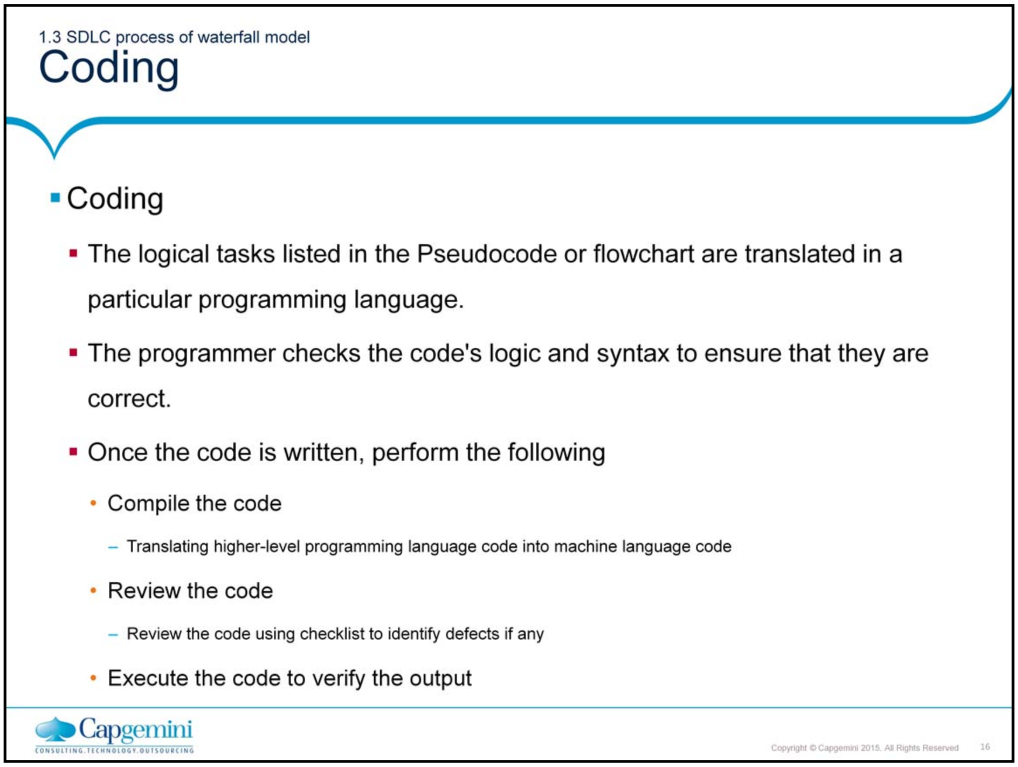


Test case

“A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.”

See the sample test cases given below for checking valid date.

Test Case ID	Test Condition / Scenario	Test Steps	Input/Test Data	Expected Result
TC_1	To check the valid Date	Enter date	10/04/2005	Date should be accepted.
TC_2	To Check the Date when days are entered as Alphabets	Enter day as alphabets. Enter valid month & year	Ten/04/2005	“Enter Valid date”
TC_3	To Check the Date when days are entered as Special Characters	Enter day as special char. Enter valid month & year	!)/04/2005	“Enter valid date”

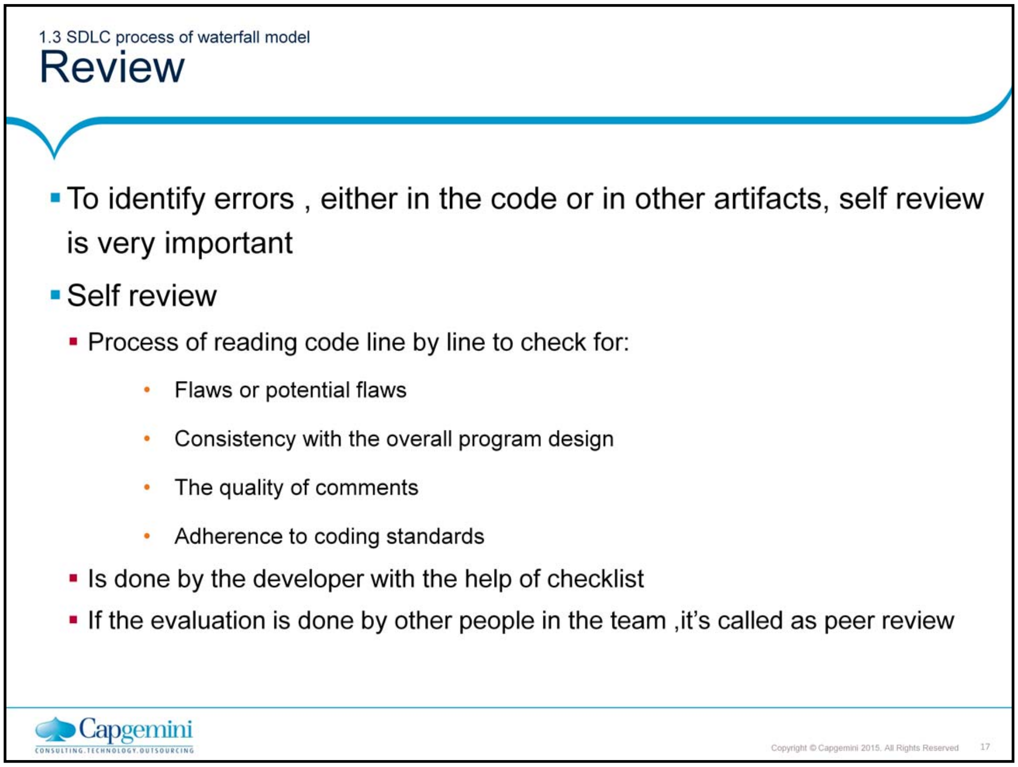


Compiling the code:

- Computers can execute machine language code only.
- The process of translating higher-level programming language code into machine language code is called compiling the program.
  Special programs, called compilers and interpreters, perform this task.
- Code that has errors in it cannot be compiled. The compiler will flag the error, and the programmer must fix the error before trying to compile the code again

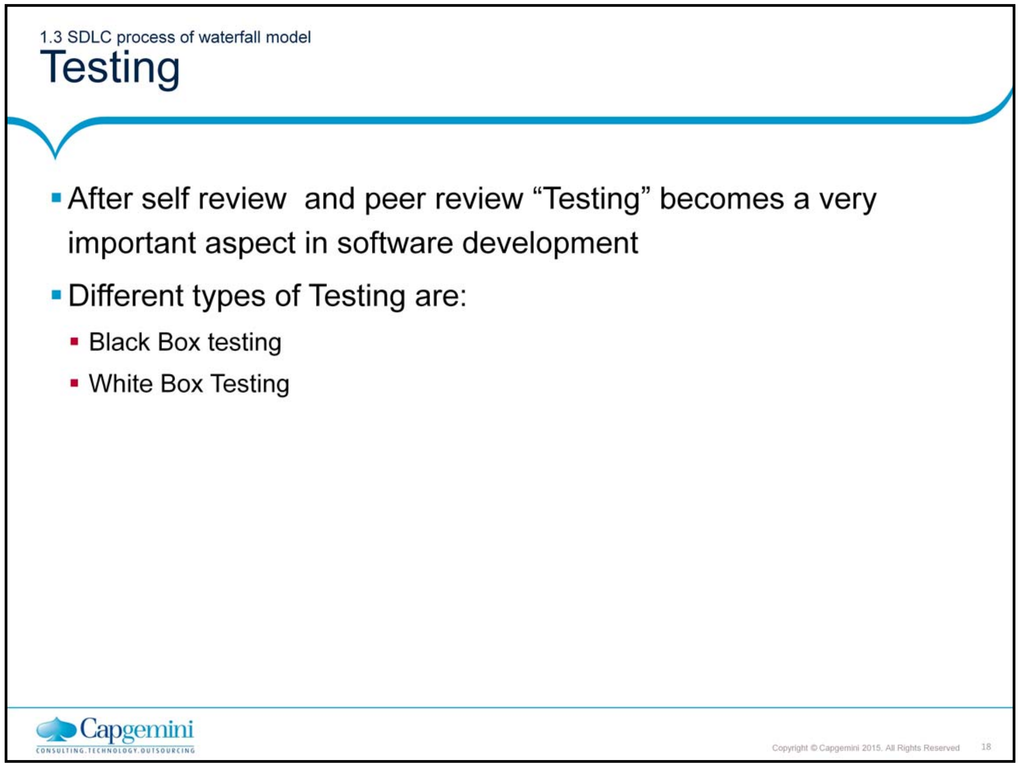
Execute the program and verify the output:

- Once a program is compiled, it can be executed.
- The programmer checks the program's output to ensure that it is correct
- Logic of the program can be checked by using sample inputs and comparing the output obtained with the expected output identified during creation of test cases.



Self review is a process done by the author himself with the aid of tools like checklist, review guidelines, etc..

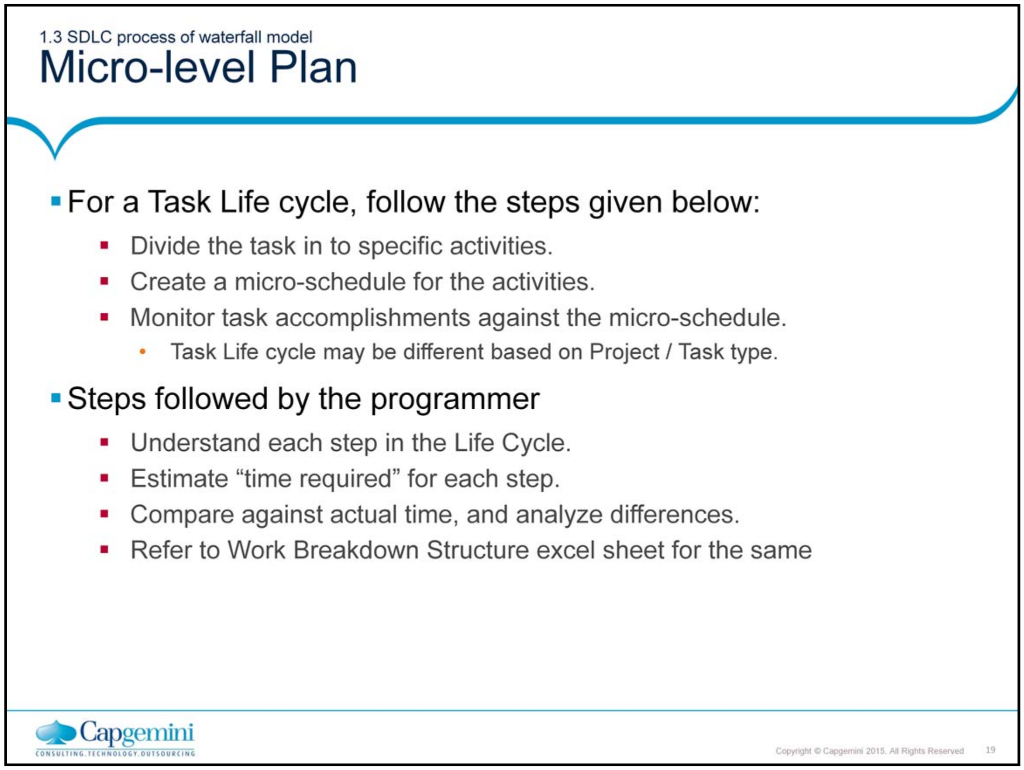
Checklist is a tool with which review can be performed. Checklist is available as an excel document used to verify that all the requirements mentioned in requirement specification is covered and best practices, coding standards are followed.



Different types of testing:

- **Black Box testing** is a software testing method focused on testing whether the input is properly accepted, and output is correctly produced in an application. For an Example, if you want to validate the given input, then use black box testing.

- **White Box testing** is also a software testing method focused on testing the internal structure of the code. For an example, any unreachable path is identifiable using white box testing.

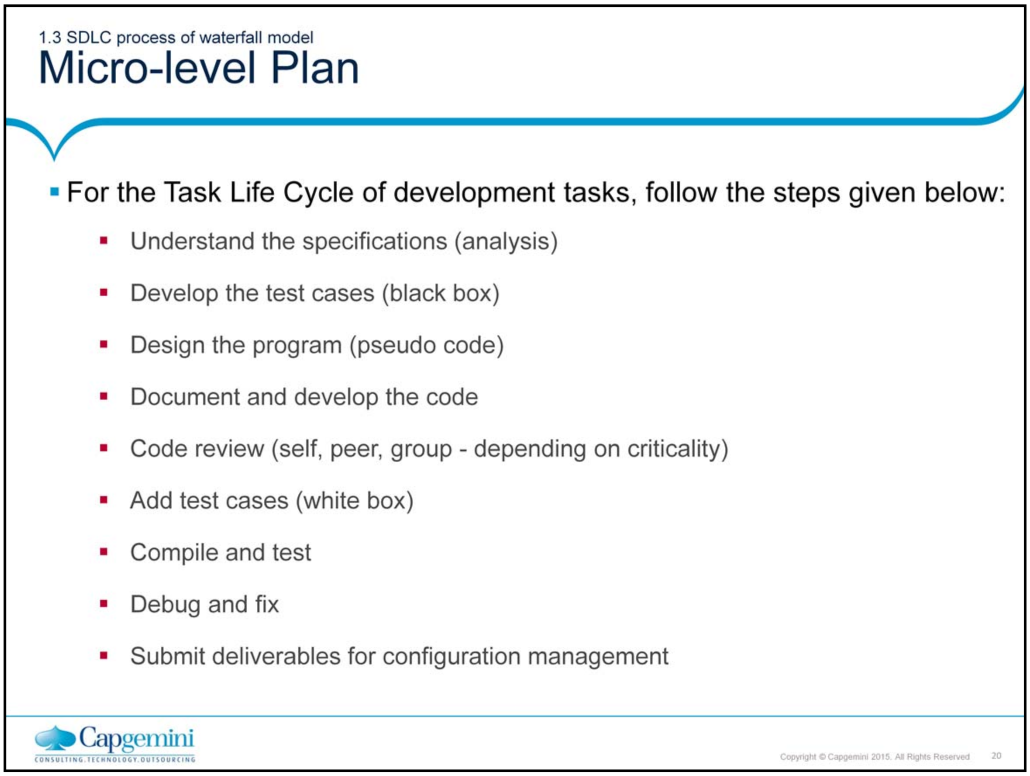


If any of the task is assigned to a programmer then to complete it in time follow the task life cycle.

1. The task life cycle means, divide the task in to smaller activities which should be performed in the sequence to complete the task.
2. Decide the time lines for every activity and prepare a micro schedule for the activity
3. Monitor the accomplishment for every activity against the schedule prepared in the previous step. So that we can take care of timely completion of the task. It will also help to improve our ability to estimate the timelines to complete the task.

Task life cycle will be different for different project or different task type

And hence programmer should first understand each step in the life cycle. Estimate the time required for each step and monitor the task completion against the plan.



In project when a particular task is assigned to developer, to complete the task follow the step given in the slide.

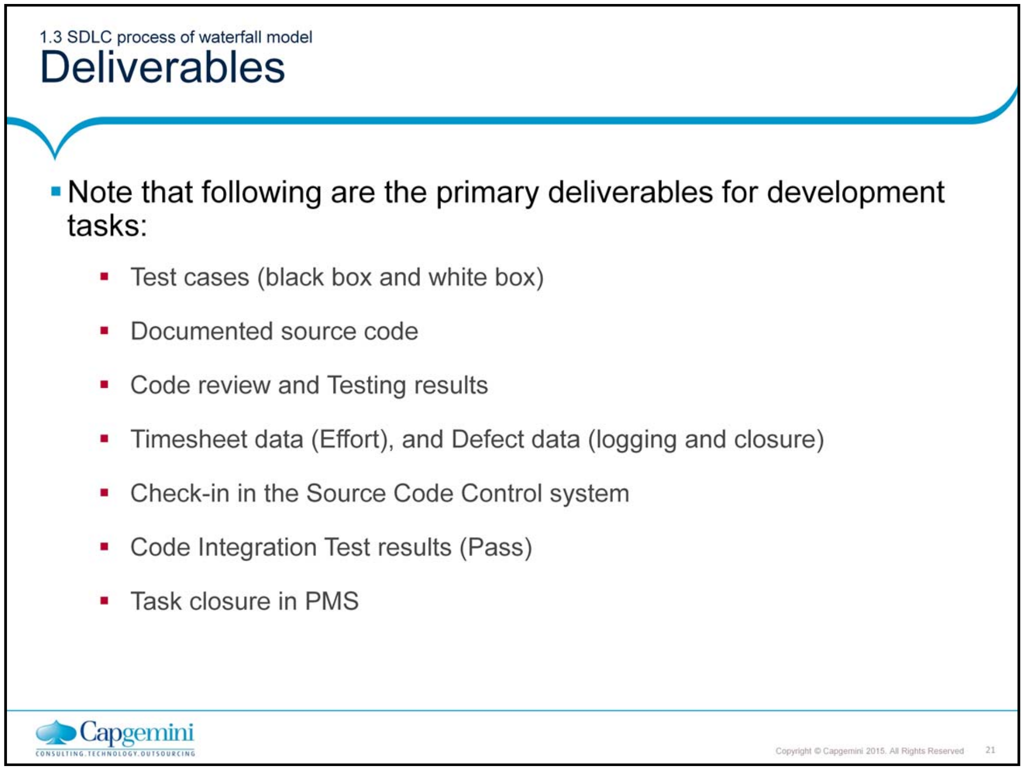
Once the code is ready it should undergo the review. Reviews can be of different types

- Self review – the developer should review his own code with the help of check list
- Peer to peer review – Once the self review is done the code can be exchanged with another developer in the team and he/she will review the code
- Group review- If the code is very critical then it will undergo group review.

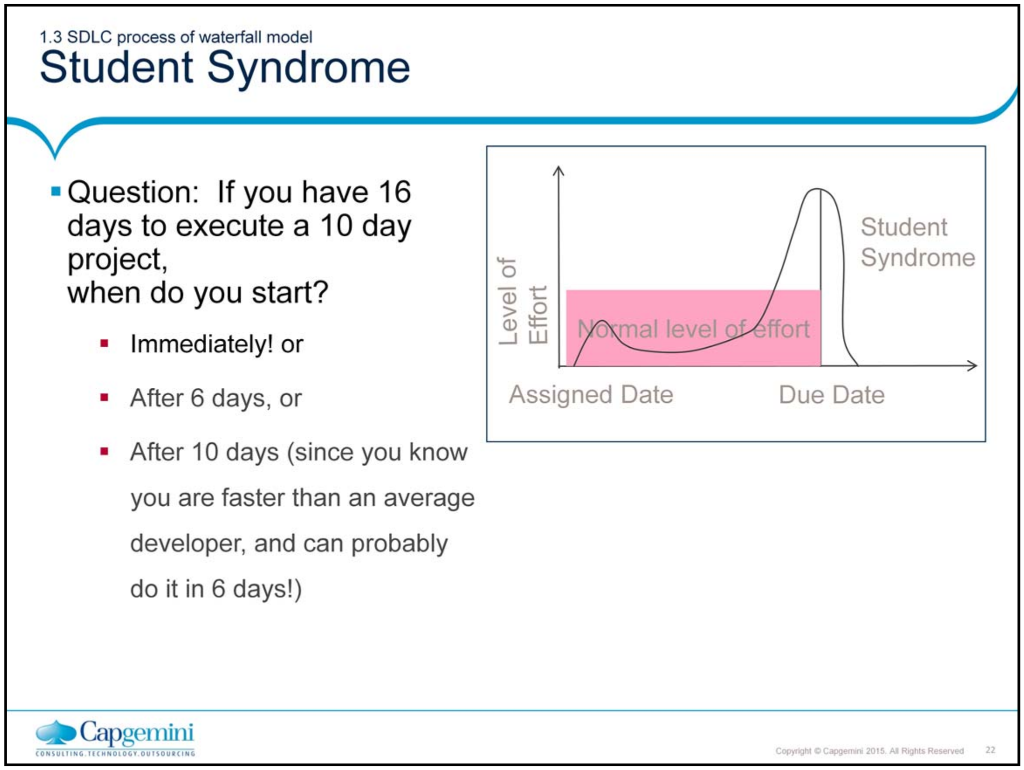
Once the compilation is done test the code with the help of test cases designed. If any bug or defect is found then document the defect and then fix it.

While fixing the defect we will be changing the source code. These changes need to be documented in change history

Once you are satisfied with the code, that it is working as per the specification then we base line the code and submit the deliverables for configuration management because we want that code should not work only on your machine but it should work in integration with other modules also.

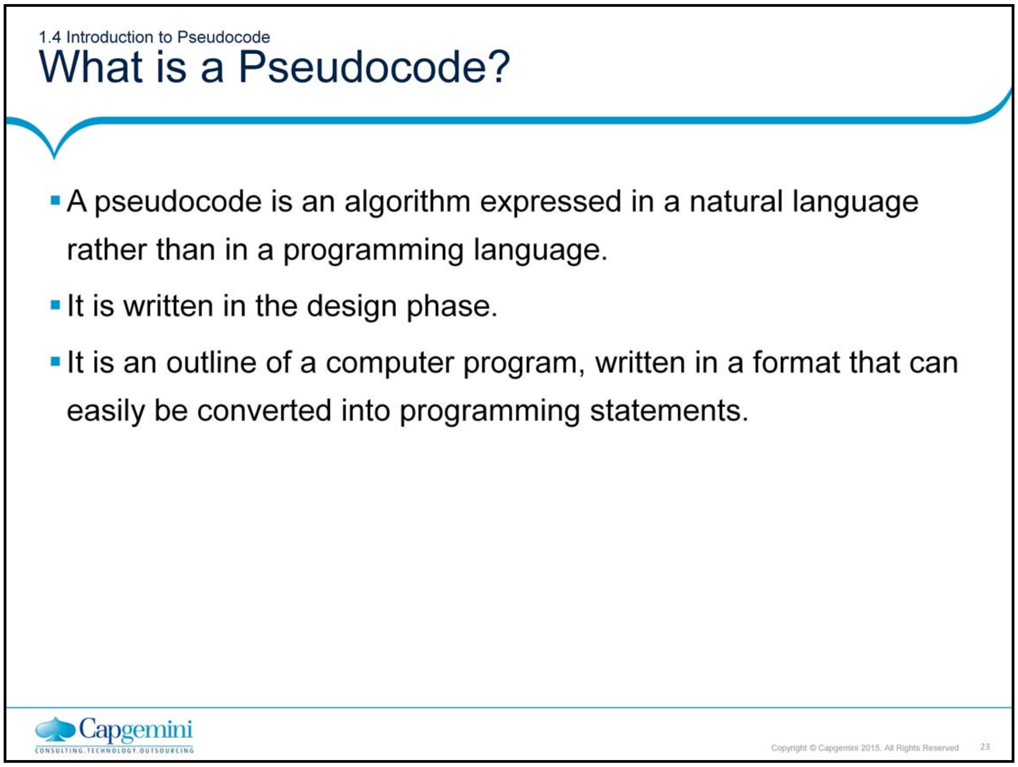


We need to submit above mentioned deliverables.
In defect data we need to log the defect and also mention the closure of it.
Submit the code to configuration management (Check-in in the Source Code Control system). Once the integration test is pass then close the task in PMS (Project Management system). i.e we are ready to work on another task.



Student Syndrome:

- In the above graph. If you observe people are working with less efforts than expected in the beginning and near the dead line people are slogging to complete the work in time.
- This is similar to how a student prepare for the exams for day and night just few days before the exam.
- Avoid student syndromes. While working in project.
- Parkinson's Law:
  - “Work expands to fill (and often exceed) the available time.”
- Murphy's Law:
  - “If anything can go wrong, it will!”



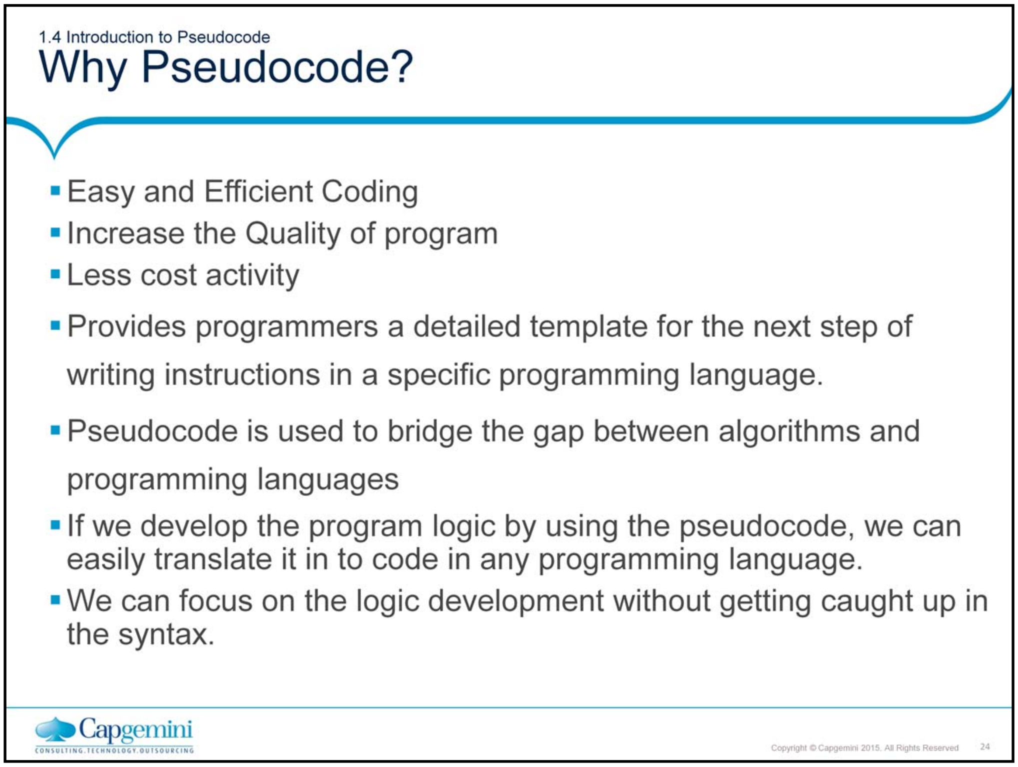
What is Pseudocode?

- Pseudocode is a natural language description of what a computer program must do rather than in a programming language.
- It allows the developer to focus on the logic of the code without being distracted by details of programming language syntax.
- At the same time, the pseudocode needs to be complete. It describe the entire logic of the task so that implementation becomes easier for translating the task line by line into source code.
- The pseudocode describes the logic of the program and acts as a blueprint for the source code to be written by the programmer.

PseudoCode for Finding Whether a number is odd or even:

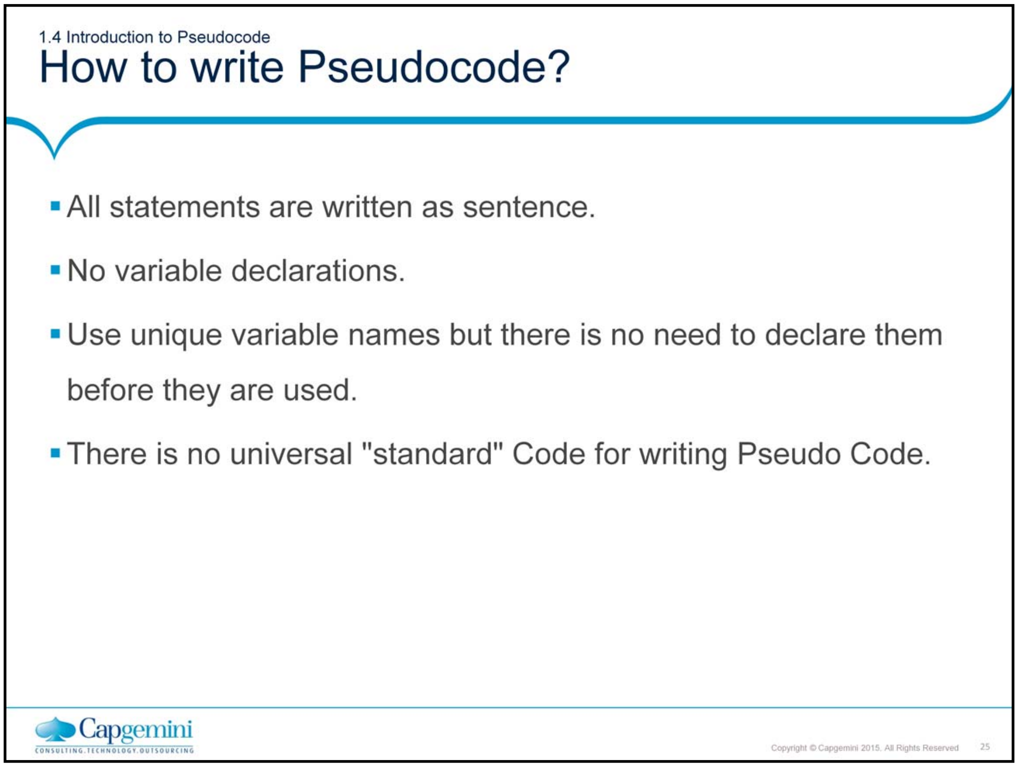
```
BEGIN
        PROMPT "Enter the number" AND STORE IN num
        IF (num MOD 2 == 0) THEN
                DISPLAY "Even Number"
        ELSE
                DISPLAY "Odd Number”
        ENDIF
END
```

PROMPT is a Pseudocode keyword used to take inputs from the keyboard and store in a variable



Why Pseudocode ?

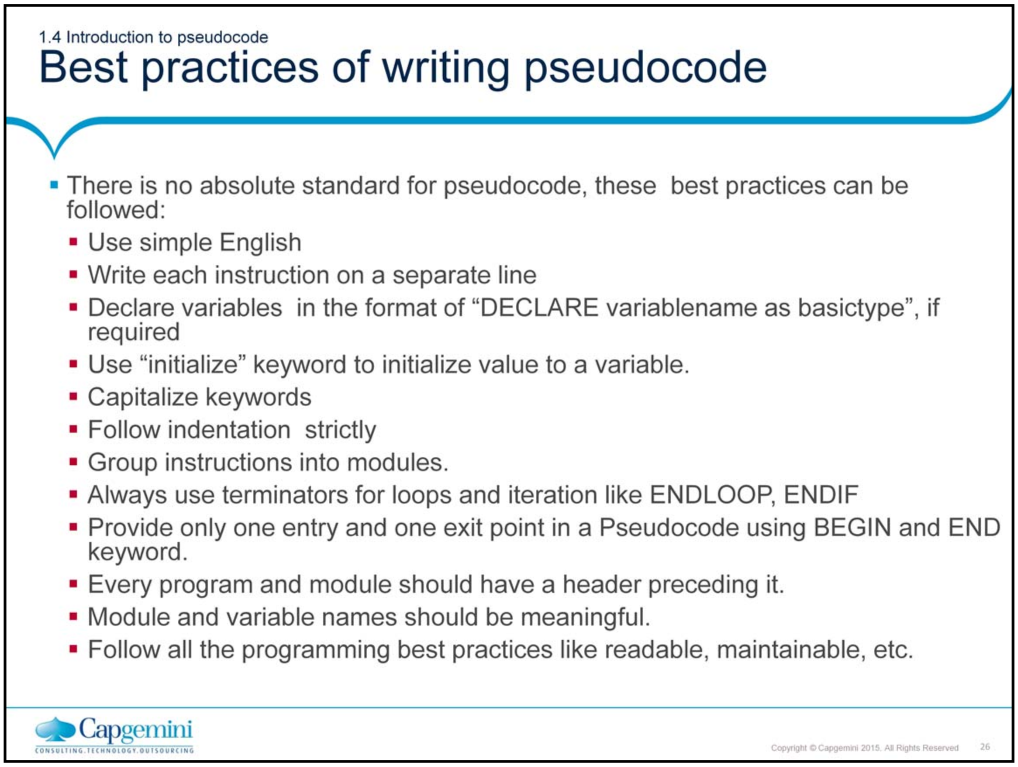
- Easy and Efficient Coding – Easy to solve the task by focusing only on logic of the code with pseudo code rather than any other programming language.
- Increase the Quality of program – Easy way for Analyst to ensure the code matches with design specifications. Once it is matched, then they can easily convert the pseudocode into project specific Language. Thus it helps to ensure requirements are met and that program code meets good software development practice.
- Less Cost activity. Since Catching Logical errors is less tedious than catching them in development process.



Rules to be followed While writing pseudo Code:

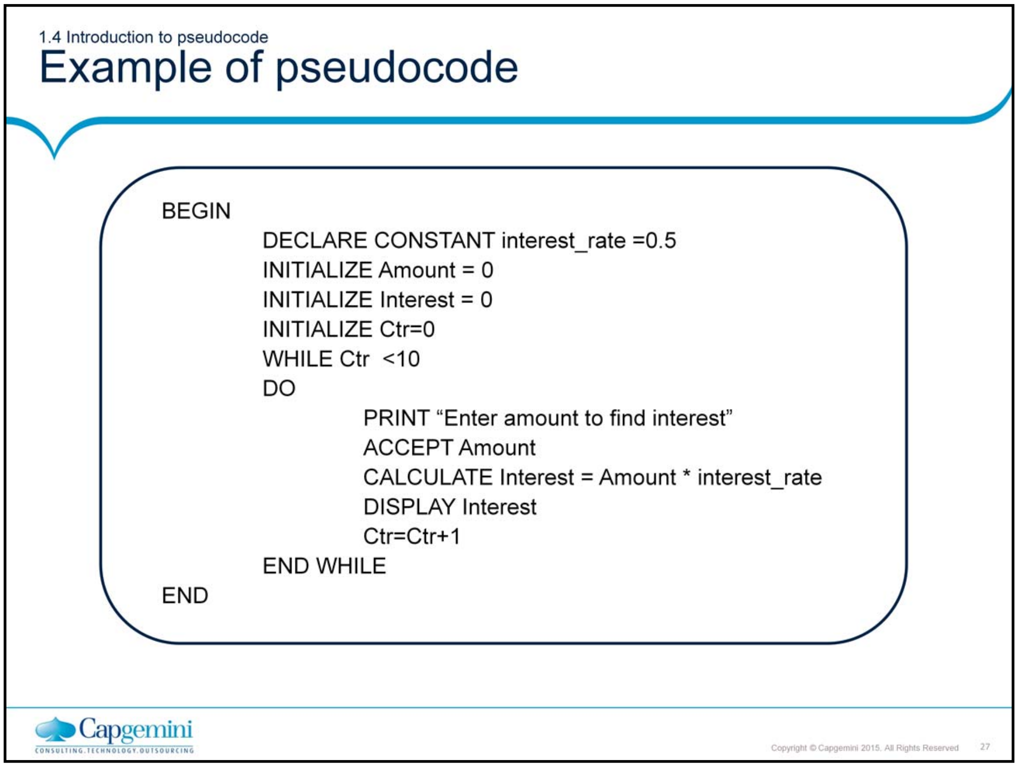
- All statements are written as sentence. Use Words and Phrases which are in line with basic computer operations.
- No variable declarations. Use unique variable names but there is no need to declare them before they are used.
- There is no universal "standard" Code for writing Pseudo Code. But for understandable by others in the project follow the common coding standards specific to the project.
- Some of the Common Coding Notations are

Pseudocode Keyword	**Function / Operation**
DISPLAY/PRINT	Output to screen
PROMPT/ACCEPT	Display a prompt and store into a variable
EQUALS or =	Assignment operation
READ	Read from data source (File)
WRITE	Write to data source (File)
INITIALIZE/SET	Give data an initial value



Pseudocode Example with best practices:

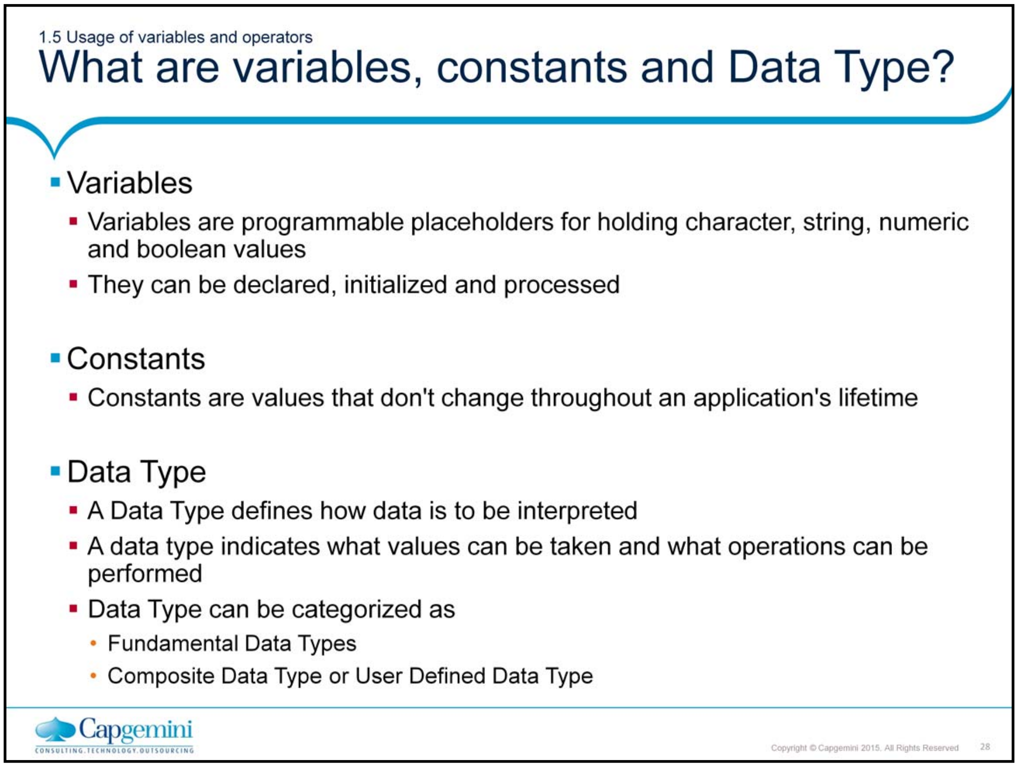
```
BEGIN
          DECLARE num AS INTEGER
           INITIALIZE num TO 0
           PROMPT “Enter the number” AND STORE IN num
           IF(num > 0) THEN
                      DISPLAY “Positive Number"
           ELSE
                      DISPLAY “NegativeNumber”
           ENDIF
END
```



The above pseudocode is used to calculate interest for a given amount based on fixed interest rate. This process will be repeatedly executed for 10 times.

Fixed Interest Rate is 0.5.

Formulae used to calculate interest is Amount * Fixed Interest Rate.



Variable:

Variable's value can be changed at any point in a program.
Each new value must be of the initial type.
Variable describes data that may change while the program is running

Constants:
Constant describes data that is set before the program is used and will remain the same while the program is running
For example, the mathematical symbols pi and e have well-defined values, which are invariant

1.5 Usage of variables and Operators

Fundamental Data Types

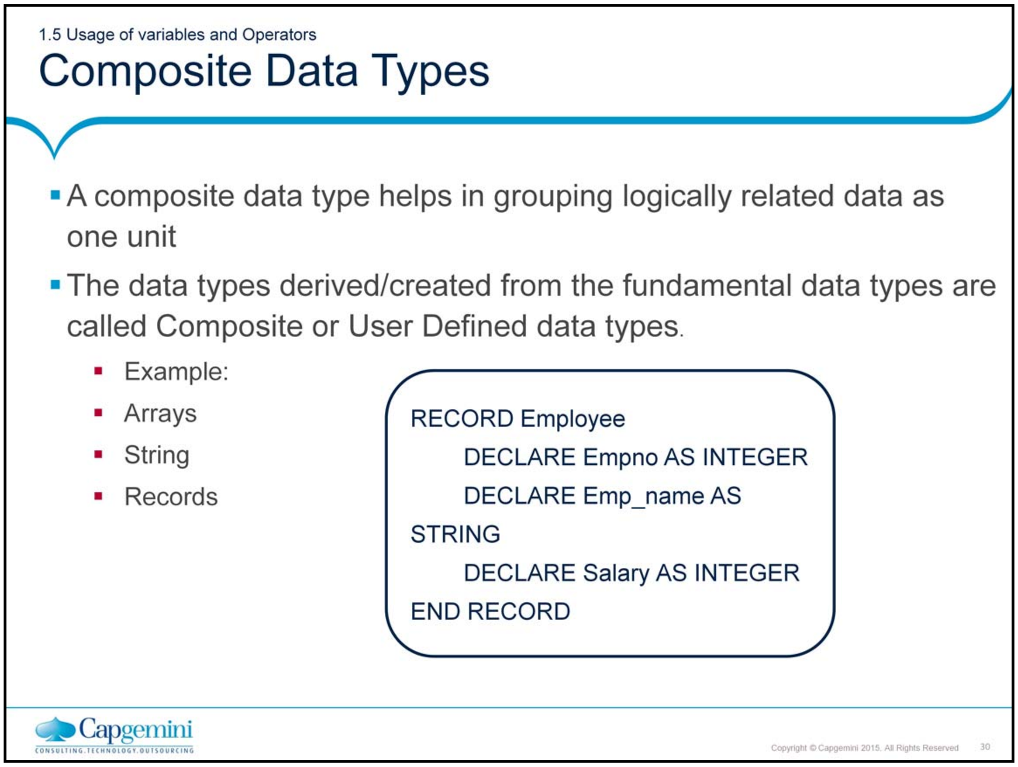
- Fundamental data types are the data type provided as basic building blocks
- They are also known as primitives or basic data type, following table shows the basic data types

Data Type	Description
Character	A character type (typically called "char") may contain a single letter, digit, punctuation mark, or control character. Some languages have two or more character types, for example a single-byte type for ASCII characters and a multi-byte type for Unicode characters
Integer	An *integer* data type can hold a whole number
Real	A *real* type stores rational number having fractional part
Boolean	A *boolean* type, typically denoted "bool" or "boolean", is a single-bit type that can be either "true" or "false".

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Character
The char data type represents single characters, such as ‘m’.
A variable made up of a number of characters is called a string.
Integers
Integers are positive and negative whole numbers
There are different sizes of integers available, including
Short integers and
Long integers
Real
Real data are numbers with a fractional part, for example 8.65 and -0.03
They also include numbers that have no fractional part but are expressed as a whole number with a decimal point, such as 4.0 or -1.0
Boolean
Boolean values are logical values and can be either true or false.

Pointer Data type is integer type
Named Constants are also integers



"Structured data" refers to data that's built up from other types. Use them to clarify relationships between related items
Example :

```
Name   =  InputName;
Address =  InputAdress;
Phone = InputPhone;
Title = InputTitle;
Department = InputDepartment
Bonus = InputBonus
```

```
Employee.Name  = InputName;
Employee.Address = InputAddress;
Employee.Phone = InputPhone;
Supervisor.Title = InputTitle;
Supervisor.Department  = InputDepartment;
Supervisor.Bonus = InputBonus;
```

Why to create user defined datatypes?

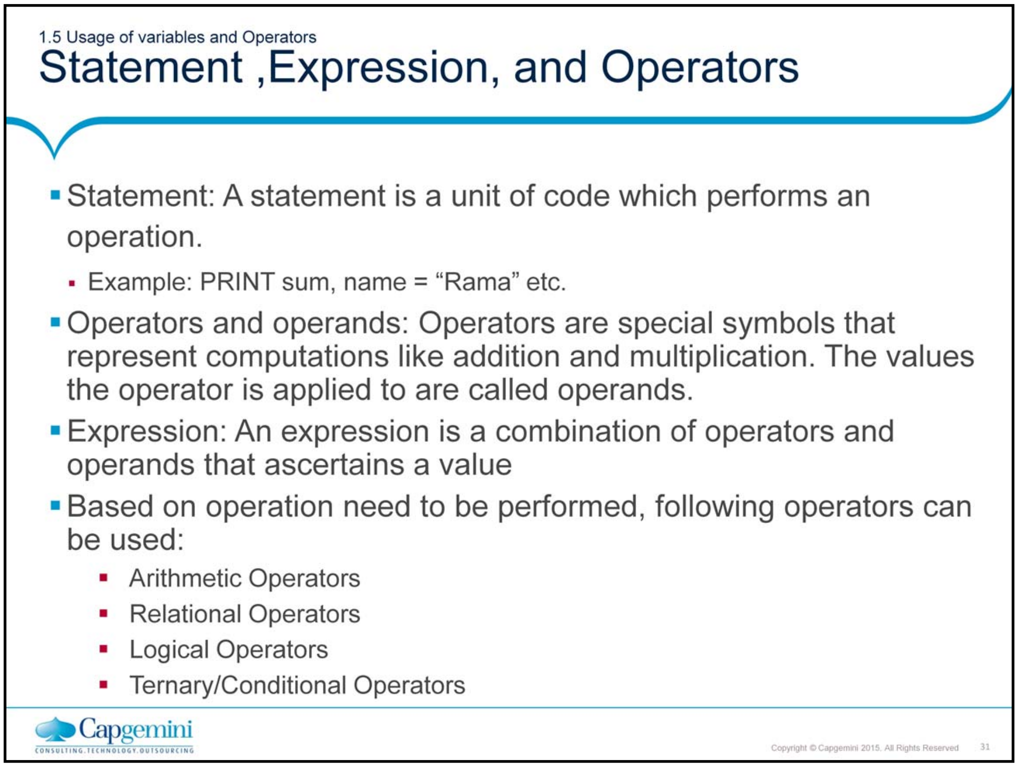
- To increase reliability

  One could specify the range of values that a variable of a User Defined data type can take.

- To make up for language weakness

  If a language does not support a type the user wants, it is possible to create it yourself.

  Example, a student in 'C



Operators and operands, Expression
An operator is a symbol that represents a specific action that must be performed on data

Ex:In the expression *sum=num1+num2*
sum,num1 and num2 are the *operands* and + is the *operator*

Operators have rules of precedence and associativity that are used to determine how expressions are evaluated.

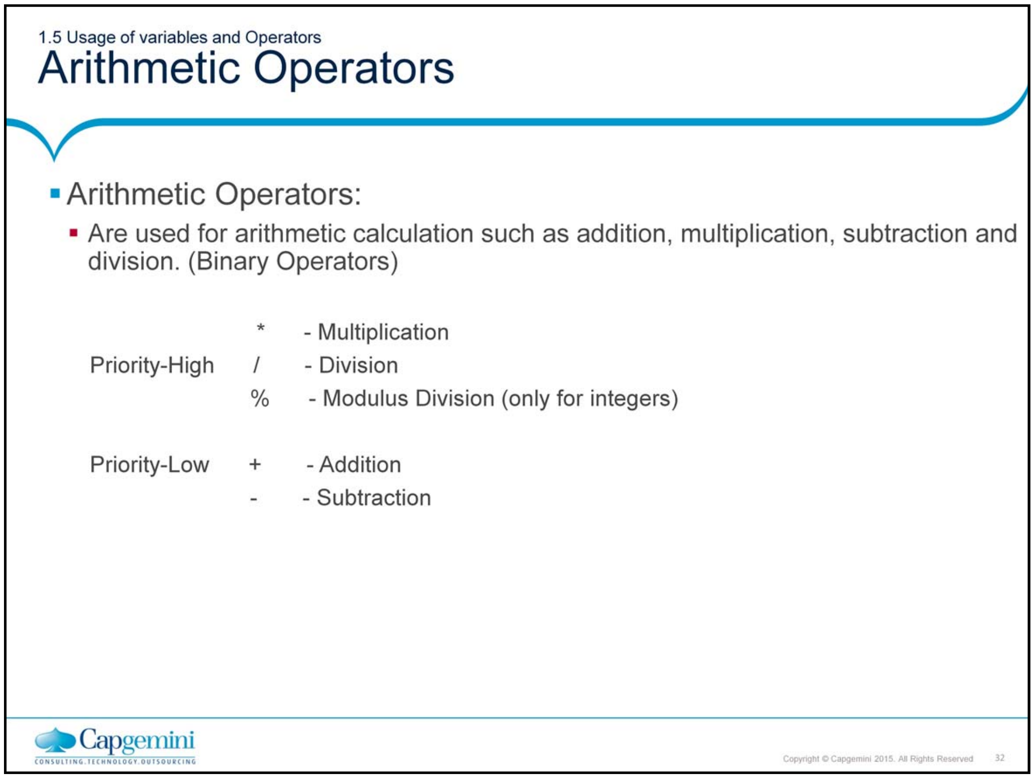
Expressions**:**

An Expression is a combination of constants and variables together with the operators.

Constants and variables by themselves are also considered as expressions. An expression that involves only constants is called a **Constant Expression**.

Note: Balanced parentheses can be used in combining constants and variables.

Operators have been classified into categories based on the operation that they perform. Refer the slide for the different categories.



Types of Operators: Arithmetic Operators:

There are five types of arithmetic operators that are used for arithmetic calculations such as, addition, subtraction, multiplication and division.

These five operators are binary operators that is, they require two operands.

Each of these operators work with values of type integers, Real and character.

1.5 Usage of variables and Operators

Relational Operators

- Relational operators are used to compare two operands to check whether they are equal, unequal or one is greater than or less than the other

Relational Operator	Meaning	Relational Expression
<	Less than	expr1 < expr2
<=	Less than or equal to	expr1 <= expr2
>	Greater than	expr1 > expr2
>=	Greater than or equal to	expr1 >= expr2
==	Equal to	expr1 ==expr2
!=	Not equal to	expr1 != expr2

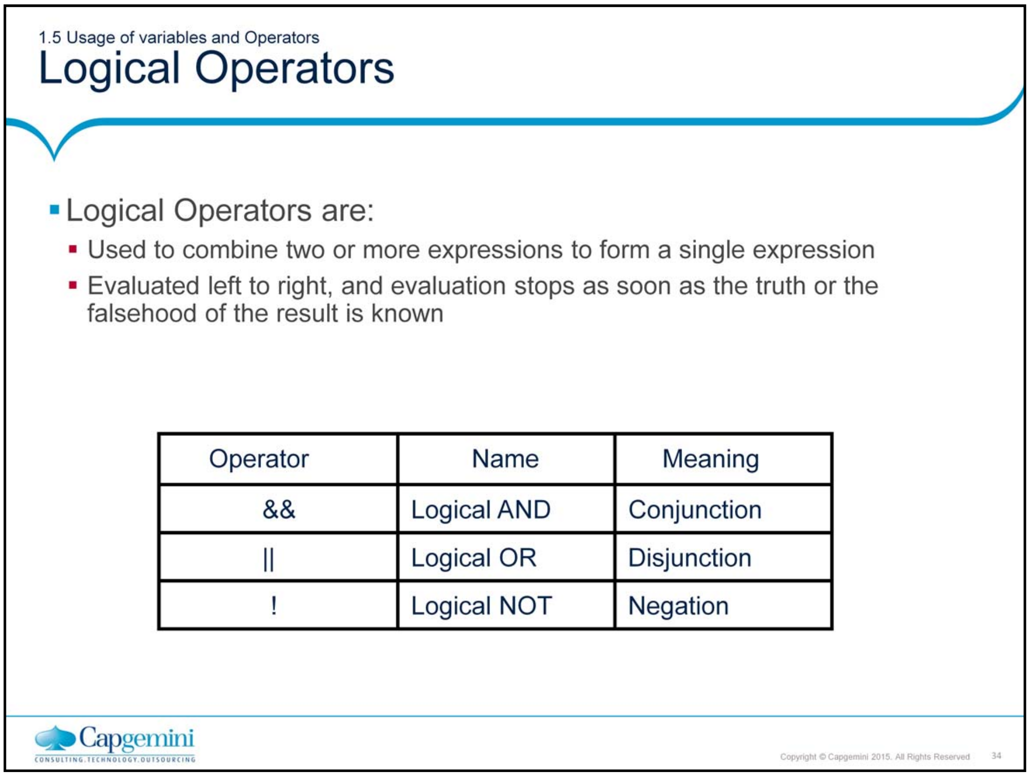
Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

<u>Types of Operators: Relational Operators:</u>

Relational Operators are used to compare two operands to check whether they are equal, unequal or one is less than or greater than the other.

There are six relational operators for comparing the values of two expressions and the expression so formed is called a Relational Expression.

Table provided in the slide shows relational operators and how they can be used to compare expressions expr1 and expr2.



<u>Types of Operators: Logical Operators</u>:
There are three logical operators for combining expressions into logical expressions.
Table in the slide shows available logical operators

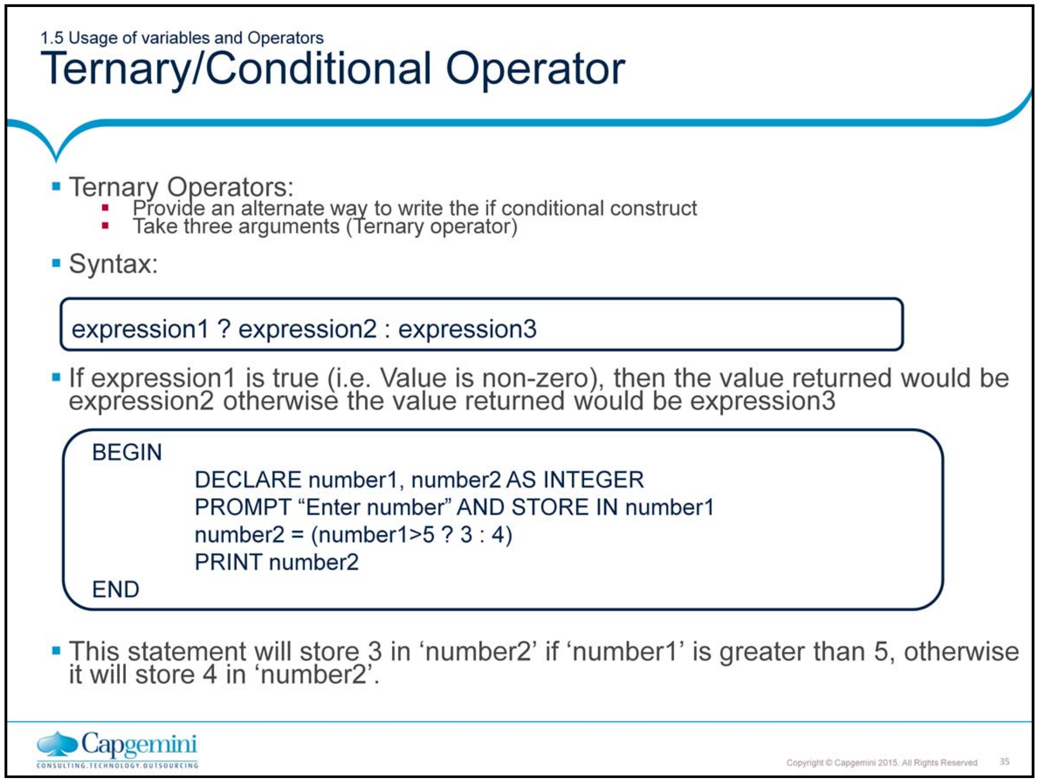
The logical operators **&&** and **||** are binary operators, and ! is a unary operator.
The value of a logical expression is either ‘1’ or ‘0’, depending upon the logical values of the operands. The operands may be of any arithmetic type. The result is always an integer.

Logical AND Operator(&&):
The && operator combines two expressions into a logical expression and has the following operator formation:

expr1 && expr2

An expression of this form is evaluated by first evaluating the left operand. If its value is 0 (false), then right operand is not evaluated and the resulting value is 0 (false).

If the value of left operand is nonzero (true), the right operand gets evaluated. The resulting value is 1 (true) if the right operand has nonzero value (true) and 0 (false) otherwise.



Types of Operators: Ternary Operators:
Simple condition operations can be carried out using the conditional operator (? :). The conditional operator is a ternary operator that is, it takes three arguments. It has following operator formation:

expression1 ? expression2 : expression3

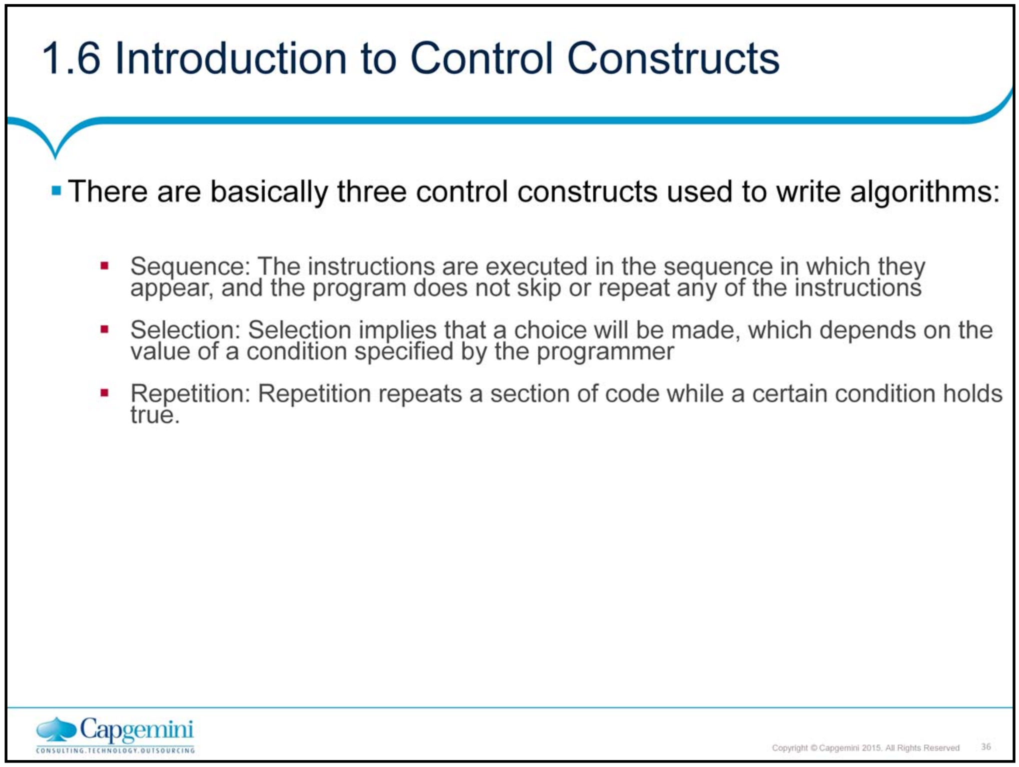
Where ? and : are the two symbols that denote this operator. A conditional expression is evaluated by first evaluating expression1. If the resulting value is true, then expression2 is evaluated and the value of the expression2 becomes the result of the conditional expression. Otherwise, expression3 is evaluated and its value becomes the result.

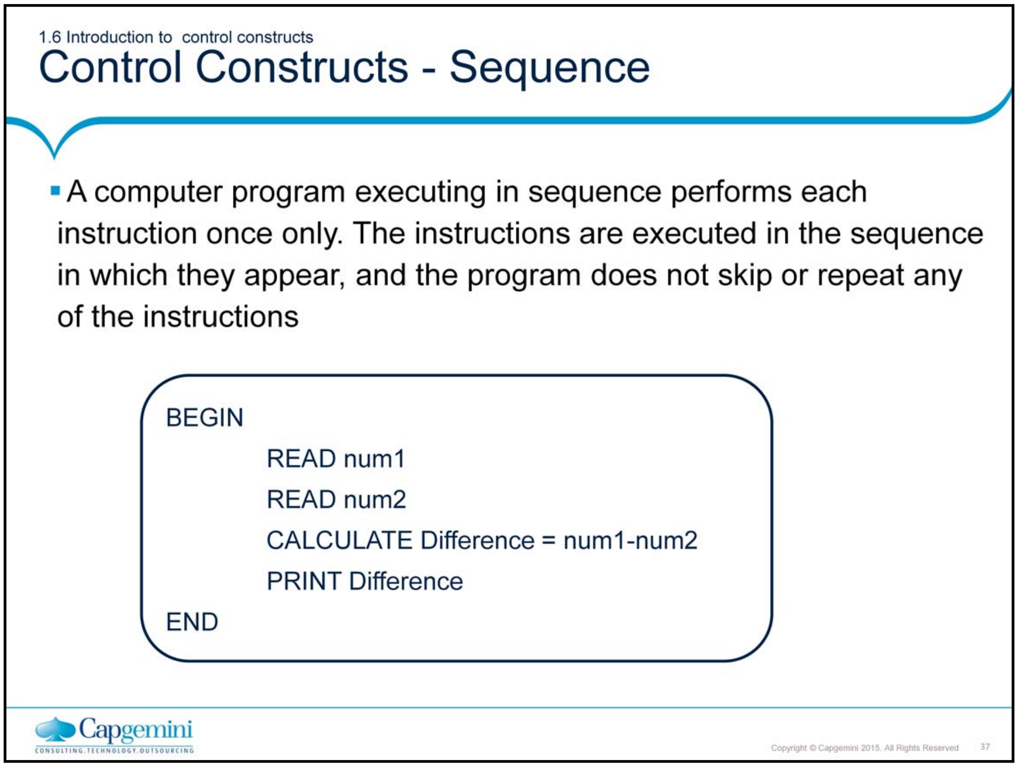
This is used to assign one of the two values to a variable depending upon some condition.

For example:

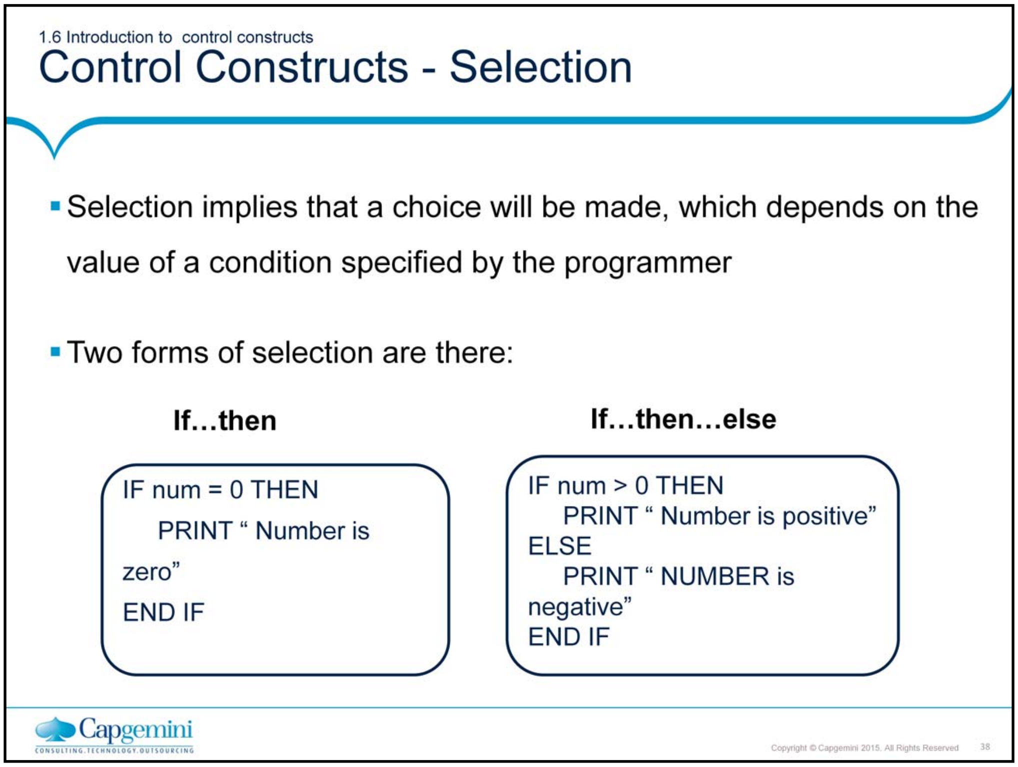
big = num1 > num2 ? num1 : num2 ;

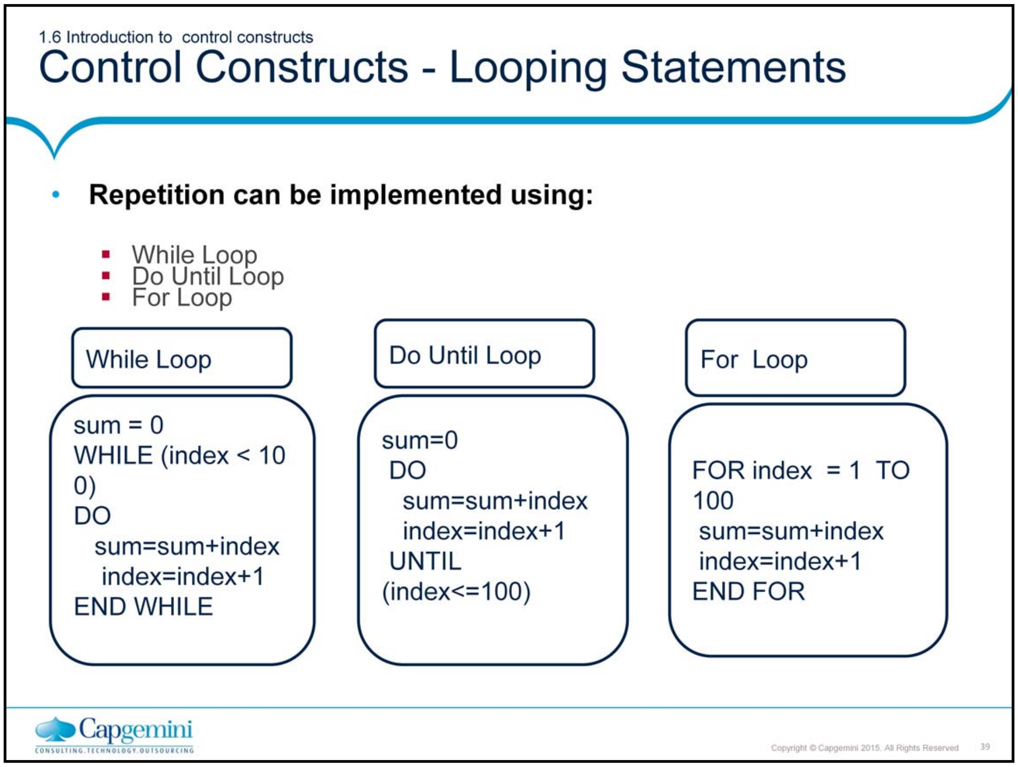
assigns value of num1 to big if num1 is greater than num2, else assigns the value of num2 to big.


1.6 Introduction to Control Constructs
There are basically three control constructs used to write algorithms:
Sequence: The instructions are executed in the sequence in which they appear, and the program does not skip or repeat any of the instructions
Selection: Selection implies that a choice will be made, which depends on the value of a condition specified by the programmer
Repetition: Repetition repeats a section of code while a certain condition holds true.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
36



Pseudocode given in the slide is used to find the difference between two numbers. In the above pseudocode, each instruction will be executed sequentially.


1.6 Introduction to control constructs
Control Constructs - Selection
Selection implies that a choice will be made, which depends on the value of a condition specified by the programmer
Two forms of selection are there:
If…then
If…then…else
IF num = 0 THEN
PRINT “ Number is zero”
END IF
IF num > 0 THEN
PRINT “ Number is positive”
ELSE
PRINT “ NUMBER is negative”
END IF
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
38



Control Constructs – Looping Statements

Do Until Loop: Set of statements inside the block is executed and then the condition is checked. Executed till the condition is true. Block will be executed at least once irrespective of the condition

Example:

```
seats_allocated = 0
 DO
   GET booking
   PRINT ticket
   ADD 1 to seats_allocated
UNTIL (seats_allocated < 60)
```

For Loop:Consists of set of statements that are executed for a fixed number of iterations Must specify a starting value, ending value and incrementing value

Example:

```
FOR seats_allocated  = 1  to 25
    Get booking
    PRINT ticket
END FOR
```

1.6 Introduction to control constructs

Control Constructs - Looping Statements (Contd…)

- **exit statement**
  - Used to exit the current loop before its normal ending
- **cycle statement**
  - Resumes iteration of an enclosing loop

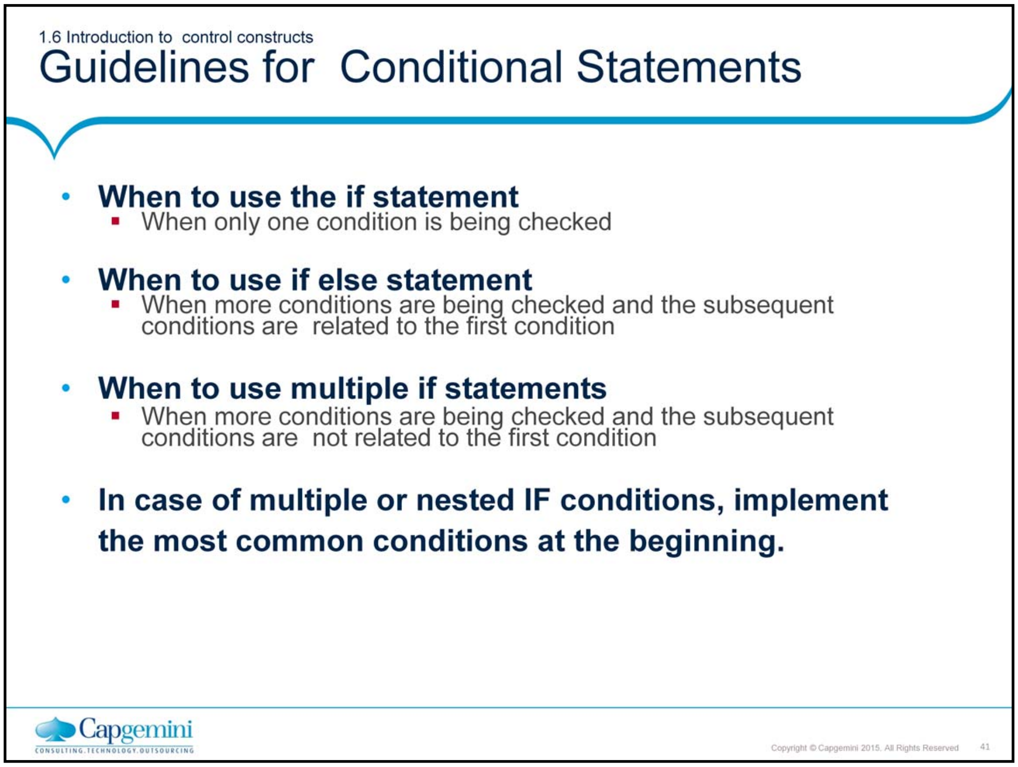
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Control Constructs – Looping Statements (contd..)

Example: Cycle and exit statement(check no is odd or even until user wants to stop)

```
BEGIN
    WHILE(TRUE)
        PRINT "Enter a NUMBER"
        ACCEPT num
        IF (REMAINDER of num/2 = 0) THEN
                  PRINT "Number is EVEN"
        ELSE
                  PRINT "Number is ODD“
        END IF
         PRINT "Enter u'r Choice , Continue with another number?[Y/N]"
         ACCEPT choice
          IF(choice='Y') THEN
                cycle
          ELSE
                exit
     END WHILE
END
```

In above example cycle statement will transfer the control to the beginning of the while loop. Exit statement will transfer the control out of while loop.



Consider Pseudocode has to be written for accepting a number from user and need to identify whether the given number is falling under any of the below mentioned category

- zero
- negative
- Positive

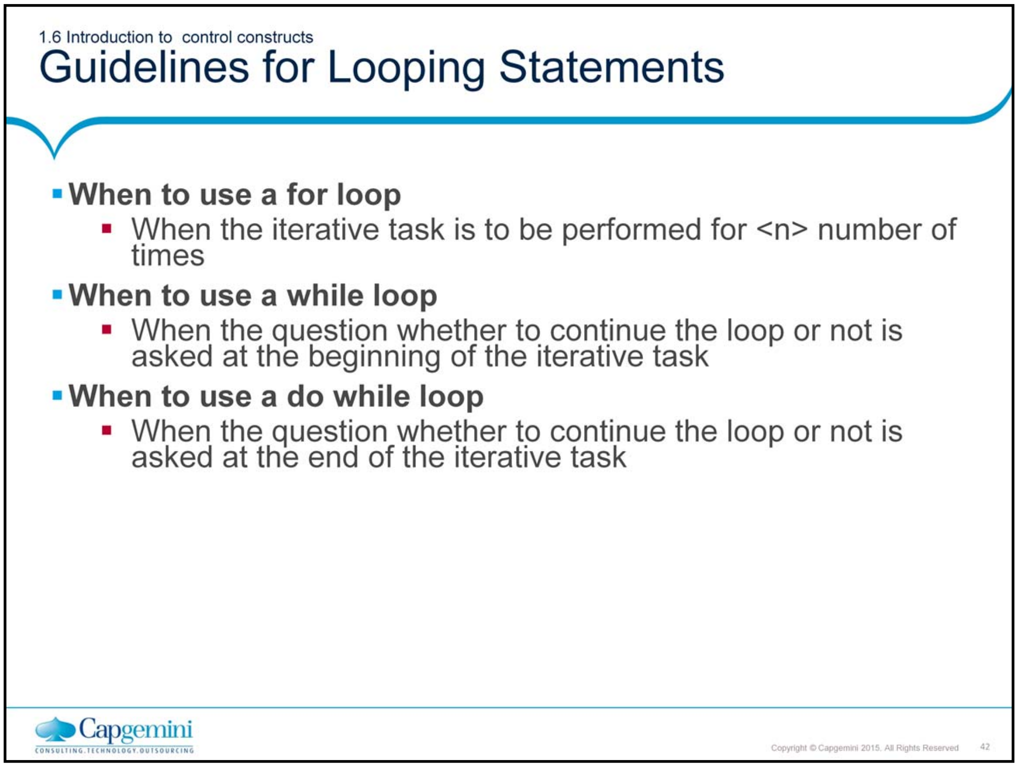
Think which condition statement is more suitable for this?

Answer: As the given number has to be compared against minimum 2 conditions, IF-ELSEIF-ELSE is more suitable

Pseudocode:

```
BEGIN
        DECLARE num AS INTEGER
        PROMPT "Enter a number" AND STORE IN num
        IF ( num== 0 )THEN
                PRINT "The value you entered was zero."
        ELSE IF ( num< 0 ) THEN
                PRINT  "The value is negative."
        ELSE
                PRINT  "The value is positive."
        END IF
END
```

I



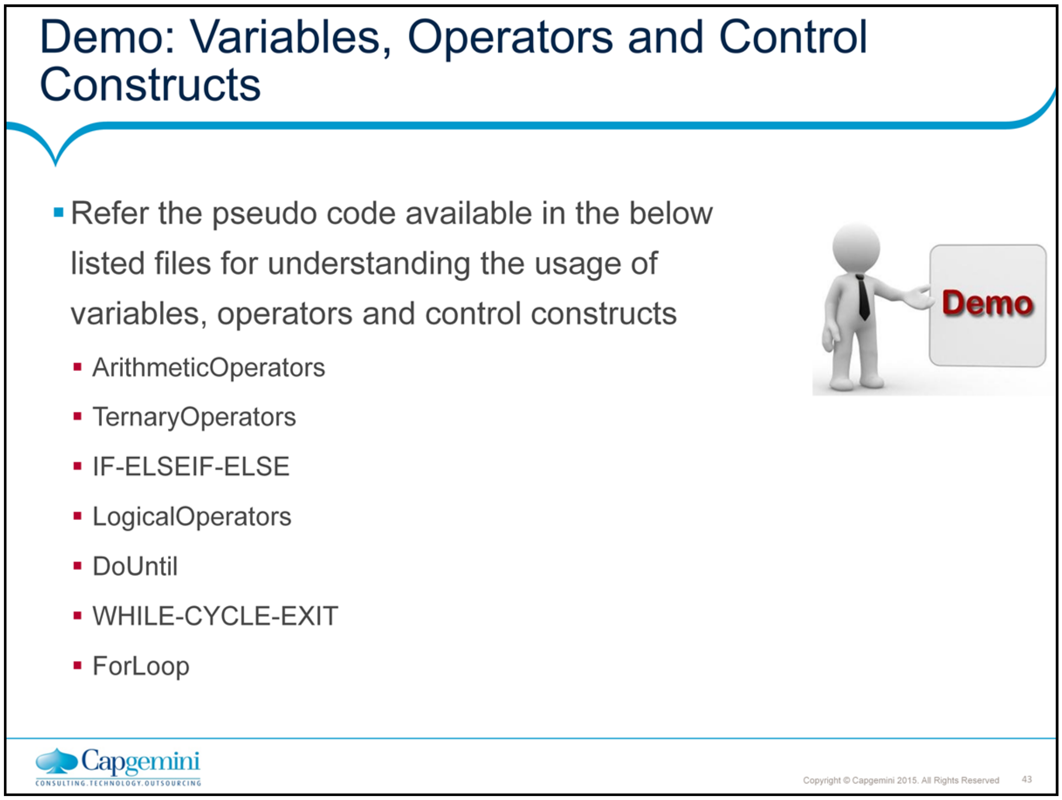
Consider Pseudocode has to be written for finding sum of n numbers by accepting “n” value from the user. For an Example, if given “n” value is 5, then sum of value is 15(1+2+3+4+5).

Think which looping statement is more suitable for this scenario?

Answer: As the given number has to be used as upper limit for process to be executed repeatedly, use for loop to describe lower limit and upper limit.

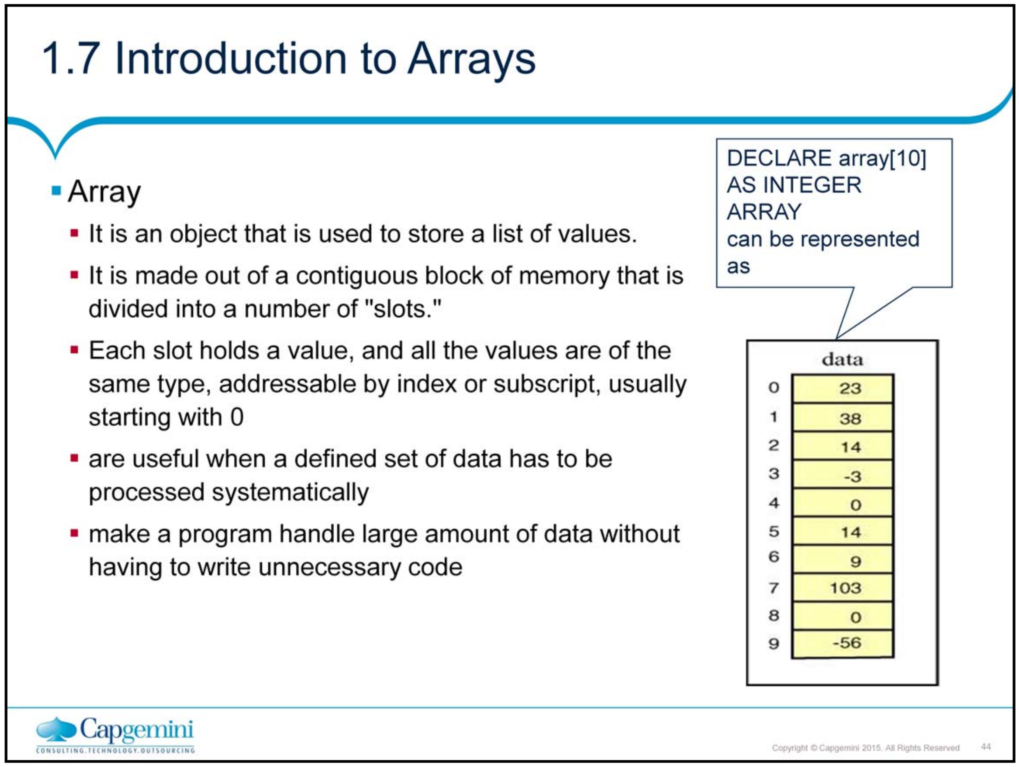
Pseudocode:

```
BEGIN
        DECLARE num, count, sum AS INTEGER
        PROMPT "Enter the value of n" AND STORE IN num
        FOR COUNT = 1 TO num
                sum+=count;
        END FOR
        PRINT "Sum is " + sum
END
```



Faculty will be sharing the below mentioned files. Refer the same to understand the usage of variables, operators and control constructs

- ArithmeticOperators.txt
- TernaryOperators.txt
- IF-ELSEIF-ELSE.txt
- LogicalOperators.txt
- DoUntil.txt
- WHILE-CYCLE-EXIT.txt
- ForLoop.txt



When to go far an Array?

- When there is a finite set of logically related information that needs similar processing
- For example calculating the grade of all students of a particular class
- All data's are similar and of one data type

Operations on an array:

The frequent operations which we perform on an array are

- Searching: searching the array for particular element
- Sorting : arranging data in particular order (ascending/descending)

1.7 Introduction to Arrays

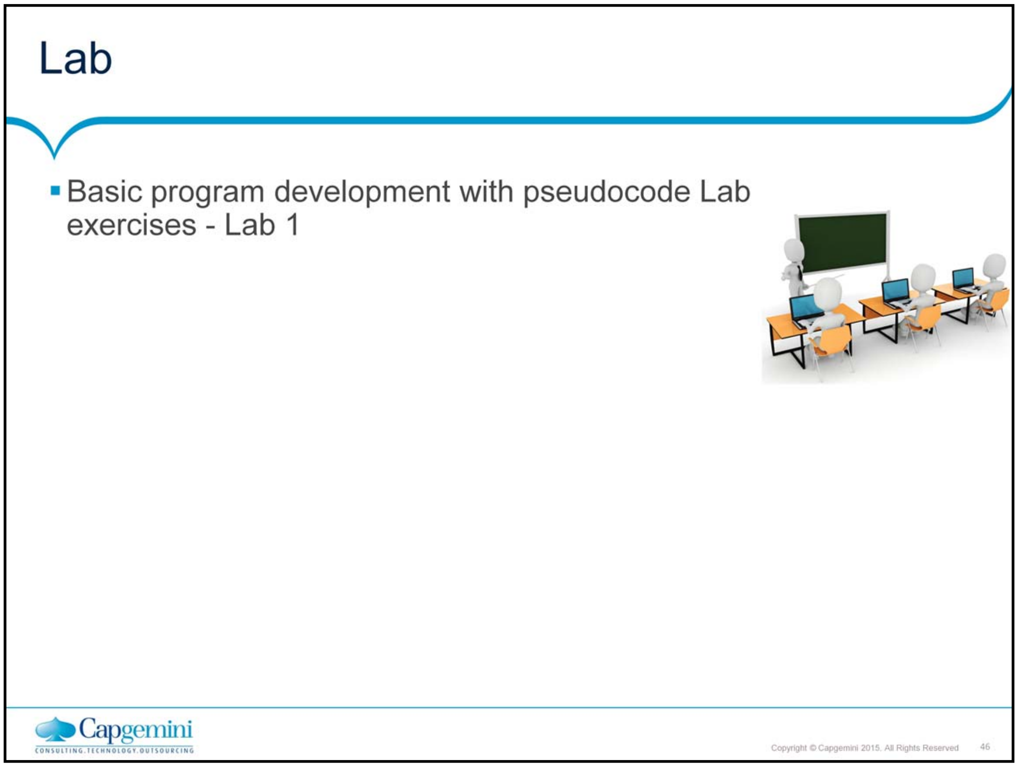
Example of Arrays

- Find out the maximum number among 10 numbers

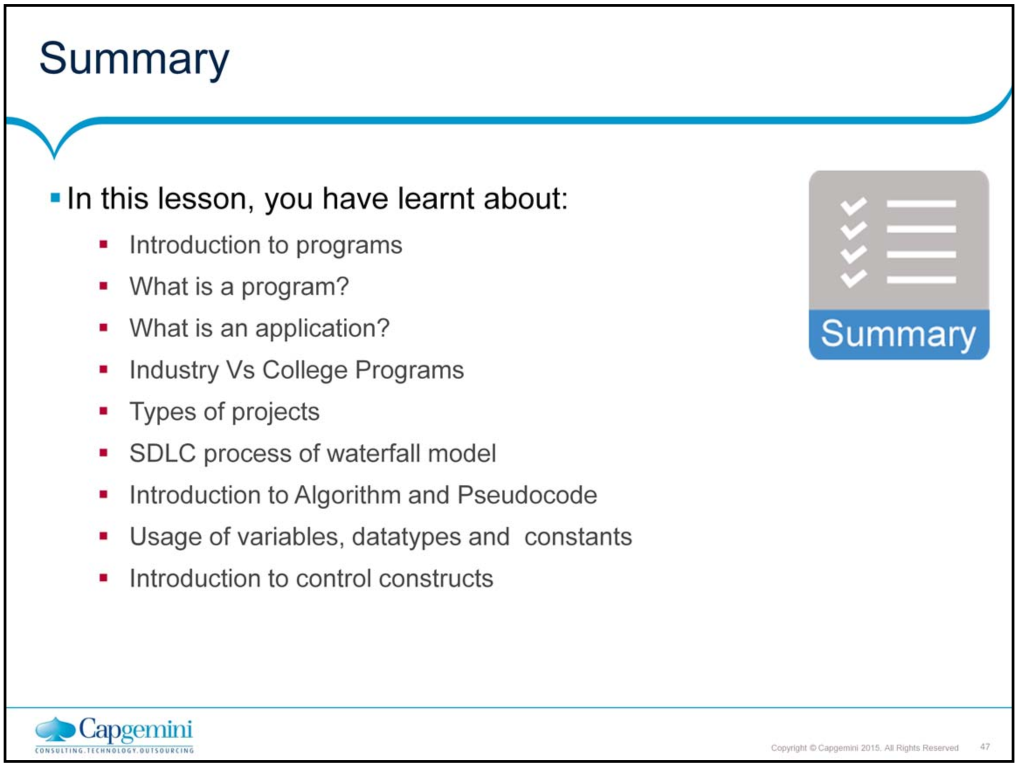
```
BEGIN
      DECLARE numbers[10] AS INTEGER ARRAY
      DECLARE max AS INTEGER
      INITIALIZE max TO 0
      FOR index=1 TO 10
            ACCEPT numbers[index]
      END FOR
      max=numbers[0]
      FOR index=1 TO 10
            IF numbers[index] > max THEN
                  max=numbers[index]
            END IF
      END FOR
      PRINT max
END
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

The above pseudocode logic is used to accept 10 numbers from user and find the largest number among 10 numbers. For storing the accepted 10 numbers of same type, array is used in this pseudocode.



Write pseudocode for all the assignments mentioned in Lab 1 for practicing on the usage of variables, operators, control constructs and arrays.


Summary
In this lesson, you have learnt about:
Introduction to programs
What is a program?
What is an application?
Industry Vs College Programs
Types of projects
SDLC process of waterfall model
Introduction to Algorithm and Pseudocode
Usage of variables, datatypes and constants
Introduction to control constructs
Summary
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
47

Review Question

- Question 1: What are different types of testing techniques?
  - A. Self testing
  - B. Black box testing
  - C. Red box testing
  - D. None of the above

- Question 2: A task which gets done in the specified time with desired quality defines ------
  - A. Maintainable
  - B. Efficient
  - C. Robust
  - D. Readable

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 48

Review Question

- Question 3: There is no absolute standard for pseudocode
  - A. True
  - B. False

- Question 4: Identify guidelines for using variables
  - A. Assigning values to variables just before values are used
  - B. Prefer local variables over global ones
  - C. Initialize variables used in looping constructs at the top of the code block
  - D. Maximize the lifetime of local variables

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 49

Review Question

- Question 5: Identify the hazardous aspects of using arrays
  - A. Empty arrays always raise errors
  - B. Errors can occur because of inappropriately defined bounds
  - C. Its easy to misuse indexes
  - D. Null values in place of empty arrays are difficult to handle

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Review Question

- Question 6: How does the following code breach best practice guidelines

```
myarray[10]= new Array{1,1,2,3,5,8,13,21,34,55};
index=0;
while(index<10)
index=index+1;
//dosomething
```

A. The array should have a clearer name

B. The index is never incremented

C. The index Is zero based so the loop passes beyond the last element

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 51

Review Question

- Question 7: Match the looping structure to their definitions

1. For	A. Condition controlled for executing choice once
2. IF	B. Condition controlled for executing choice multiple times
3. While	C. Count controlled

 52

Review Question

- Question 8: You need to create a loop whose exit condition depends solely on the maximum number of employees being reached when you increment the number of employees by one. Which construct should you choose to ensure greatest clarity

A. For

B. DO UNTIL

C. while

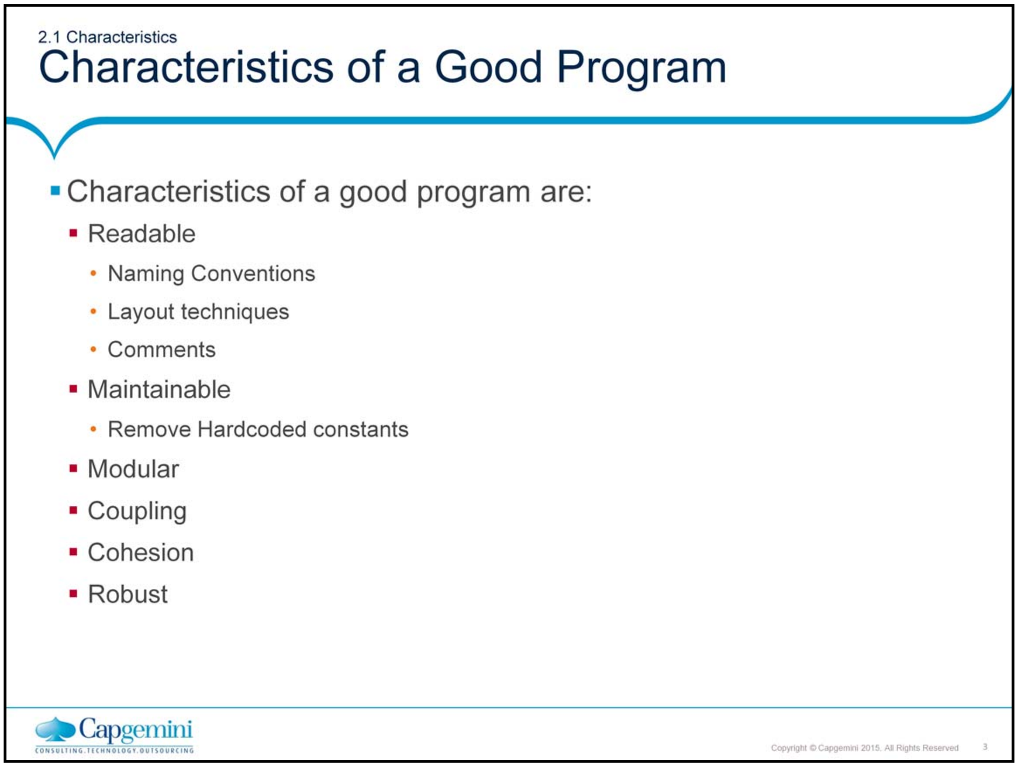
Programming Foundation

Lesson 2: Good Programming Practices

Lesson Objectives

- To understand the following concepts
  - Characteristics of a good program
    - Readable
    - Maintainable
    - Modular
  - Guidelines for writing good code
  - Coupling and Cohesion
  - Robust program
    - Difference between correctness and robustness

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



If any program has following features then it is called as good program

1. Readable

If the names of variables are meaningful and the code is easy to understand

2. Maintainable

If the program is easy to understand and If it is easy to modify then the program is called as maintainable

3. Modular

A small unit of code for a single purpose

4. Coupling

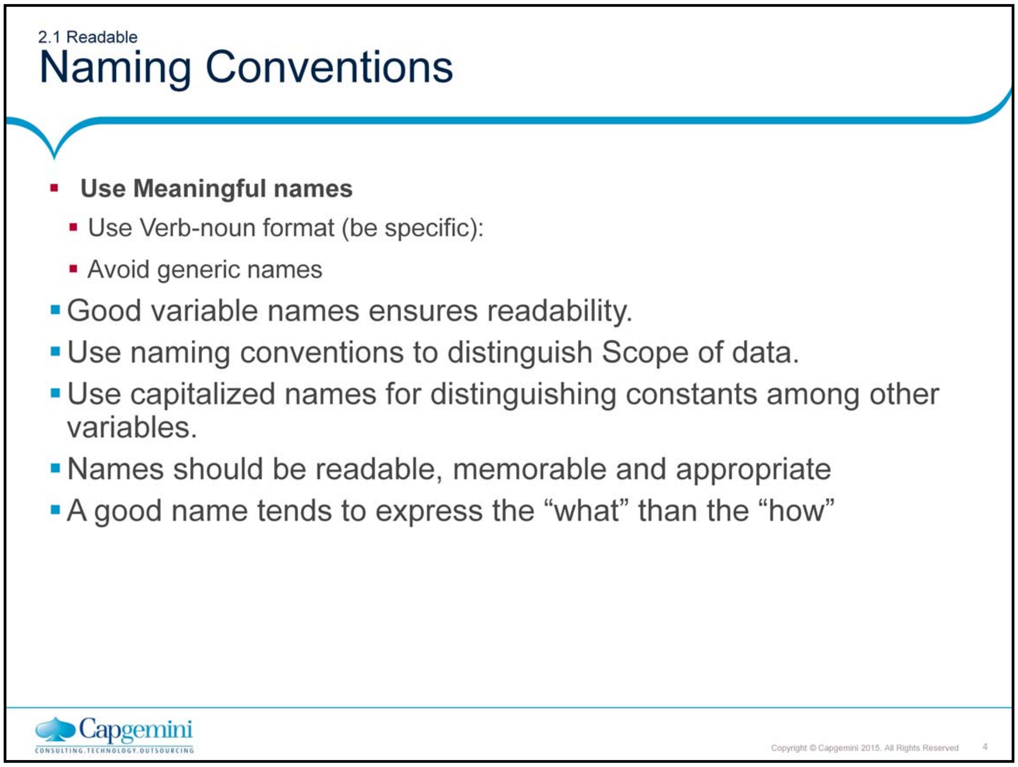
Coupling or Dependency is the degree to which each program module relies on each other.

5. Cohesion

A cohesion is a measure of how the activities within a single module are related to one another.

6. Robustness

A program is said to be robust when it is fault tolerant. You should test your program extensively with positive and negative test conditions to prove its robustness



Naming Conventions

- Meaningful Names:
  - The name fully and accurately describes what the variable represents
  - The name should refer to the real-world problem rather than to the programming-language solution
  - Use Verb-noun format (be specific):

    For example: Read-Employee-Record, Calculate-Deductions, Print-Pay-slip
  - Avoid generic names:

    For example: Process-inputs, Handle-calculations
- Good variable names are a key element of program readability.
- Use naming conventions to distinguish Scope of data like local/global.

  For an Example, MAX_USERS_G variable scope is global
- Use capitalized names for distinguishing among type names, named constants, enumerated types, and variables.

2.1 Readable

Naming Conventions (Contd…)

- Example of Poor Variable Names:

```
X          =  X – XX;
marypoppins = (superman + starship) / god ;
X          = X + Interest( X1, X );
```

- Example of Good Variable Names:

```
Balance  = Balance – LastPayment;
Balance  = Balance + Interest ( CustomerID,Balance);
```

2.1 Readable

Naming Conventions (Contd…)

Variable Type	Strategy	Variable Name	Example
Temporary Variable	**Bad**	**Temp**	**Temp = sqrt(b^2 – 4 *a*c);**
	Good	**Discriminent**	**Discriminent== sqrt(b^2 – 4 *a*c);**
Boolean Variable	**Bad**	**NotFound,NotSuccess**	**If (Found)** **PRINT “ELEMENT IS PRESENT”** **else**
	Good	**Found , Success**	**PRINT “ELEMENT IS NOT PRESENT”**
Status Variable	**Bad**	**StatusFlag = 0x80.** **DataReady=r**	
	Good	**CharacterType = CONTROL_CHARACTER** **DataReady = TRUE;**	

Naming Status Variable: Think of a better name than “flag” for status variables.

if (DataReady) …..

if (CharacterType & PRINTABLE_CHAR) …

if (ReportType == AnnualRpt) …

DataReady = TRUE;

CharacterType = CONTROL_CHARACTER ;

ReportType = AnnualRpt;

RecalcNeeded = FALSE;

CharacterType = CONTROL_CHARACTER is more meaningful than StatusFlag = 0x80.

Naming Temporary Variable: Use meaningful, descriptive names for Temporary variables. Don’t use Temp, x or some other vague name.

Bad Temporary variable Name

Temp = sqrt(b^2 – 4 *a*c);

A Better approach is

Discriminant = sqrt(b^2 – 4 *a*c);

Never use a numeric suffix to differentiate variables

2.1 Readable

Optimum Name Length

- The name is long enough that you don't have to puzzle it out.

Too Long	NumberOfPeopleOnTheUSOlympicTeam, NumberOfSeatInTheSaddleDome
Too Short	N, NP, NTM, M, MP, Max, Points
Just Right	NumTeamMembers, TeamMbrCount, MaxTeamPoints, RecordPoints, cPoints

Naming Boolean Variables: Use names that imply true or false like done, error, found, success as boolean variables

Avoid using negative names like NotFound, NotDone and NotSuccessful

"If not notFound" is difficult to read

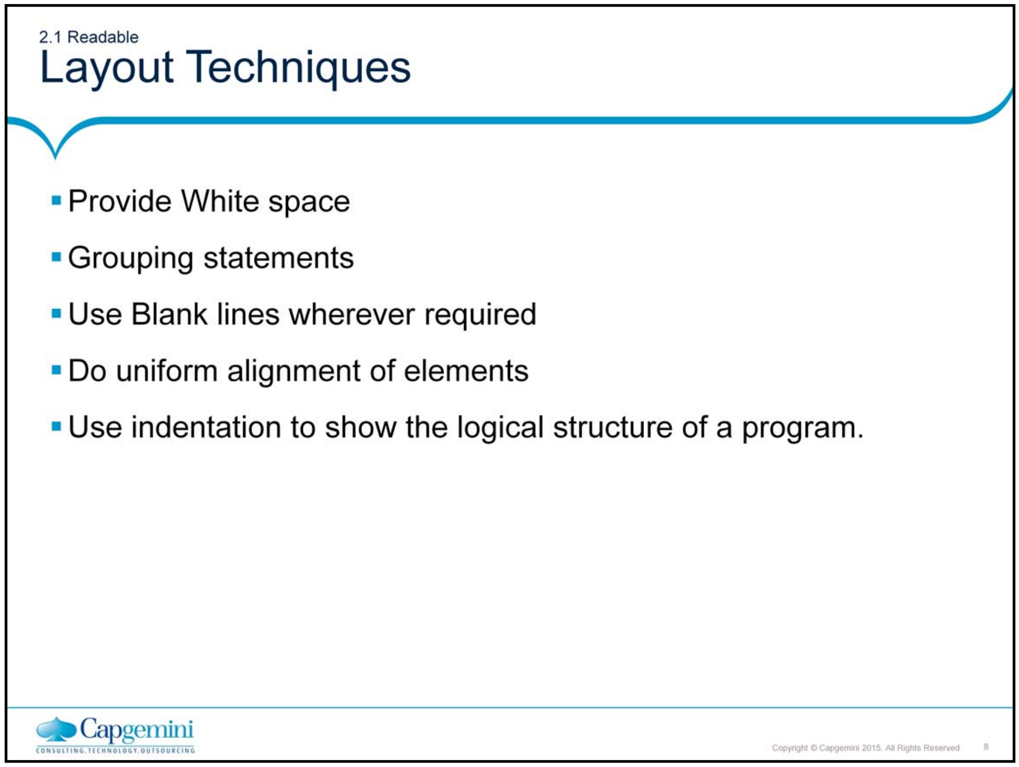
Better way is to use "if found" instead

Give meaningful names for Loop index.

```
i = 0
ACCEPT Emp, Basic
G = B * 1.8 + 1700
PF = 0.12*B
T = ((G*12 - 150000)*0.3 + 19000)/12
N = G - PF - T – 200
PRINT Emp, B, G, PF, T, N
i=i +1
IF i==10  THEN Stop
Continue
```

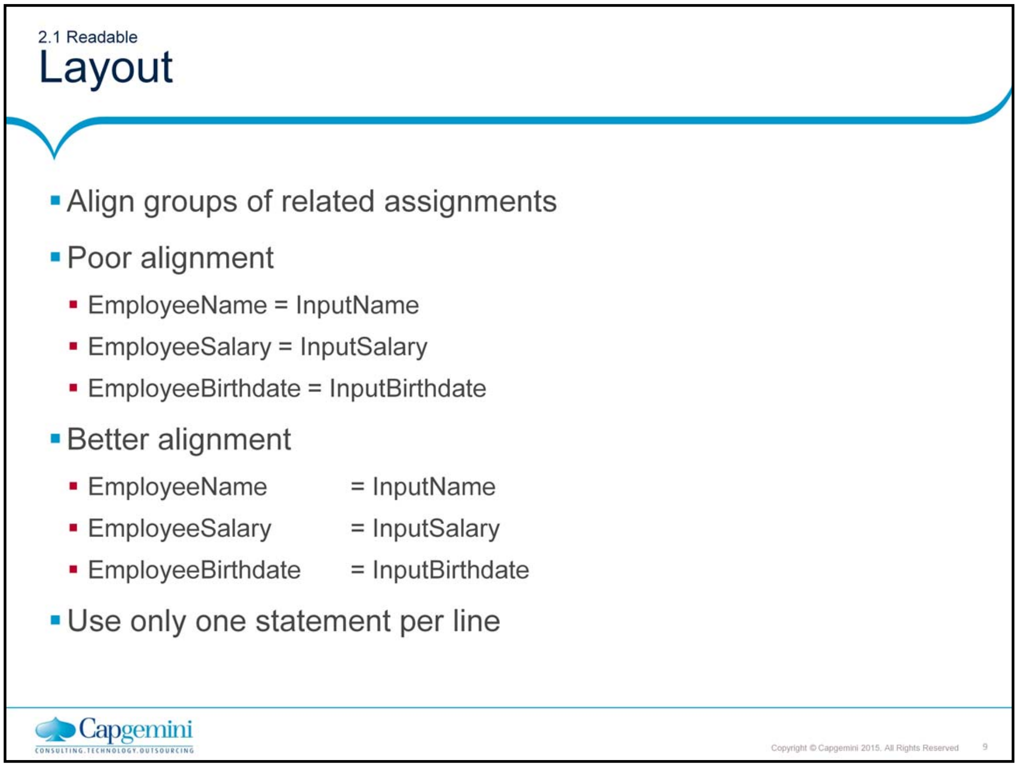
In this example variable 'i' is used to count how many pay slips have been generated.

'Rec_Count' can be used instead of 'i'



Layout Techniques :

- Use white space to enhance readability.
    - Use vertical and horizontal whitespace generously.
    - Indentation and spacing should reflect the block structure of the code.
    - A long string of conditional operators should be split onto separate lines
- Grouping: ensure related statements are grouped together.
- Blank lines: separate unrelated statements from each other. Use blank lines to divide groups of related statements into paragraphs, to separate routines from one another, and to highlight comments.
- Alignment: align elements that belong together.
- Indentation: use indentation to show the logical structure of a program



Layout :

- Use “=“ in a uniform positions in a group of statements.
- Use only one statement per line
- Within a block, Provide tabspace to start with a statement.

2.1 Readable

Layout (Contd…)

- Example of bad layout : unformatted complicated expression

```
IF((‘0’  <= InChar and InChar <= ‘9’) or (‘a’ <= InChar and
      InChar <= ‘z’) or (‘A’ <= InChar and InChar  <= ‘Z’ )) THEN
```

- Example of a good layout : readable complicated expression

```
IF (( ‘0’  <= InChar and InChar <= ‘9’ ) or
     ( ‘a’ <= InChar and  InChar <= ‘z’ ) or
    ( ‘A’ <= InChar and InChar  <= ‘Z’ ) ) THEN
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Layout Techniques:

- Statement belongs to if/loop should be started after a tab space
- A long string of conditional operators should be split onto separate lines. For example the below if statement

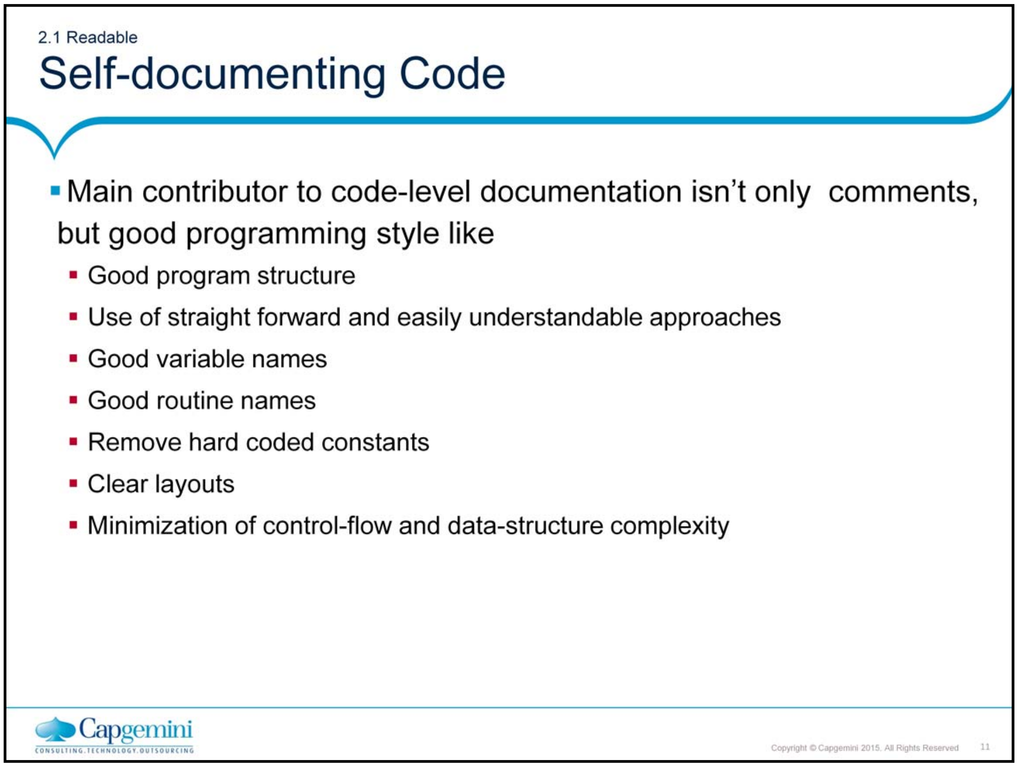
```
if (foo->next==NULL && number < limit && limit
        <=SIZE    && node_active(this_input)) {...
```

might be written better as:

```
if (foo->next == NULL
    && number < limit && limit <= SIZE
   && node_active(this_input))       {        ...
```

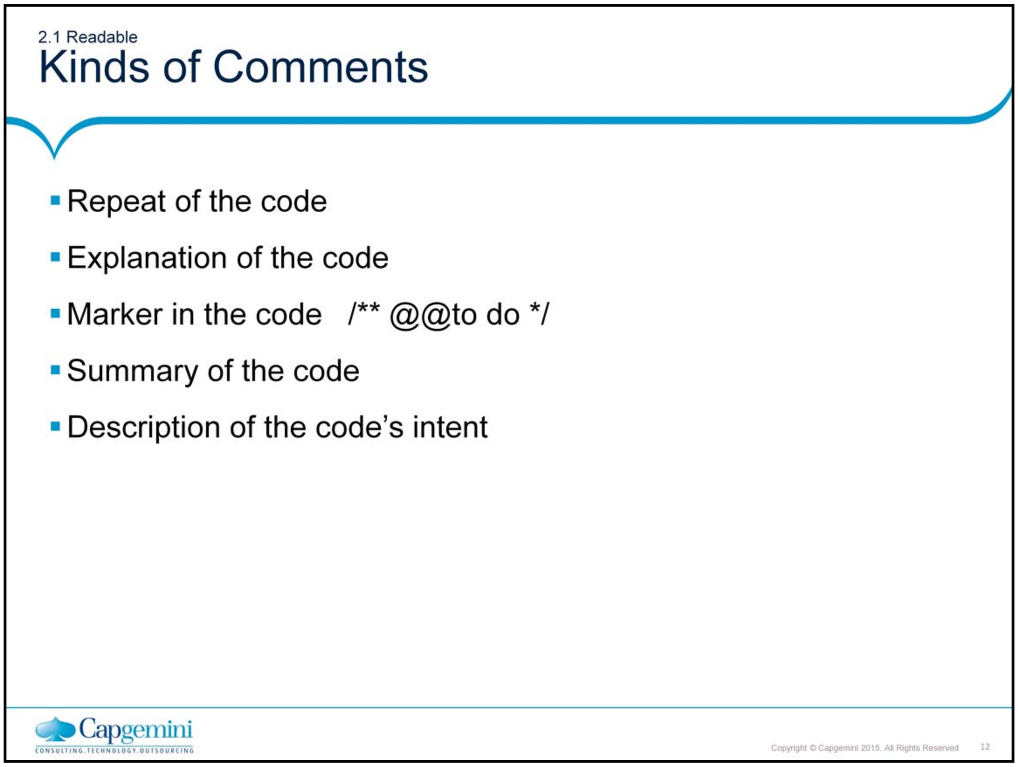
Better version of using For LOOP: Consider for loop have 3 sections

```
FOR (curr = *varp, trail = varp;
     curr != NULL;
     trail = &(curr->next), curr = curr->next )
{
```



Self documenting code:

- Good Program Structure: Follow program structure like modularity
- Use of straight forward and easily understandable approaches.
- Good variable names : Meaningful variable names
- Good routine names : Meaningful routine names
- Remove hard coded constants: Use of named constants instead of literals
- Clear Layout : Follow all the layout techniques



Kinds of comments

Repeat of the code:

A repetitious comment restates what the code does in different words. It merely gives the reader of the code more to read without providing additional information

Explanatory Comments:

Explanatory comments are typically used to explain complicated, tricky or sensitive pieces of code. If the code is so complicated that it needs to be explained, its nearly always better to improve the code than it is to add comments. Make the code itself cleared and then use summary comments

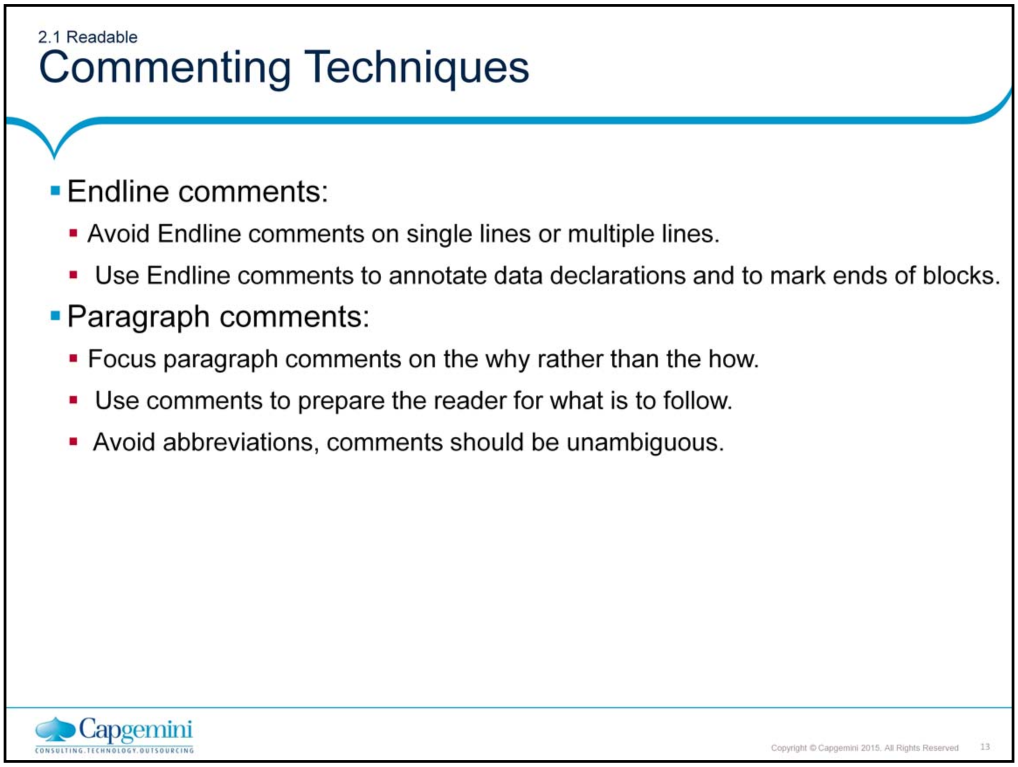
Marker in the code /** @@todo*/ :

todo is a form of comment used to convey that the code is yet to be completed by replacing todo comment with the functionality logic. For an example, /**@@todo*/

Summary of the code : Use comment to provide description of the program like header block comment. For an example,

//Program Description:

Description of the code's intent : Use this comment, to describe the layout used



Commenting Techniques

Commenting is amenable to several different techniques depending on the level to which the comments apply : program, file, routine, paragraph, or individual line

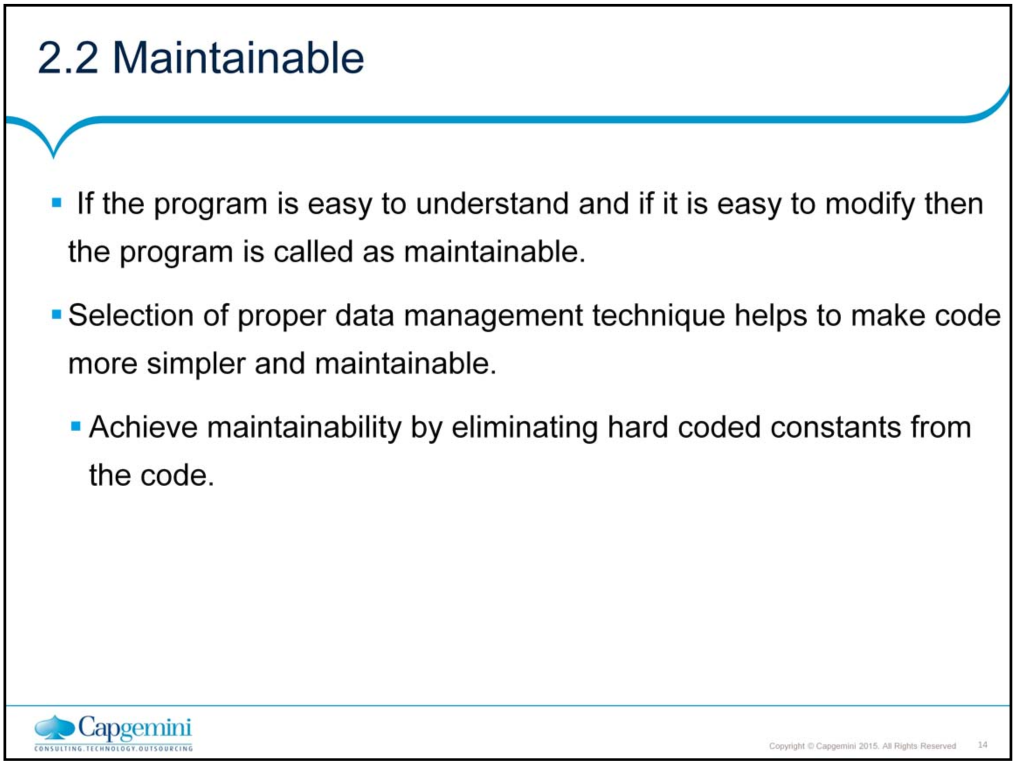
- Endline Comments
    - Endline comments are comments that appear at the ends of lines of code
    - Use endline comments to annotate data declarations like

    Declare index as integer and store 0. //upper index of an array

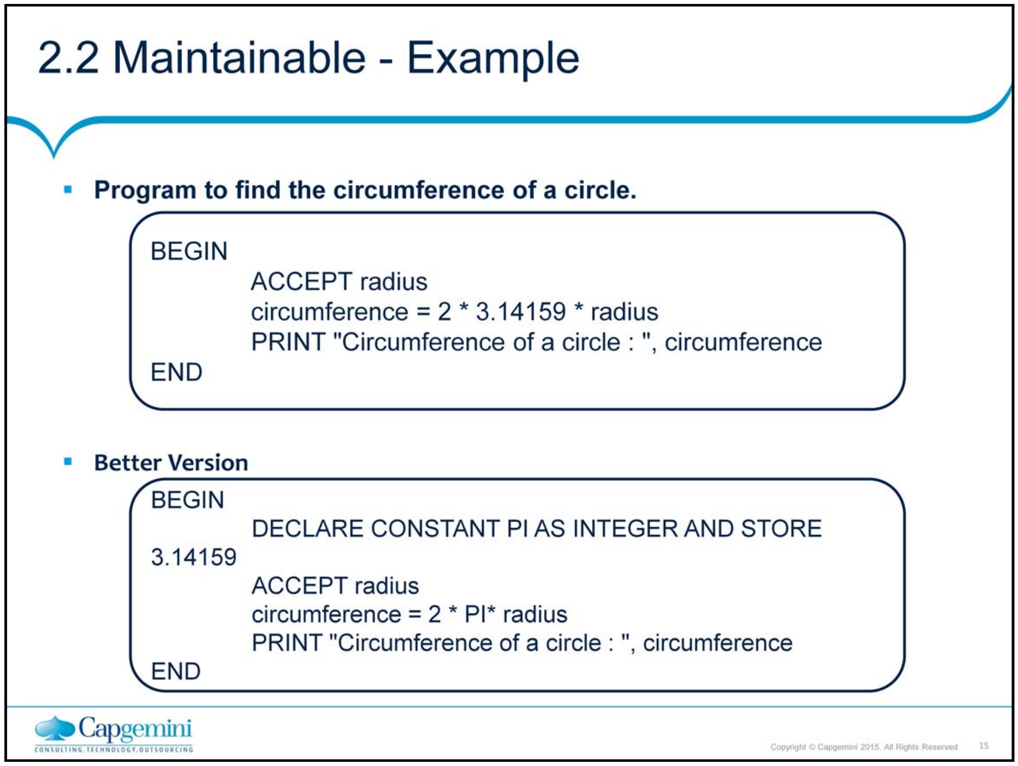
    - Use endline comments to mark ends of blocks

- Paragraph Comments
    - Most comments in a well documented program are one sentence or two sentence comments that describe paragraphs of code
    - Use to describe the purpose of the block of code
    - For an example,

```
/*******************************
*       Search for an employee
* ******************************/
```

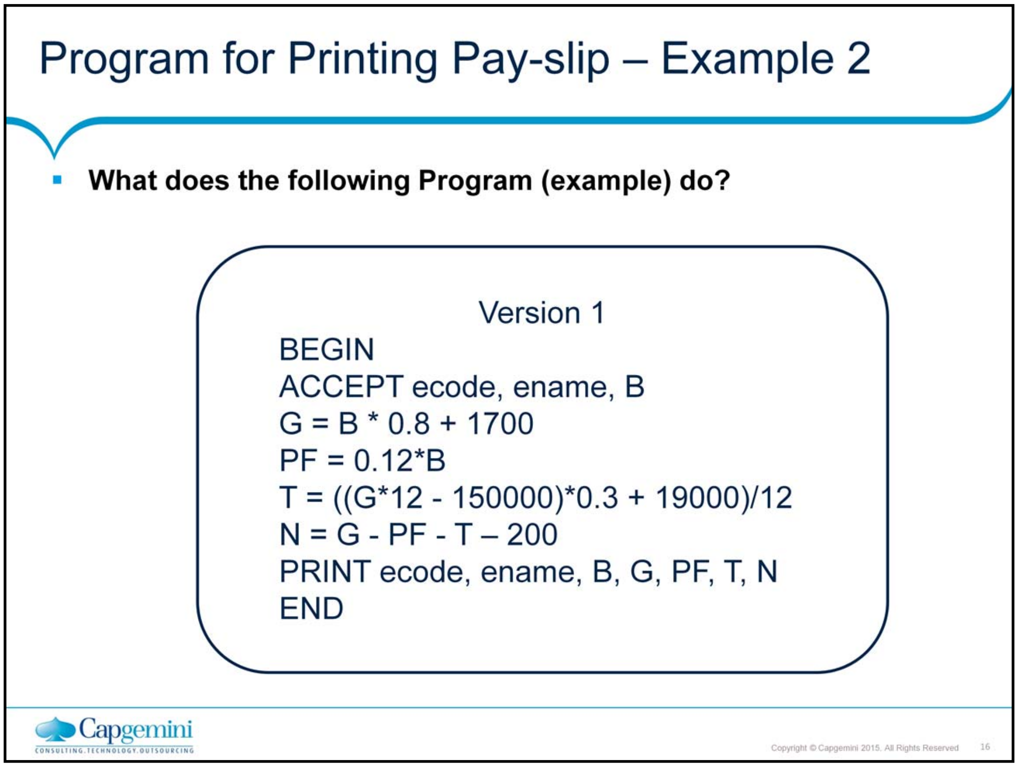


If the program is easy to understand and if it is easy to modify then the program is called as maintainable. Selection of proper data management technique helps to make code more simpler and maintainable. Achieve maintainability by eliminating hard coded constants from the code.



Example:

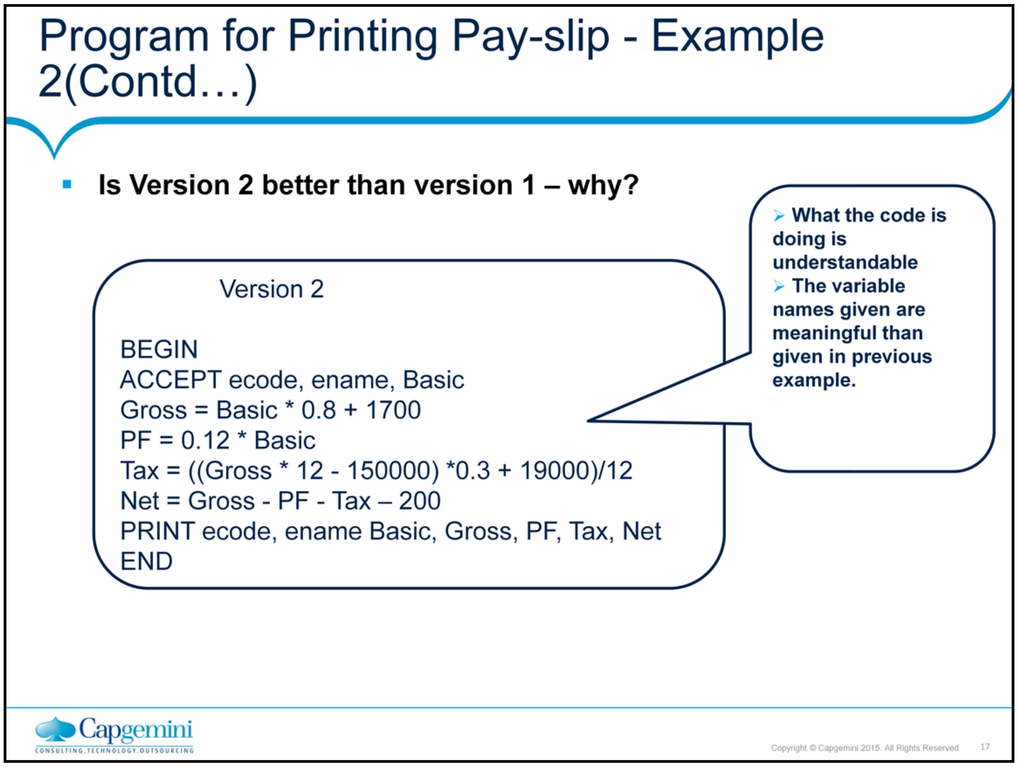
The given program is used to find the circumference of a circle based on radius. In the given code, hardcoded constant exists which is eliminated by using hard coded constant.



What are problems in the code above:

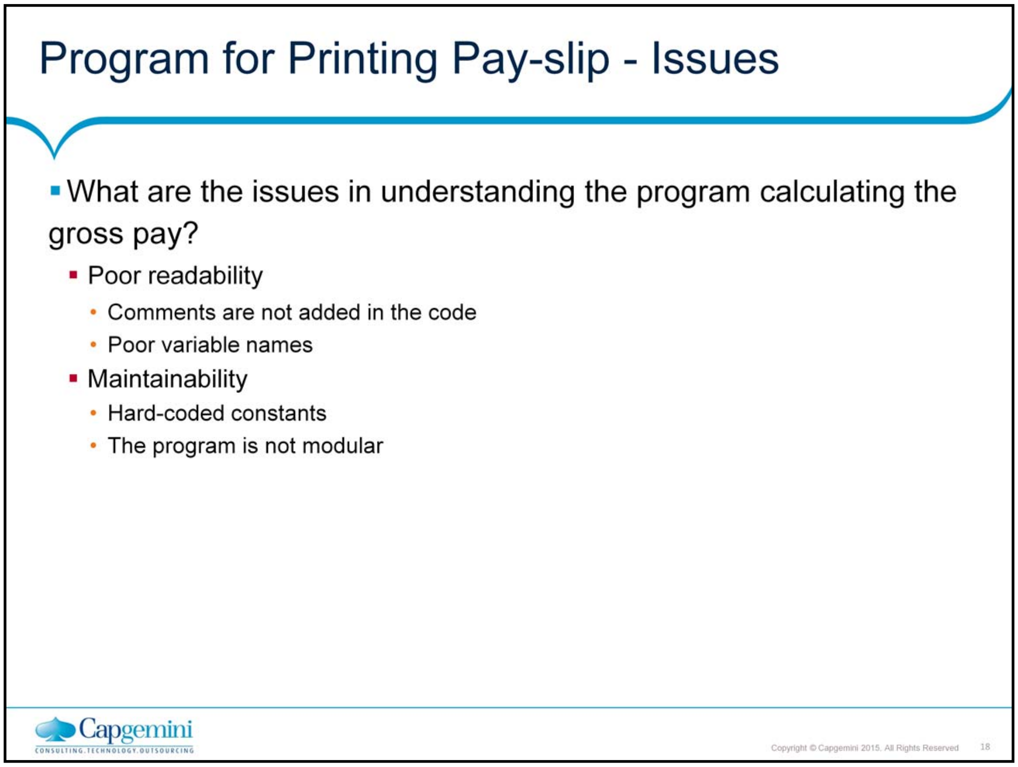
Variable names are not meaningful. Understanding meaning of variables G, B, T, N etc is difficult. Hence the code is not easily understandable

We don’t understand what the given code is doing?



See the above example what it is doing?
It reads employee code, employee name and basic salary from keyboard.
Calculates Gross pay, Provident Fund (PF), Tax and Net pay.
Prints the pay slip

It is understandable because the variable names given are meaningful than given in previous example.
It shows that if you give meaningful variable names then it helps you to understand program better.



Programming standards that are to be followed are given below:

- Use meaningful variable names. It helps to easily understand the program. Avoid using hard-coded constants in the program. Instead, declare them as constants at the beginning of the program, and use these constant names through the program. If value of constant changes later, the value can be modified for the declared constant at one place. The effect will be visible everywhere this constant name is used.
  For example: tax rate

- Include comments at the header and module level
- Don’t use literal values in the program instead create constant variable to store fixed literal value for making program to be more maintainable.
- Ensure that you have created more modular program.

Program for Printing Pay-slip - Solution

- What solutions do you recommend for these issues?
  - Use Header block for comments.
  - Use meaningful variable names.
  - Eliminate hard coded constants from the code.
    - Avoid use of obscure code

      For an example:

      ```
      HRA = 0.5 * Basic   /*** avoid obscure code G = B * 0.8 + 1700 ***/
      OPA = 0.3 * Basic   /*** Offshore project allowance ***/
      Conveyance = 1700
      ```
  - Use comments to describe program flow or complex sections of code.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

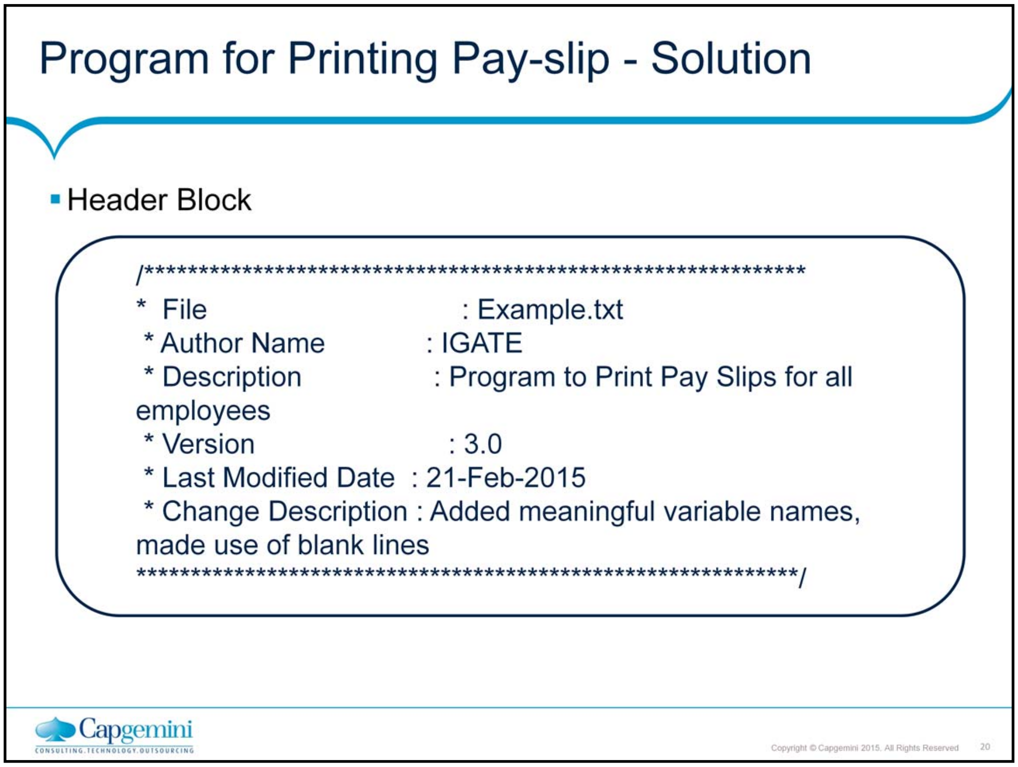
 19

If we use above solution to update the code then the code will become more readable, maintainable.

By reading the code, G = B * 0.8 + 1700, we do not understand what is meant by 1700, nor what is meant by 1.8 in the gross calculation.

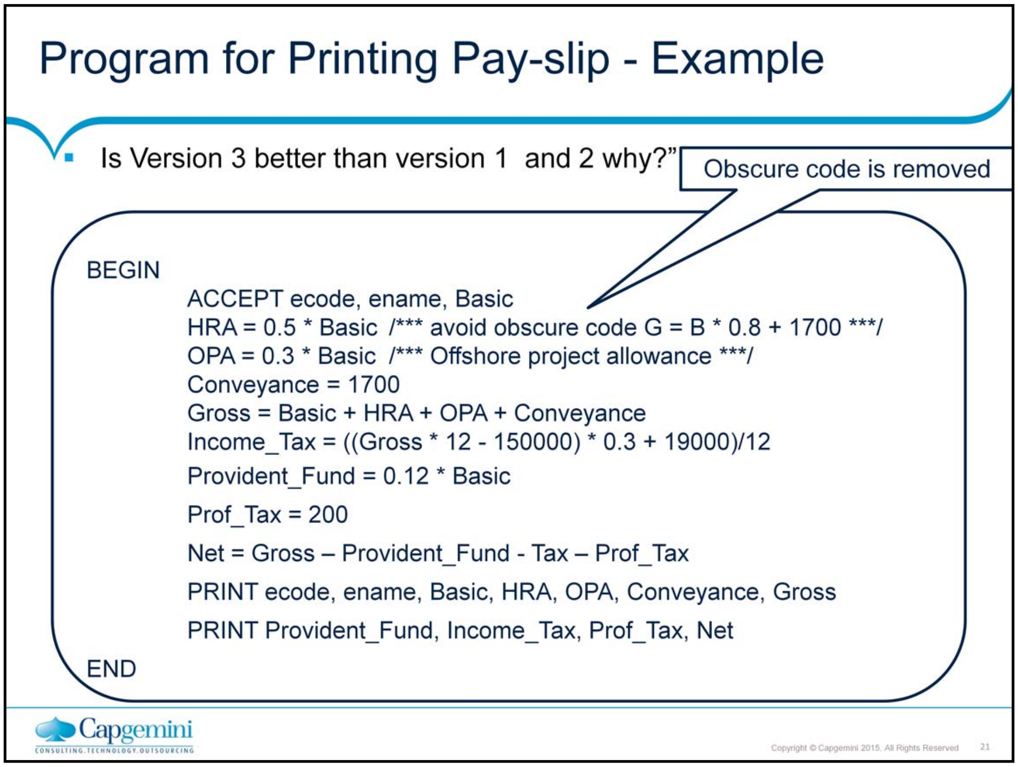
However, the given code explains that Conveyance is 1700. HRA is 50% of Basic. Offshore Project allowance is 30% of Basic. So do not try to club these equations. Otherwise, maintenance of the code will be difficult, and understandability of code will reduce, as well.

Hence avoid such obscure code. Keep it simplified.



See the given header block of comments
It includes:

- File : Give the name of the file.
- Author name : Provide the author name who involved in the development of program
- Description : Detailed description of the program
- Version : Version of the program
- Last modified date : Date on which the program is last modified
- Change Description: History of changes happened in the program



Why version 3 is better than version 1 and 2

1. Hard coded constants are given proper names. E.g – HRA, OPA, Conveyance.
2. The given code is easy to maintain because if conveyance amount , percentage for HRA or OPA changes then we can easily understand where to make the change in the code, which was difficult in version 1 and 2
3. Code is readable as naming conventions are followed and layout is applied

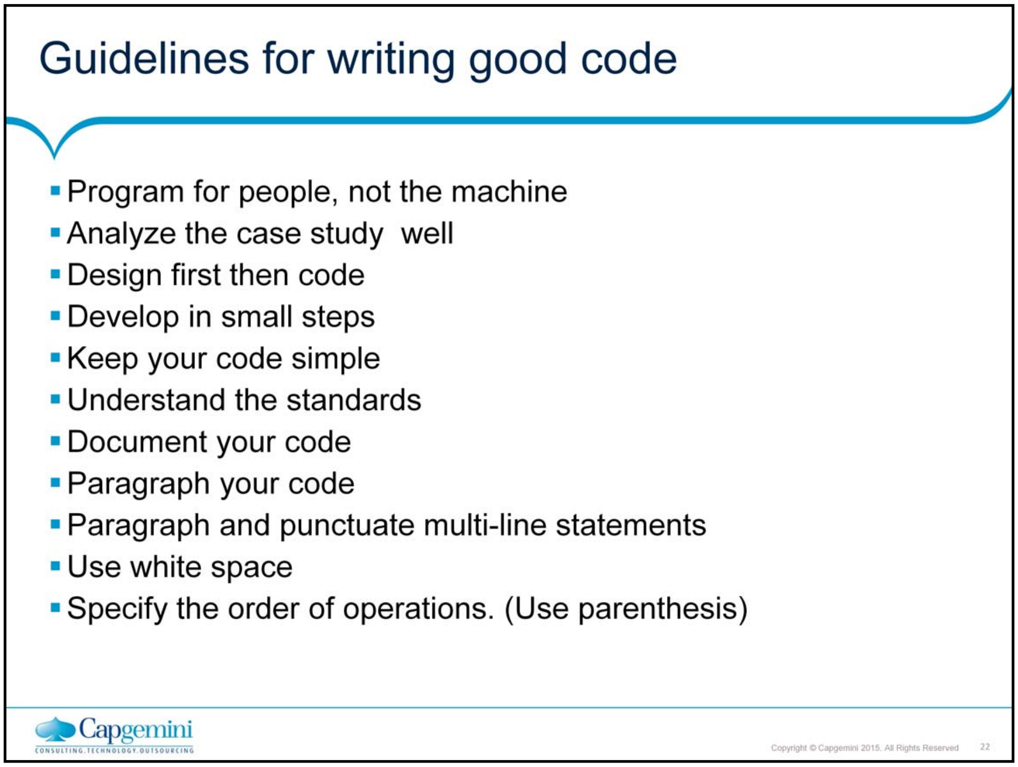
Following improvements are required in the code:

a) If your code includes any complicated calculations It is necessary to document it in the code. In the above example the calculation of income tax includes many operations. What is meaning of it need to be documented in the code.

Steps involved in income tax calculations

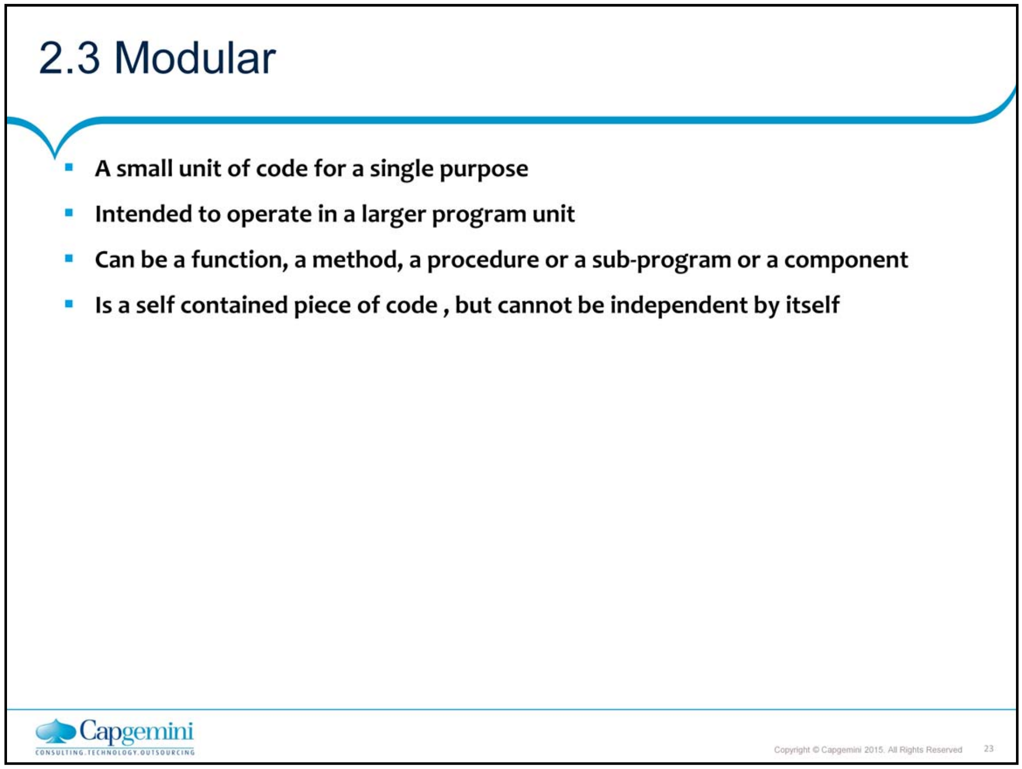
1. Calculate annual salary : Gross * 12
2. Calculate taxable amount: 1,50,000 will be subtracted from annual salary
3. Calculate annual tax: Annual Tax = 30% of taxable amount + 19000
4. Calculate monthly tax Monthly tax = Annual tax/12

b) The given code is not modular. It doesn't have any modular structure.
If the code is huge. It is performing various functions then it is better to create separate module for each function which increases reusability of the program.
If any of these modules can be reused in some other application Programmer's efforts and time will be saved.
A good example is login module. Almost every application requires login screen which authenticate users. Such modules can be written once and used in multiple applications.

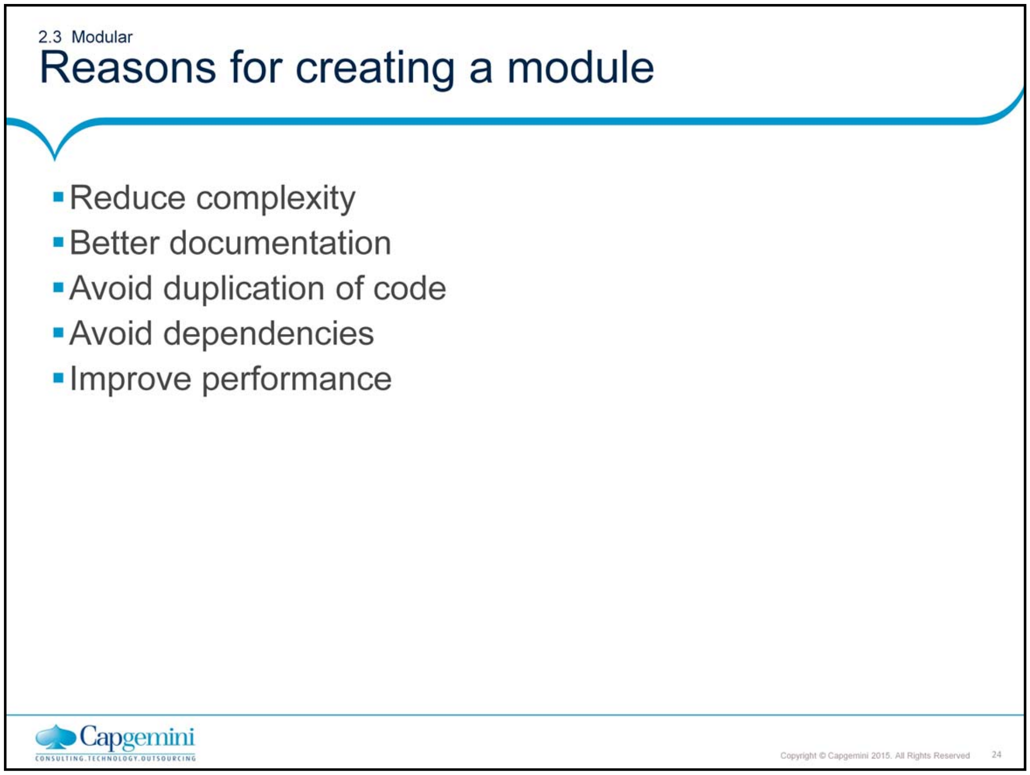


Guidelines for writing good code

- While writing program, keep in your mind that program will be used by people, so make program to be user friendly.
- Before starts with development , analyze the case study well to incorporate all the requirements.
- Do the high level (like database design) and low level designing(pseudocode) of an application before working in coding phase.
- Create program in an incremental approach, so that after implementation of each logic it can be easily tested for finding defects.(As finding defects in earlier stage, decreases cost and save development time).
- Make your code to more simple by avoiding the usage of complex data structure/constructs
- Understand the standards:
    - All the coding standards as well as processed should be understandable and apply the same in your code.
    - Believe in them
    - Make them part of your quality assurance process
    - Adopt the standards that make the most sense for you
- Document your code using comments for making it to be more readable
- Paragraph your code by applying modularity to make code reusable.
- Use whitespace as a layout technique to make code more readable
- Specify the order of operations using parenthesis for prioritizing

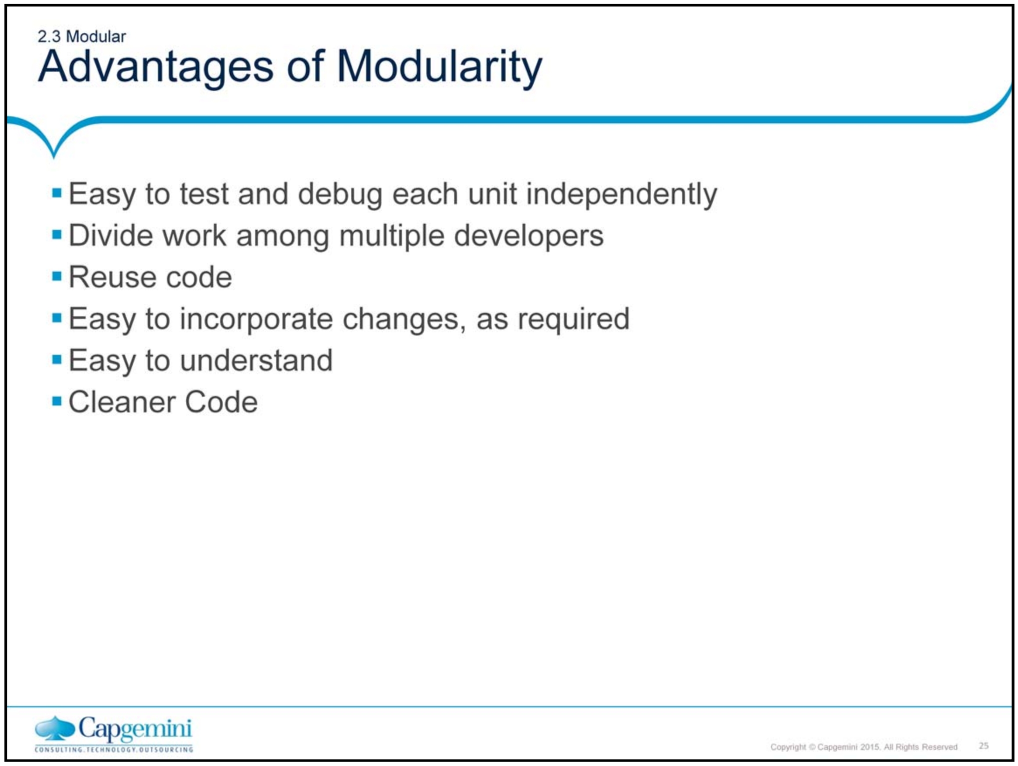


Modular : A small unit of code for a single purpose intended to operate in a larger program unit . It can be a function, a method, a procedure or a sub-program or a component. Module is a self contained piece of code , but cannot be independent by itself



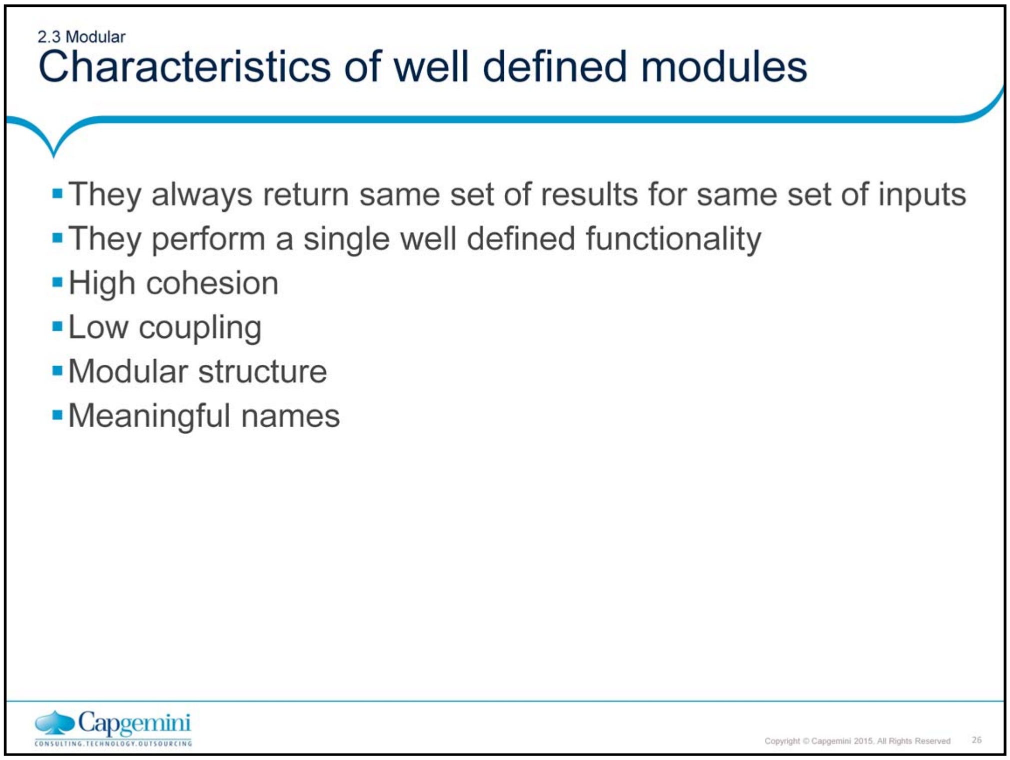
Reasons for creating a module

- Reduce complexity : By using the abstracting power of modules, complex code can be made to appear simpler and easy to understand
- Better documentation : By putting a set code into a well defined module, makes the code self –explanatory
- Avoid duplication of code
- Avoid dependencies : Sections of code that depend on each other makes changes difficult to incorporate
- Improve performance : Easy to test and debug units of code, than a long one



Advantages of Modularity

- Easy to test and debug each unit independently so that defects will be identified at the earlier stage.
- Divide work among multiple developers so that application development time will be reduced.
- Reuse code: Once a module is written, the same module can be reused in another application.
- Easy to incorporate changes, as required so that the code will be more maintainable.
- Easy to understand as the code is written in a single unit called module.
- Cleaner Code



Characteristics of well defined modules

- They always return same set of results for same set of inputs
- They perform a single well defined functionality
- **High Cohesion** – do one thing, and do it well
- **Low Coupling** – reduce dependencies between modules

Ideally when there is “high cohesion” and “low coupling”, one can change the implementation of a routine without impacting the call interface.

For example: A sort routine can change its algorithm from “Bubble” to “Quick sort” – without causing the calling code to break.

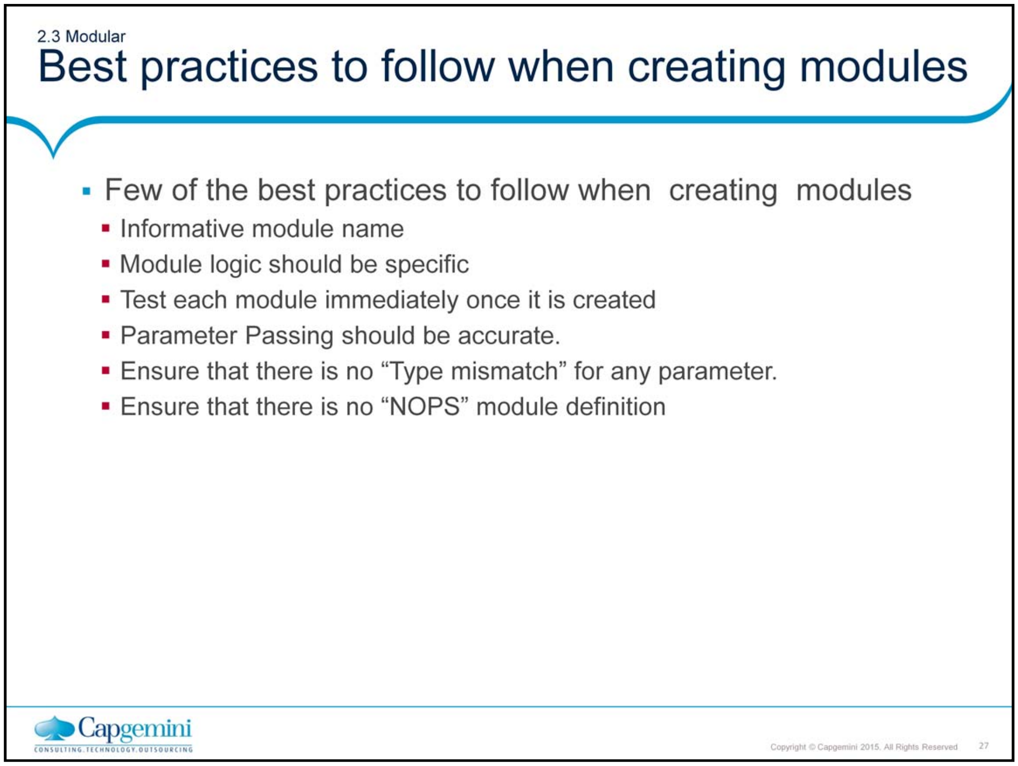
- **Meaningful names**

Use Verb-noun format (be specific):

For example: Read-Employee-Record, Calculate-Deductions, Print-Pay-slip

Avoid generic names:

For example: Process-inputs, Handle-calculations



Best Practices to follow when creating modules

- **Informative Module Name:** Give the module an informative name like getProductPrice, calculateSalary, printDetails.
- **Module logic should be specific:** **I**dentify a clear purpose for the module before you start writing
- Test each module as it is created by performing unit testing
- Parameter passing should be accurate: Ensure that the number of parameters, and the sequence is correct for the module call.
- Ensure that there is no "Type mismatch" for any parameter.
- Ensure that there is no "NOPS"(No Operation) Module definition. i.e. Empty module definition shouldn't be there.

calculateTotal(Integer price, Integer quantity)

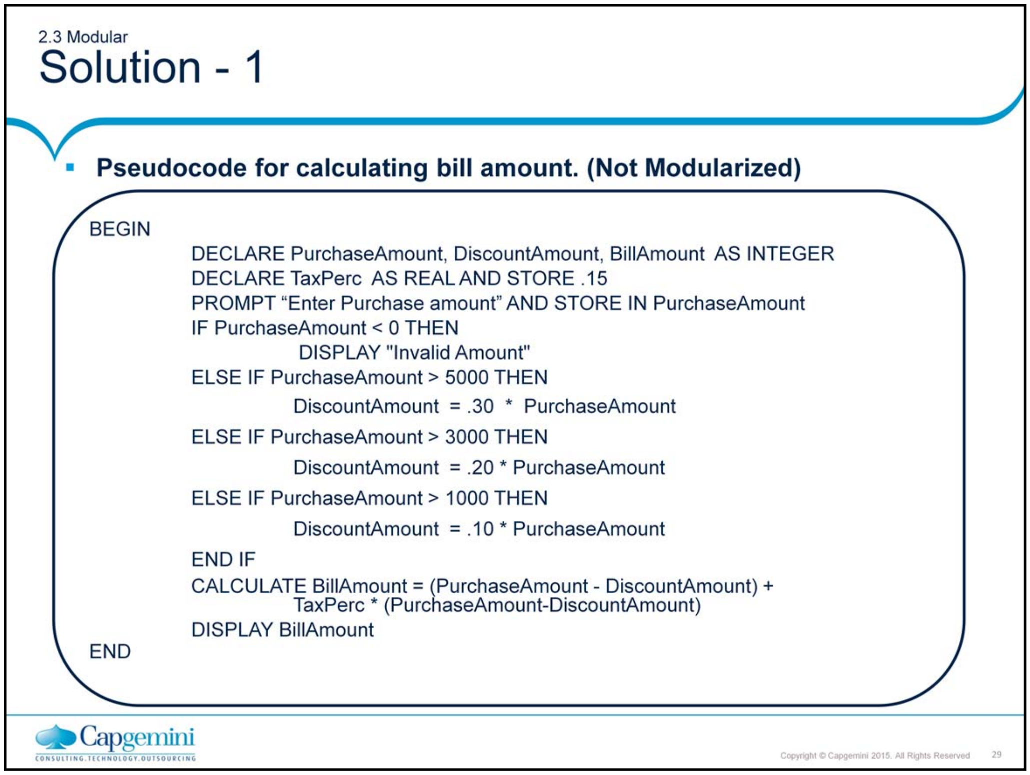
Refer the valid and invalid statements to invoke a module

- **calculateTotal(3,5); //Valid**
- **calculateTotal(4,3,4); //Invalid**
- **calculateTotal('Test',3); //Invalid**

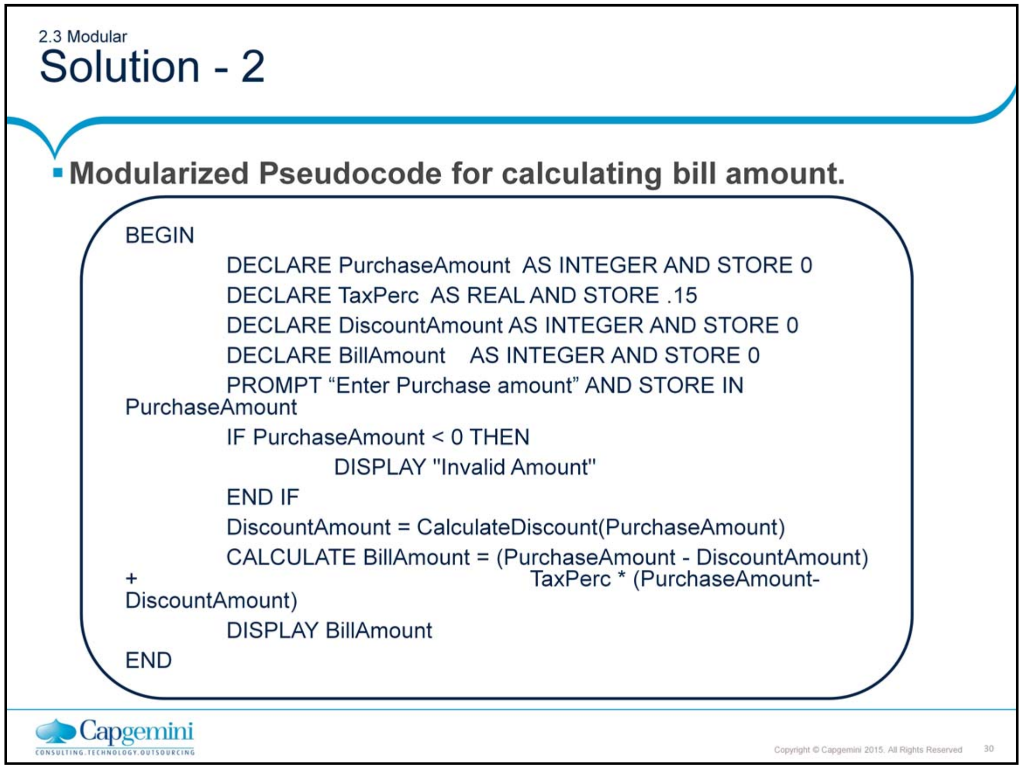
2.3 Modular

Example - 1

- Pseudocode to calculate the net billing amount to be paid by the customer. The discount is calculated on Purchase amount as given below
  - 30 % above 5000
  - 20 % for 3001 – 5000
  - 10 % for 1001 – 3000



The above pseudocode for calculating bill amount is not modularized



The above pseudocode for calculating bill amount is modularized

2.3 Modular

Solution - 2

- **Modularized Pseudocode for calculating bill amount.(Contd..)**

```
SUB  CalculateDiscount(PurchaseAmount)
        DECLARE DiscountAmt AS INTEGER AND STORE 0
        IF PurchaseAmount > 5000 THEN
                DiscountAmt = .30 * PurchaseAmount
        ELSE IF PurchaseAmount > 3000 THEN
                DiscountAmount = .20 * PurchaseAmount
        ELSE IF PurchaseAmount > 1000 THEN
                DiscountAmt = .10 * PurchaseAmount
        END IF
        RETURN DiscountAmt
END SUB
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31

The above pseudocode for calculating bill amount is not modularized (Contd..)

2.3 Modular

Code Considerations During Modularization

- During modularization of the code we need to decide:
  - Input Parameters:
    - For each lower level routine, what input parameters should be passed?
  - Output Parameters:
    - Should the output be in the form of a “parameter” or a “return value”?

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 32

2.3 Modular

Code Considerations

- Analyze parameters and return values for Accept_Employee_Details, Compute_Deductions , Compute_Gross_Pay

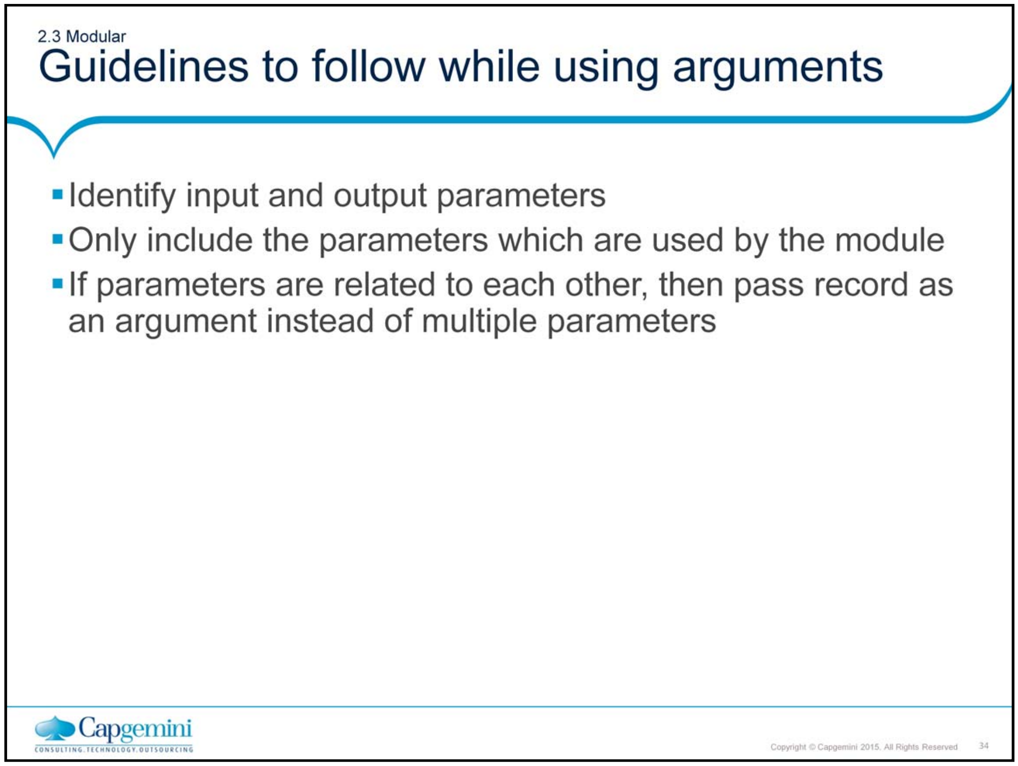
```
SUB Accept_Employee_Details()
      ACCEPT emp_code, Basic
END SUB
```

```
SUB Compute_Deductions(Basic)
  Provident_Fund = 0.12 * Basic
  Prof_Tax = 200
  Compute_Income_Tax (Basic)
END SUB
```

```
SUB Compute_Gross_Pay(Basic)
  HRA = 0.5 * Basic        /*** House Rent Allowance ***/
  OPA = 0.3 * Basic        /*** Offshore project allowance ***/
  Conveyance = 1700
  Gross = Basic + HRA + OPA + Conveyance
END SUB
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 33



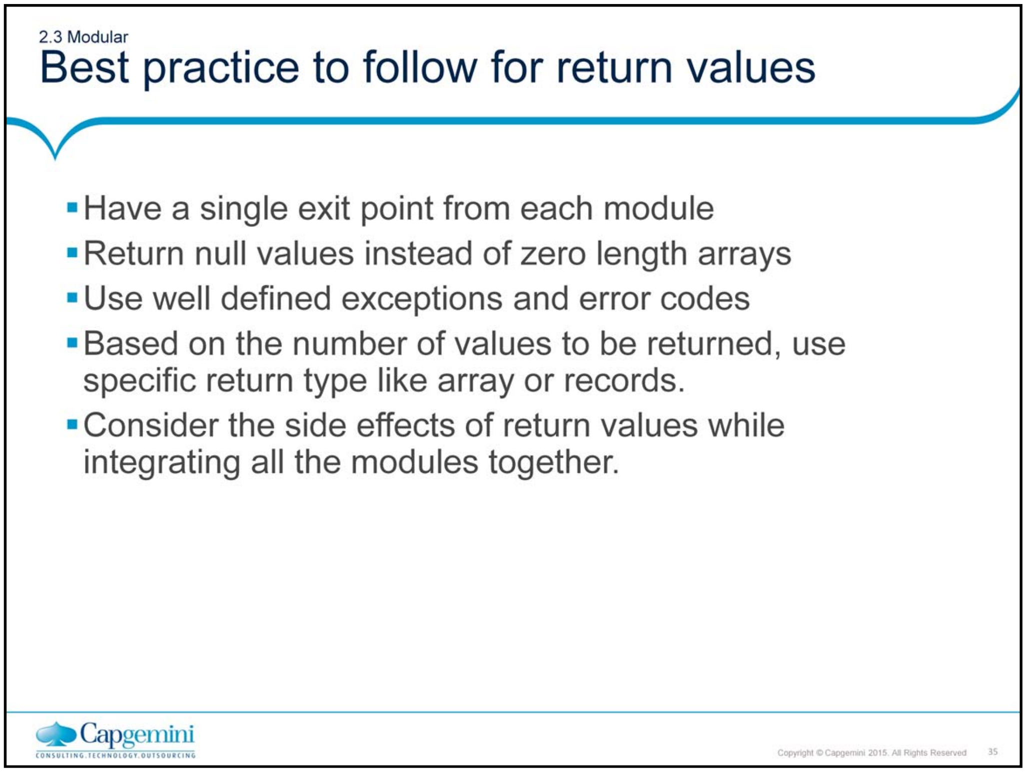
Guidelines to follow while using arguments

- Identify input and output parameters: After analyzing the requirement, identify input and output parameters before writing module code. For an Example, if you want to create a module for implementing a logic related to retrieving product details based on product id, then input parameter is productid and output parameter is variable of type record which contains product details.
- Only include the parameters which are used by the module, never pass unused parameters.
- If parameters are related to each other, then pass record as an argument instead of multiple parameters which strive for high cohesion and low coupling. For an example, refer the below module code snippet to add an employee details

```
SUB addEmployee(empId, name, salary)
END SUB
```

Instead use the below module signature

```
SUB addEmployee(Employee emp) //Consider Employee is a record
END SUB
```



Best Practice to follow for return values

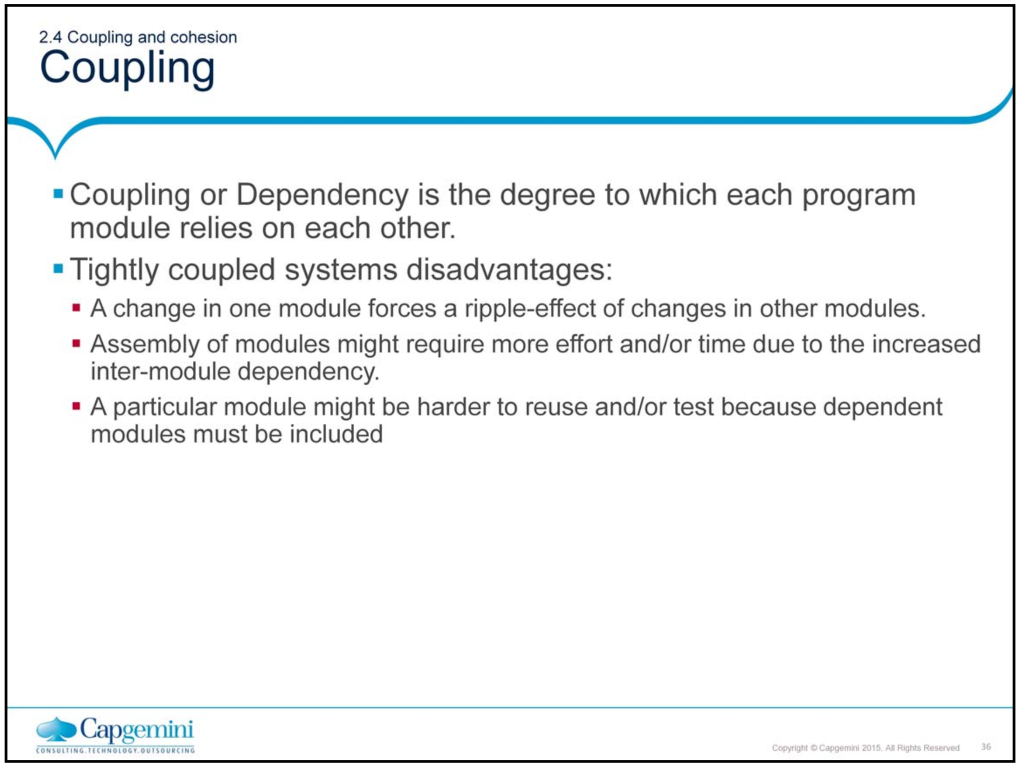
- Use only one return statement in each module as shown below:

```
 SUB checkDigit(number)
        DELCARE result AS BOOLEAN
        IF(num>0) THEN
                result=true
        ELSE
                result=false
        END IF
        RETURN result
END SUB
```

- Return null values instead of zero length arrays.
- Return exceptions/error code if an invalid input is accepted.

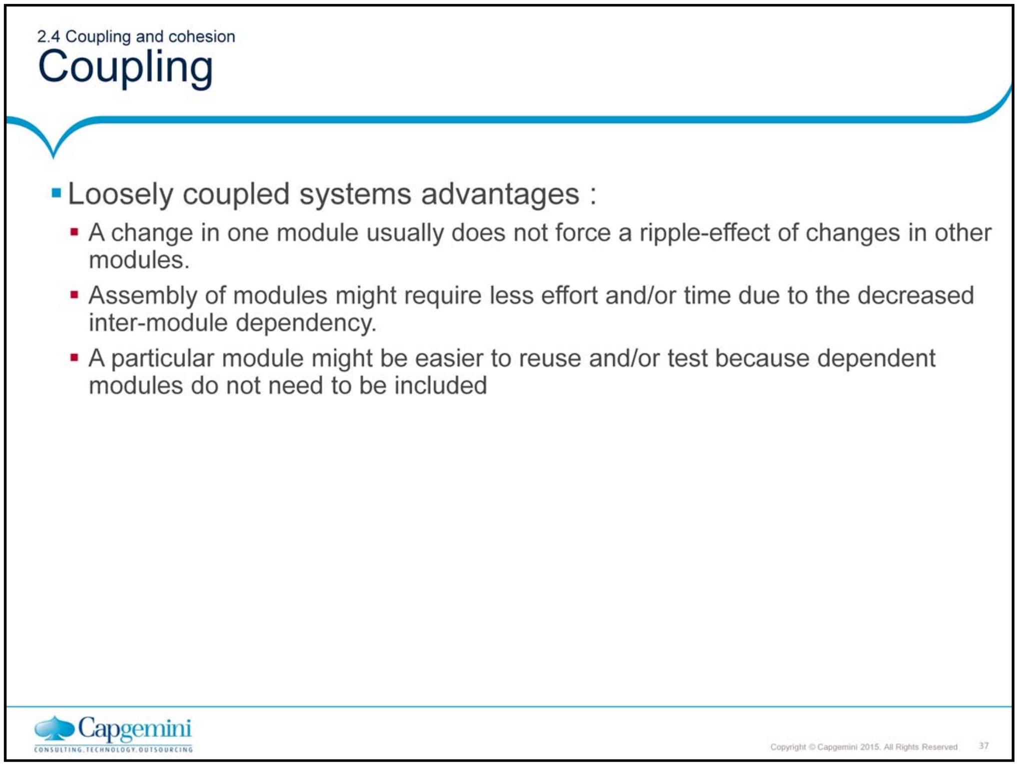
```
SUB getProductPrice(productId)
                DECLARE errorcode AS INTEGER AND STORE 0
                IF(elementfound(productId)) THEN
                        RETURN productPrice
                ELSE
                                errorcode = -1
                                RETURN errorcode;
                END IF
END SUB
```

- Use array as return type if more than one value has to be returned of same type or use record if more than one value has to be returned of different type.



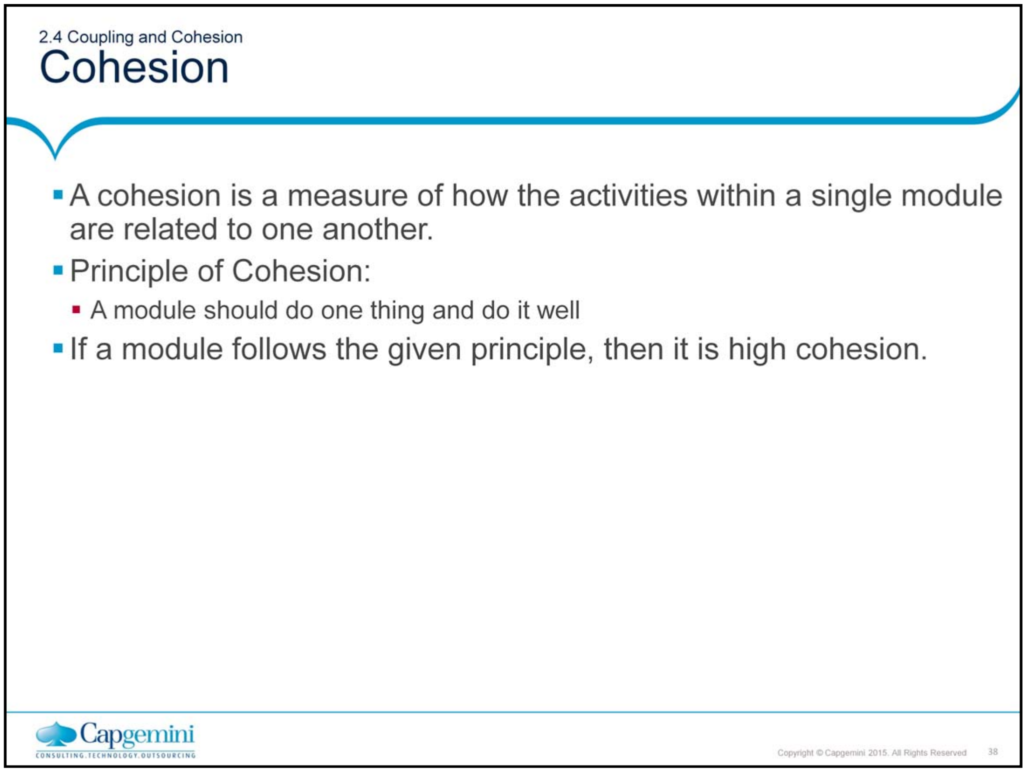
If a module does only one thing, we need to pass less parameters.
If a module does too many things, we need to pass more parameters.
Passing more parameters means high coupling.

We should strive for low coupling.



Low Coupling – reduce dependencies between modules

- Note that changes in one routine should not normally impact other routines, as long as the interface is the same.
- Remember that, in case too many things are done in one routine, a lot of data needs to be shared. This increases the dependencies, and also the chances of defects
- Consider “Smaller interface” (low coupling) versus “long list of parameters” (high)
- Consider data sharing through “parameters” (low) versus “global data” or “global files” (high)
- Note that passing “flags” that control the processing implies High coupling.



Writing function with high cohesion is always good practice.
Ideally, we should strive for **High Cohesion** and **Low Coupling**.

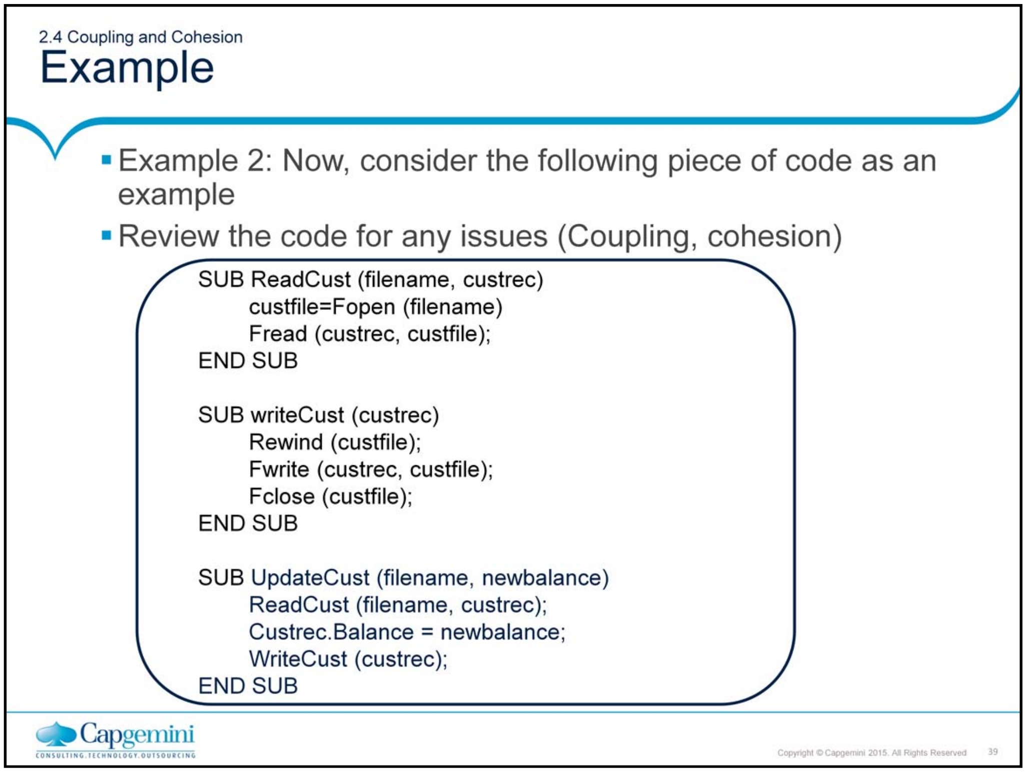
<u>For example</u>:
1. Sin (x); Cos (x); Tan (x); (High Cohesion, Low Coupling)

Note:
A good program requires high cohesion and low coupling.

2.Trig (Type, x)
where type is Sin, Cos, or Tan; (Less Cohesion, High Coupling)

Note:
Since in function trig we are trying to add all three functions together, we require to pass one extra parameter i.e. Type. This parameter indicates whether to calculate Sin, Cos, or Tan.

a. More number of parameters are required to pass, so it is High Coupling.
b. b. The function is not performing just a single task, so it is Low Cohesion.



Consider Fopen, Fwrite, Rewind, Fclose are predefined functions.

Fopen : To open a file
Fwrite : To write data to a file
Fclose : To close the opened file
Rewind :Moving cursor back to the first position of a file.

2.4 Coupling and Cohesion

Change Request - Example

- Suppose there is a Change Request to code in the above example:
  - Do not update balance, in case the new balance is less than 0. How will you implement this change?
  - Are any problems created due to the change?

 40

2.4 Coupling and Cohesion

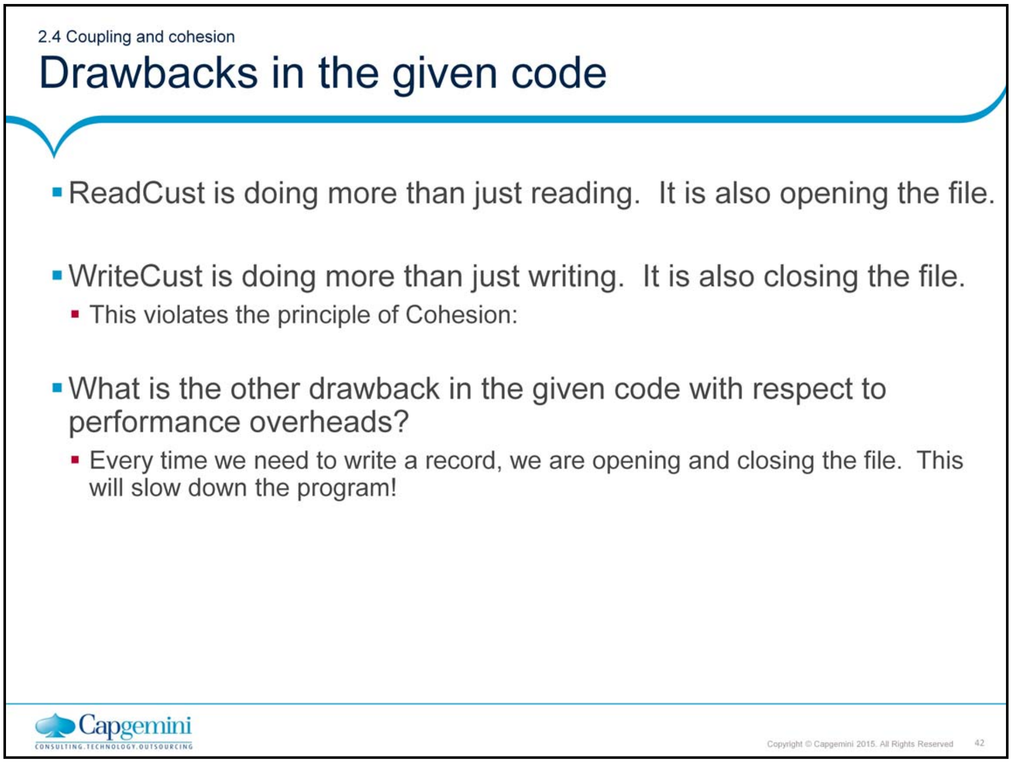
Change Request - Example

- **Sometimes the change is made as follows**

```
SUB UpdateCust (filename, newbalance)
    ReadCust (filename, custrec);
    IF(newbalance >= 0) THEN
            Custrec.Balance = newbalance;
            WriteCust (custrec);
    END IF
END SUB
```

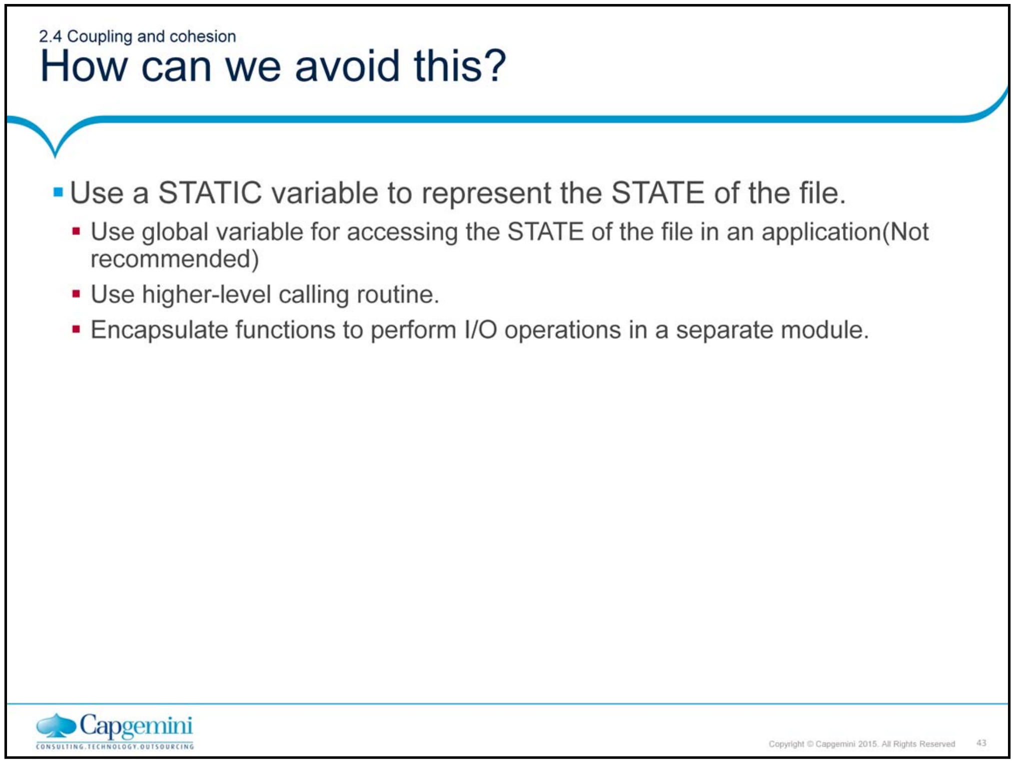
- **This means file will not be closed whenever “newbalance < 0”. This is because file gets closed in writecust, and writecust will not be called.**

- **After a few hours the program will crash saying “too many open files”.**



<u>Cohesion</u>:

- It means, the function should do one thing only, and the code should not be mixed up.
- As a result, the code becomes more readable and more maintainable.



Steps to be taken care for avoiding drawbacks on the implementation of cohesion and coupling:

- Use a STATIC variable(cust-file-already-open) to represent the STATE of the file as OPEN or not
- We cannot access the STATIC variable cust-file-already-open outside this routine
- We will be able to access it outside the routine if we declare it as GLOBAL. However, that is not a good practice.
- It is better to open and close the cust-file in the higher-level calling routine. The routine should call UpdateCust only to write the record.
- The calls to fread / fwrite / fopen / fclose are encapsulated or wrapped in the modules ReadCust / WriteCust / OpenCust / CloseCust respectively.

2.4 Coupling and cohesion

Revised Code

```
SUB ReadCust (filename, custrec)
      Fread (custrec, sizeof (custrec), 1, custfile);
END SUB
SUB WriteCust (custrec)
      Fwrite (custrec, sizeof (custrec), 1, custfile);
END SUB
SUB OpenCust (filename, mode, Cfile)
      Cfile = Fopen (filename, mode);
END SUB
SUB CloseCust (Cfile)
      Fclose (Cfile);
END SUB
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 44

The calls to fread / fwrite / fopen / fclose are encapsulated or wrapped in the modules ReadCust / WriteCust / OpenCust / CloseCust respectively.

2.4 Coupling and Cohesion

Revised Code (Contd..)

```
SUB UpdateCust (filename, newbalance)
      STATIC BOOLEAN cust-file-already-open = FALSE;
      IF (newbalance < 0) THEN
                {return;}
      ELSE IF (NOT cust-file-already-open) THEN
                OpenCust (filename, “r+”, Cfile);
      END IF
      ReadCust (Cfile, custrec);
      Custrec.Balance = newbalance;
      WriteCust (custrec);
END SUB
/**** CloseCust (Cfile);
Now we need to close the file only once at the end ***/
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Used STATIC variable(cust-file-already-open) to represent the STATE of the file as OPEN or not

2.4 Coupling and Cohesion

Advantages

- Wrapper modules help isolate system specific code in one place rather than all over the application.
  - They are easier to change during migration to different versions of Compiler or OS.
  - They are extremely useful when porting code:
    - To different platforms, or
    - To different database management systems
      - For example: In the code given in Example 2, “Read_cust” is acting as “wrapper module”. It is hiding the details about how to read the file.

2.5 Robust Program

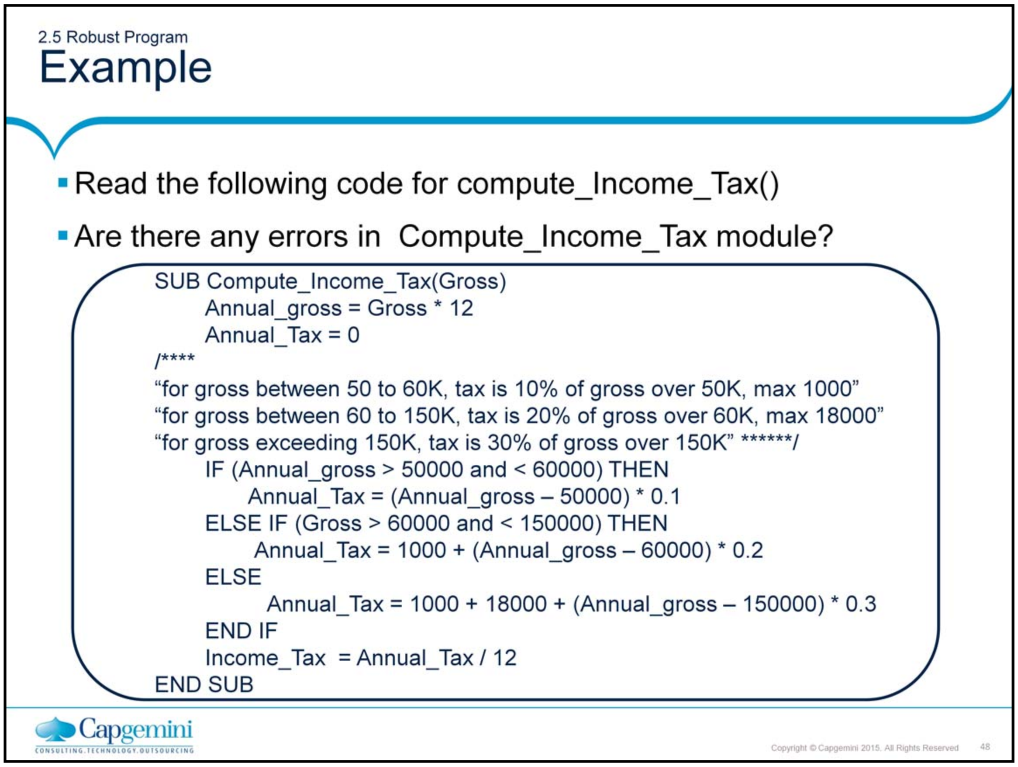
What is Robust program ?

- Robust program anticipates common and uncommon problems
- To ensure software is well defended, one should write robust program
- Robust program ensures that software handles invalid inputs reasonably preventing abnormal termination
- The program should terminate gracefully and provide appropriate debugging information for the programmer

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 47

Robust program ensures that software handles invalid inputs reasonably preventing abnormal termination. Robust program anticipates common and uncommon problems. To ensure software is well defended, one should write robust program. The program should terminate gracefully and provide appropriate debugging information for the programmer



In the above program we are not considering a case where annual_gross is less than 40000. As a result, in case the statements at the bottom of the program are executed, we will get wrong result. Note: When we use a nested IF, be careful while writing the last ELSE. This is because, the program will be executed for all unhandled conditions, as well.

The above program has one more mistake. We have used Gross instead of Annual_Gross. This error will not be detected by the compiler because we are using Gross as a variable form during calculation of monthly gross.
This will result in wrong output.

2.5 Robust Program

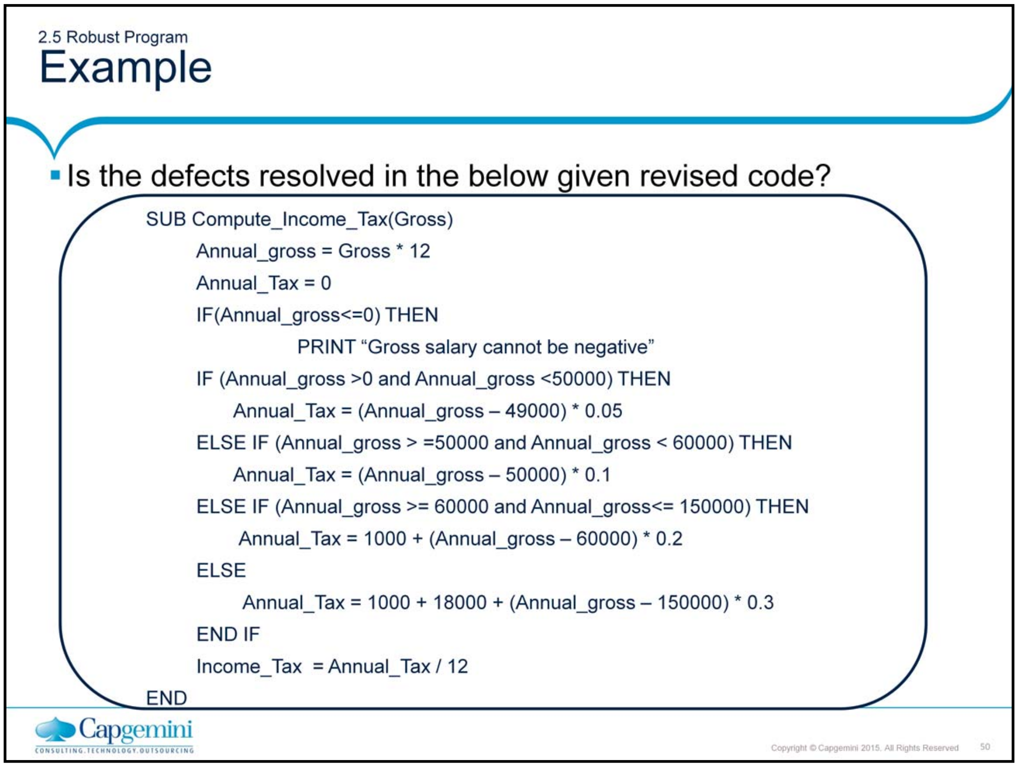
Defects Introduced in the Program

- Do you see any new defects introduced in the program?
  - What tax will be calculated in the Compute_Income_Tax module for an income of 40000?
  - Nested IF-THEN-ELSE should take care of all possible conditions. Is the nested clause doing so?
  - Be careful about the last ELSE – it may end up doing more than it should
  - Program has mistakes in variable name references:

    Example: Gross instead of Annual_Gross

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

For resolving all the defects mentioned in the slide, rework on the code.



In the above program we have considered a case where annual_gross is less than 40000. We have renamed, Gross as Annual_Gross. The statements in ELSE block will be executed only if Annual_Gross value is greater than 150000.

Annual Gross salary is calculation is taken care based on the below data

/****

“for gross less than 50K, tax is 5% of gross over 49K”

“for gross between 50 to 60K, tax is 10% of gross over 50K, max 1000”

“for gross between 60 to 150K, tax is 20% of gross over 60K, max 18000”

“for gross exceeding 150K, tax is 30% of gross over 150K” ******/

2.5 Robust Program

Make a Program Robust

- Will the program work (or fail gracefully) with unexpected input?
  - Check if it can display error message as “Input cannot be negative ” .
- Do not assume everything will be alright.
- Check for unexpected inputs or conditions.
- Remember GIGO – Garbage In, Garbage Out.
- Provide meaningful error messages

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Is the program robust?

- We test the program with different expected values, and it works fine as per our expectation.
- However, what if somebody provides an unexpected input?
  - will the program give proper error message and stop gracefully, or
  - will the program fail and give some garbage output
- Hence to make the program robust, add appropriate messages to handle unexpected input as mentioned in the slide.
- How can we make the program robust?

  Check if it can display error message as “Input cannot be negative”

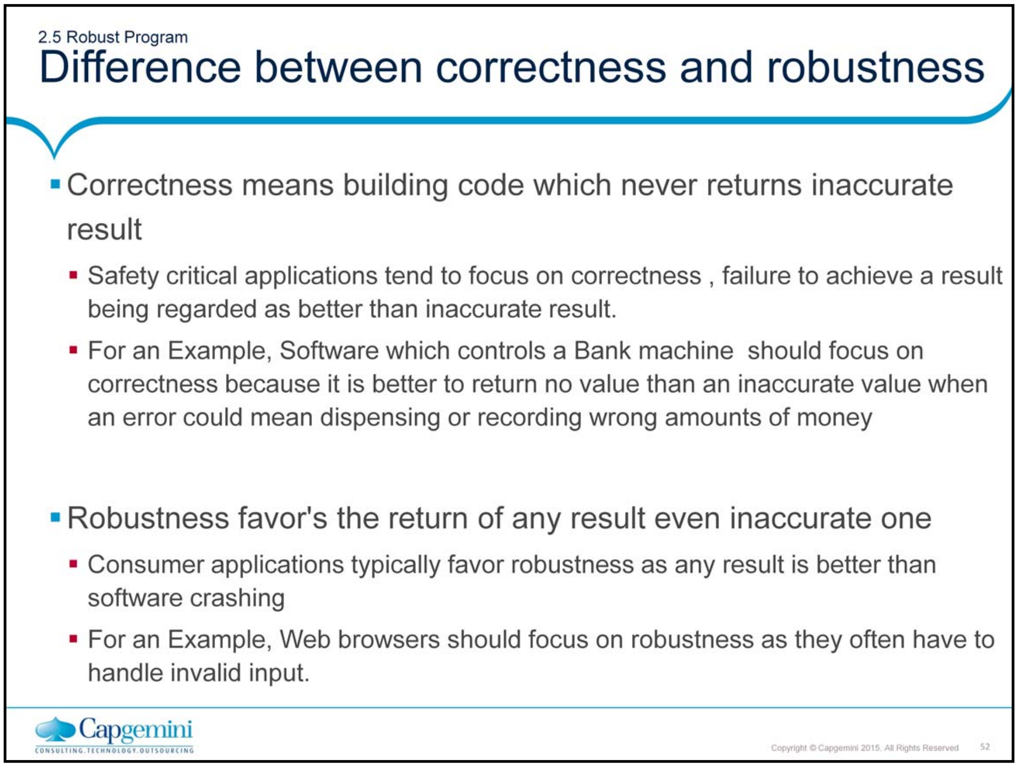
  Check if it can accept extreme values for inputs.

  For example: Basic = 1 crore

  Check if it can handle Output / Printer related problems.
- Check whether:

  “Can the Tax calculated be negative?” or

  “Can the Net Pay calculated be negative?”



First determine whether you want the program to primarily offer robustness or correctness

- Examples of Robustness
    - Consumer applications typically favor robustness as any result is better than software crashing
    - You may want a word processing program to sometimes display unwanted characters rather than to shut down when it detects them
    - Web browsers should focus on robustness as they often have to handle invalid input.
- Examples of Correctness
    - Safety critical applications tend to focus on correctness , failure to achieve a result being regarded as better than inaccurate result.
    - E,g software which controls radiation equipment for patients is best shut down if it receives bad input for a radiation dosage.
    - Software which controls a Bank machine should focus on correctness because it is better to return no value than an inaccurate value when an error could mean dispensing or recording wrong amounts of money

Lab

- Implementation of good programming practices - Lab 2

Write pseudocode for all assignments in lab 2 to apply the characteristics of good programming practices like modularity approach.

Summary

- In this lesson, you have learnt about:
  - Characteristics of a good program
    - Readable
    - Maintainable
    - Modular
  - Guidelines for writing good code
  - Coupling and Cohesion
  - Robust program
    - Difference between correctness and robustness

Review Question

Question 1: Why should you make code self documenting ?

A. To help the review process

B. To improve the quality of the code

C. To make it easier to debug

D. To meet international standards

Question 2: How can you minimize hard coding ?

A. By changing named constants in several places in your code

B. By using enumerated types instead of named constants

C. By using named constants to replace Booleans when you require more answers than true of false

Review Question

Question 3: What are the objectives of good layout ?

A. An accurate logical structure
B. To improve compatibility with compiler
C. To improve readability
D. To withstand modifications

Question 4 : What are the guidelines for good layout ?

A. Use braces as boundaries to demarcate block of code
B. Use indentation to show logical structure of code
C. Use line breaks for statements over 80 characters
D. Use white space sparingly

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 56

Review Question

Question 5 : You are writing a module to update the visitor counter on a web page ,what steps can you take to make this module as strong as possible ?

A. Call the module pageCountUpdate()

B. Call the module webPage6()

C. Test the module as soon as it is complete

D. Wait until all modules have been added to the program before testing it

Question 6 : Which of these applications should be robust ?

A. Bank machine software

B. Radiation machine software

C. Web browser software

D. Word processing software

Programming Foundation

Lesson 3: File Handling & Refactoring

Lesson Objectives

- To understand the following concepts:
  - Records
  - File Handling
  - Re-factoring
  - Avoidance of common programming mistakes

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

3.1 Introduction to Records

What is a record ?

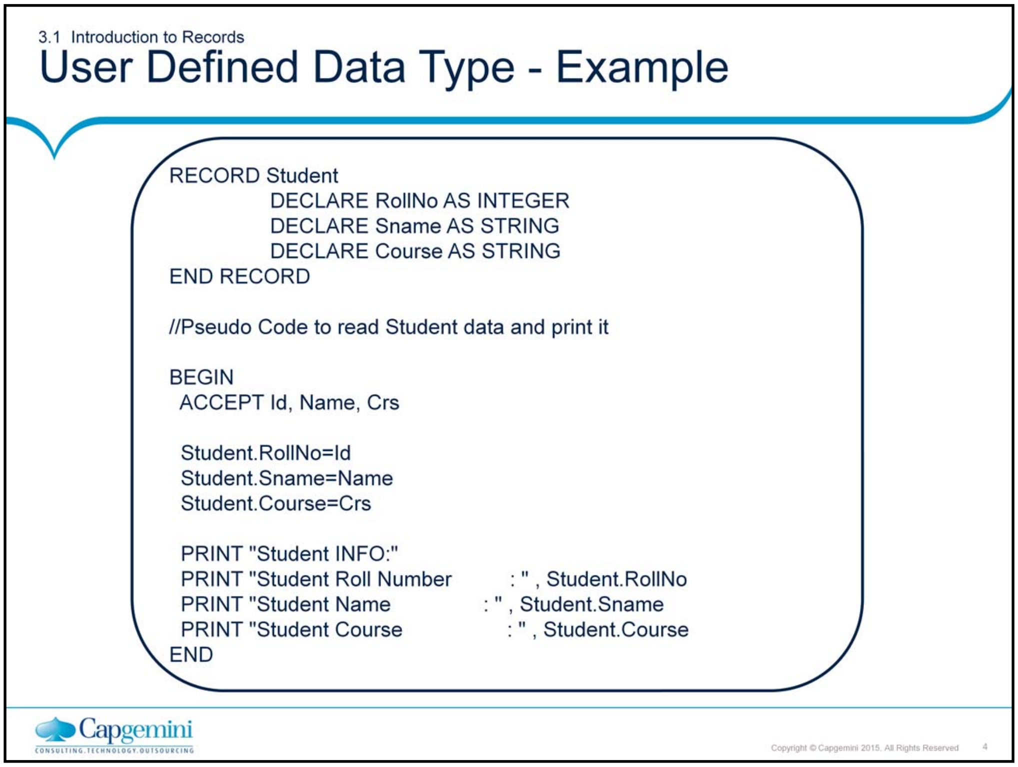
- Record is one of the composite data type, consisting of two or more values or variables stored in consecutive memory positions of different data types.
- Use them to simplify operations on blocks of data
- Use them to simplify module parameter lists
- Example:
  - Call Hardway (Name, Address, Phone, SSN, Sex, Salary)
  - Call Easyway (EmployeeRec)
- Use them to reduce maintenance of related data as changes to a record is easier to implement.
- Used to create user defined data types

Record: Record is one of the composite data type, consisting of two or more values or variables stored in consecutive memory positions .

Example for record:

```
RECORD Employee
          DECLARE ecode AS INTEGER
          DECLARE ename AS STRING
          DECLARE esal AS INTEGER
          DECLARE edept AS STRING
END RECORD
```

The above record is used to hold details about an employee such as employee code, employee name, employee salary and department in which employee is working.

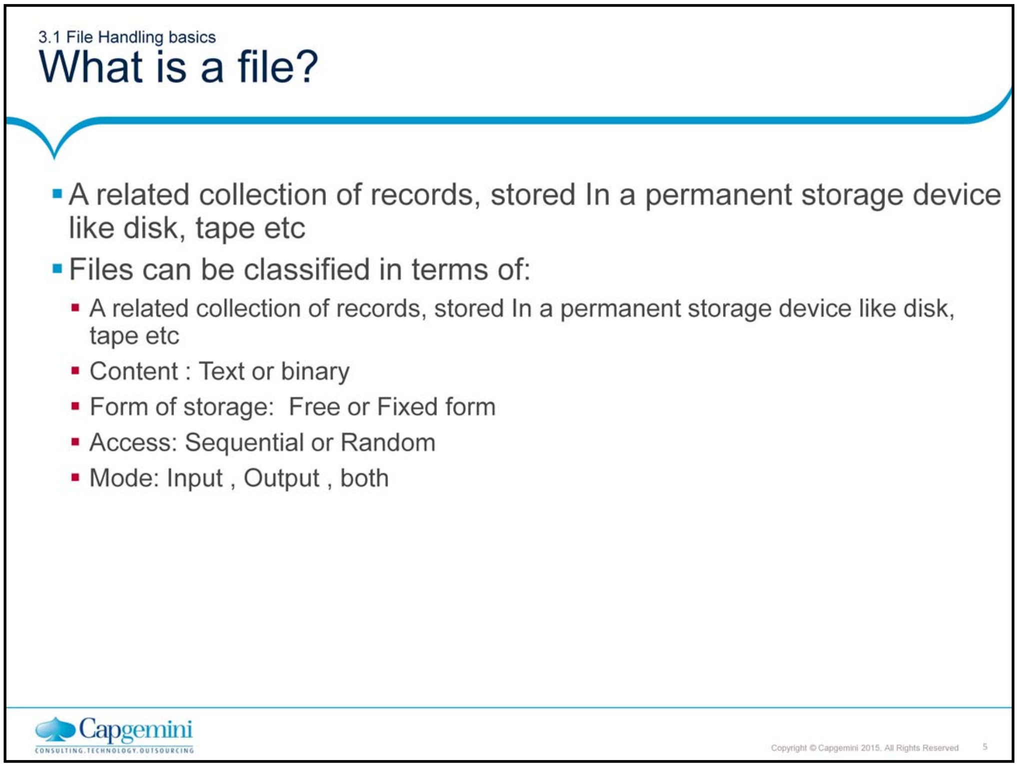


The code given in slide is used to maintain information about a student like rollno, sname and course details.

Functionalities implemented in the above code are

Accepting student details like id, name and course

Printing the same.



Files:

So far the input data was read from standard input and the output was displayed on the standard output. These programs are adequate if the volume of data involved is not large. However many business-related applications require that a large amount of data be read, processed, and saved for later use. In such a case, the data is stored on storage device, usually a disk in the form of file.

There are two types of files:

Binary file:

It is efficient for large amount of numeric data.

Text file:

It stores text and numbers as one character per byte.

It consumes large amount of storage for numbers.

It is useful for printing reports and text documents.

Access: Data from the file is accessible either sequentially or randomly.

Mode: Data from the file can be readable(Output) or data can be writable to the file(Input).

3.1 File Handling basics

Basic File Handling Logic

- Reading from a File
  - Open the file in read mode
  - Read record into memory variable
  - Do the processing
  - Close the file
- Writing to a File
  - Open the file
  - Write the record from the memory variable written
  - Close the file

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

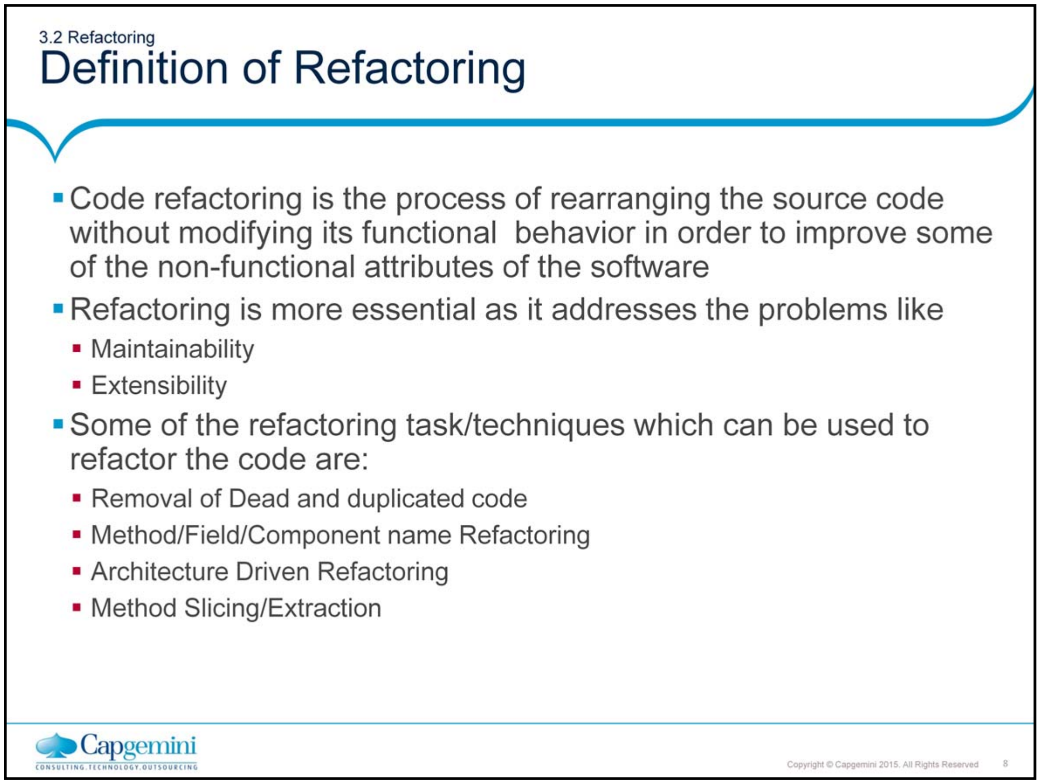
File I/O requires four functions:

- **open:** It allows access to file.
- **close:** It ends access to a file.
- **read:** It gets data from file.
- **write:** It adds data into file.

Demo : Reading data from file

- A product file contains the following fields
  - Product Id
  - Name
  - Price
- Refer the pseudo code in the notes page which will list all the products available in the file whose price is above 5,000


```
RECORD ProductRec
        DECLARE productId AS INTEGER
        DECLARE name AS STRING
        DECLARE price AS INTEGER
END RECORD
BEGIN
        DECLARE file AS FILE
        DECLARE product AS ProductRec
        PROMPT "Enter the filename" AND STORE IN file
        IF(fileExists(file)) THEN
                OPEN file
                READ data from the file AND STORE IN product
                IF(product.price>5000) THEN
                                DISPLAY "Product Id" + product.productId
                                DISPLAY "Product Name" + product.name
                                DISPLAY "Product Price" + product.price
                END IF
        END IF
END
```



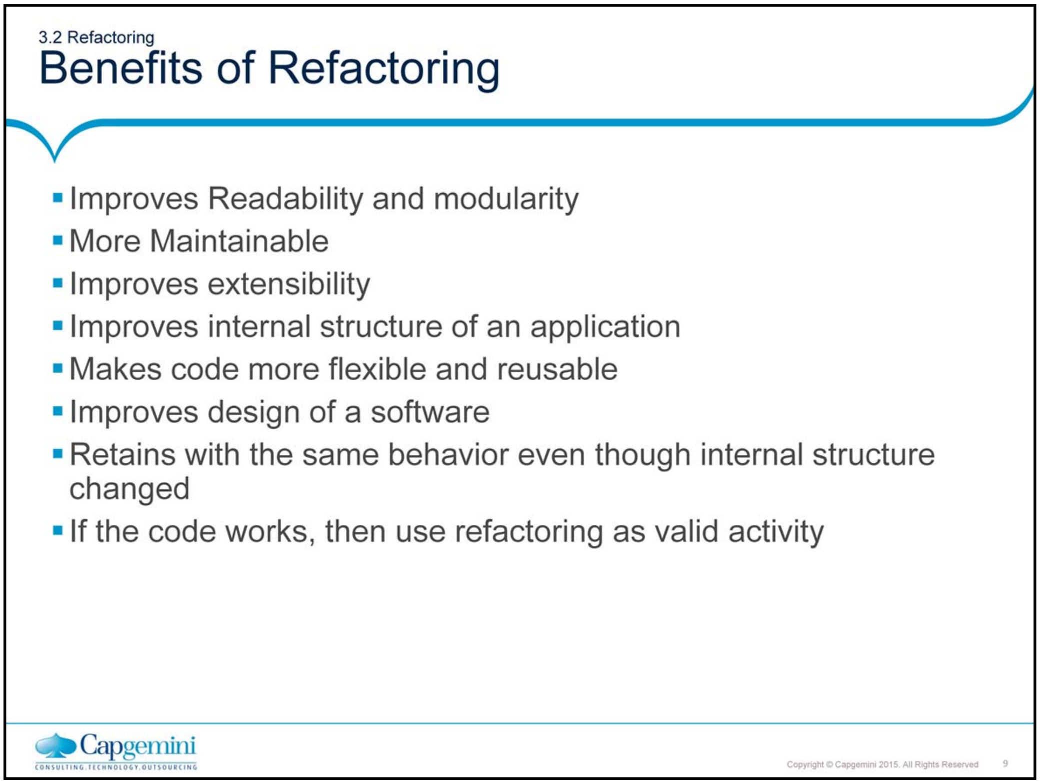
Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behavior. Refactoring requires care because it can introduce bugs to the code

Refactoring is more essential as it addresses the problems like

- Maintainability – Code is not maintainable
- Extensibility – Very difficult to add new feature or upgrade an existing feature

Refactoring task/techniques are:

- Removal of Dead and duplicated code - Remove the unused variables/code, unreachable statements and duplicated code.
- Method/Field/Component name Refactoring – Name of a method/field/component might be confusing, provide a meaningful name for the same and ensure the changes are reflected in other references wherever the variable/field/component is used.
- Architecture Driven Refactoring - Few functionalities in an application can be potentially reused in other applications. In order to reuse such a functionality, separate the code in a separate component by adhering to the architecture design.
- Method Slicing/Extraction - Longer method needs to be broken up into smaller ones to enhance readability and maintainability. Also need to achieve cohesion by implementing only one logic in a method with more suitable name.



Benefits of Refactoring:

- Improves code readability and modularity by improving code structure and design
- Reduced complexity to improve the maintainability of the source code
- Improves extensibility: Easier to add new features
- Refactoring involves improving internal structure of a software without altering its external behavior
- Goal of refactoring is to make the software more flexible and reusable
- It involves changing the design of the software after it has been coded to improve the design of a software. Refactoring is mainly used to improve badly designed software.
- Ensure software's external behavior remains the same after refactoring
- Refactoring is a valid activity only when the code is already in working state

3.2 Refactoring

Match each refactoring task to the code issue it addresses

Refactoring Task	Code Issues
Removal of Dead and duplicated code	Lack of Reusable components
Method/Field/Component name Refactoring	Poor Coding Style
Architecture Driven Refactoring	Lack of Modularity
Method Slicing/Extraction	Code Redundancy
	Lack of Layered Architecture
	Readability
	Lack of Pluggable Components
	Maintainability

Some of the Code Issues are:

•Lack of Reusable components : Potentially component may not be reused.

•Poor Coding Style : Coding standards/naming conventions has not been followed

•Lack of Modularity : Application components can't be reused as it is tightly coupled.

•Code Redundancy : Duplicate code and dead code exists.

•Lack of Layered Architecture : Any change in one layer causing changes in all other layers

•Readability : Code is not readable due to lack of meaningful name and comments.

•Lack of Pluggable Components : Existing components are not easily replaceable due to the tight coupling code.

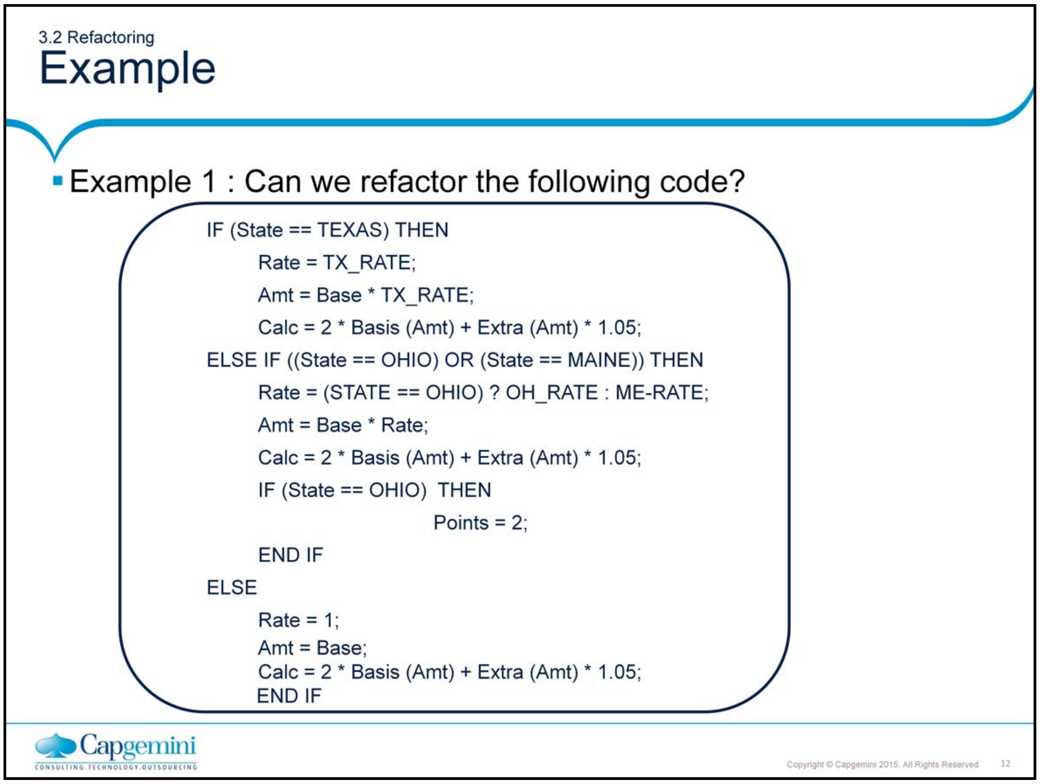
•Maintainability: Hard to maintain the code.

3.2 Refactoring

Match each refactoring task to the code issue it addresses - Solution

Refactoring Task	Issue Addressed
Removal of Dead and duplicated code	• Lack of Reusable components • Code Redundancy
Method/Field/Component name Refactoring	• Poor Coding Style
Architecture Driven Refactoring	• Lack of Layered Architecture • Lack of Modularity • Lack of Pluggable Components
Method Slicing/Extraction	• Readability • Maintainability

Each refactoring task is addressed to the code issue it addresses.

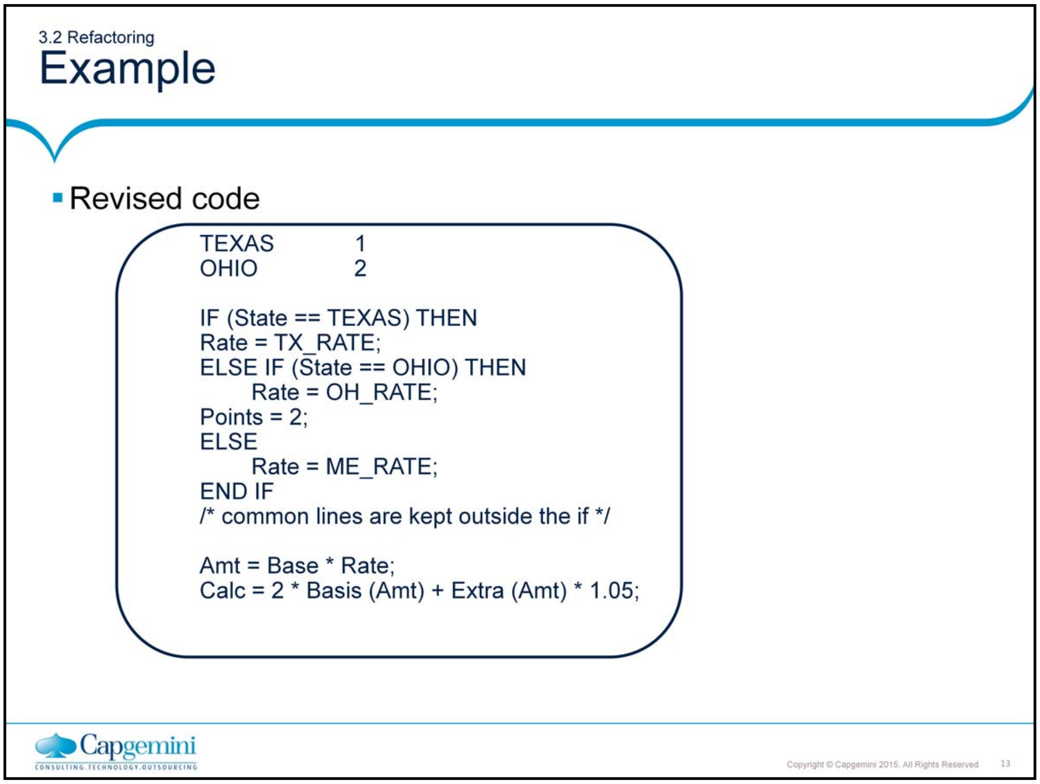


The above code is used to calculate tax based on the state. Tax rate varies for each state but similarity exists in the tax calculation.

Rate, Amt and Calc variables are assigned values in each if-else condition.

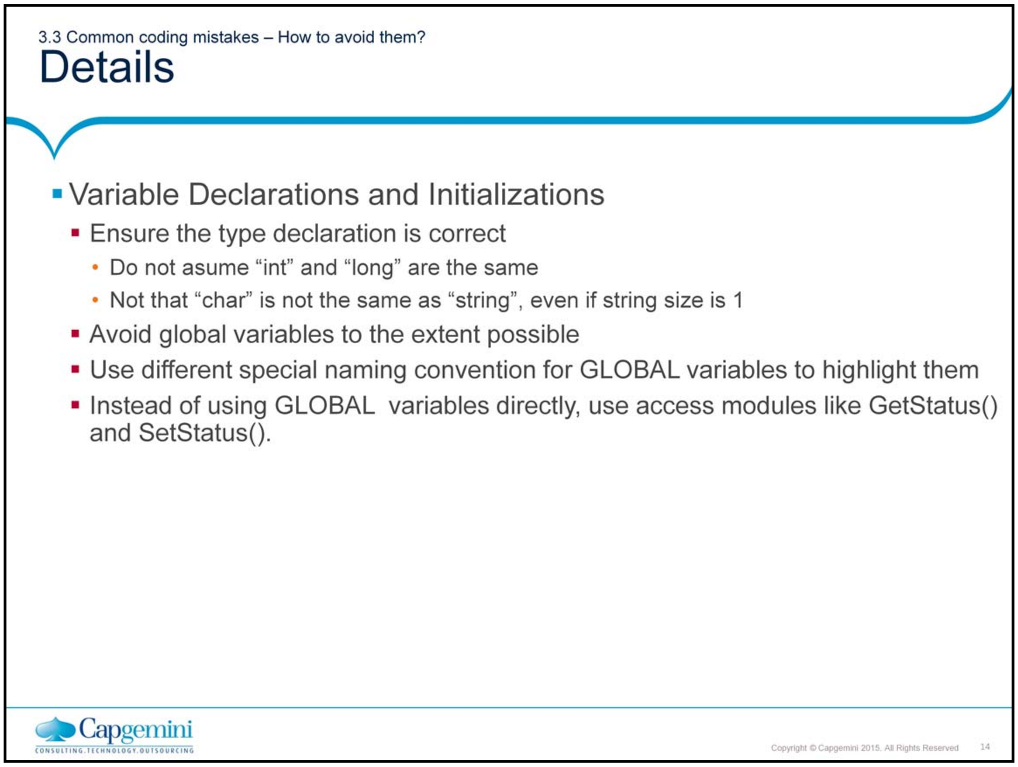
Use Refactoring for performing the below task:

- Avoid Duplicate code in each if else condition
- Common code should be kept outside if condition



Note: The code is more efficient and readable as compared to the previous code.

Are any further improvements possible?
If the program is used for three states, it is likely to be used for other states, as well, in the future.
Instead of using a nested IF ELSE statement, using a table or an array for tax rates of all states will make it easy to add more states later on.



Variable Declarations and Initializations:

Code Snippet - Example:

```
Boolean more-records = TRUE;
…
while ( more-records ) do …
```

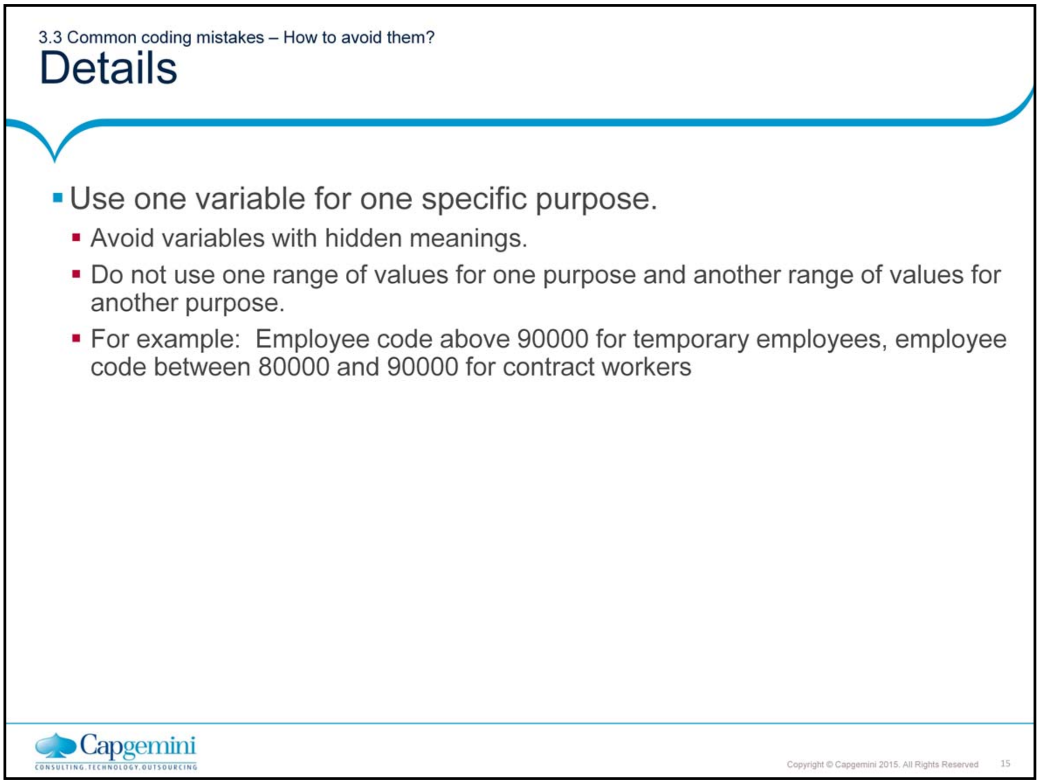
Better version of the above code snippet:

```
Boolean more-records;
…
more-records = TRUE;
while ( more-records ) do …
```

Avoid GLOBAL variables to the extent possible. : Note that it is difficult to debug, because a GLOBAL variable can be changed from any module. It may create problems with a recursive module.

Use different special naming convention for GLOBAL variables to highlight them. For example: MAX_USERS_G for global

Instead of using GLOBAL variables directly, use access modules like GetStatus() and SetStatus(). Note that this ensures that the variable is used only at once place inside those modules. This becomes easier to change or debug.

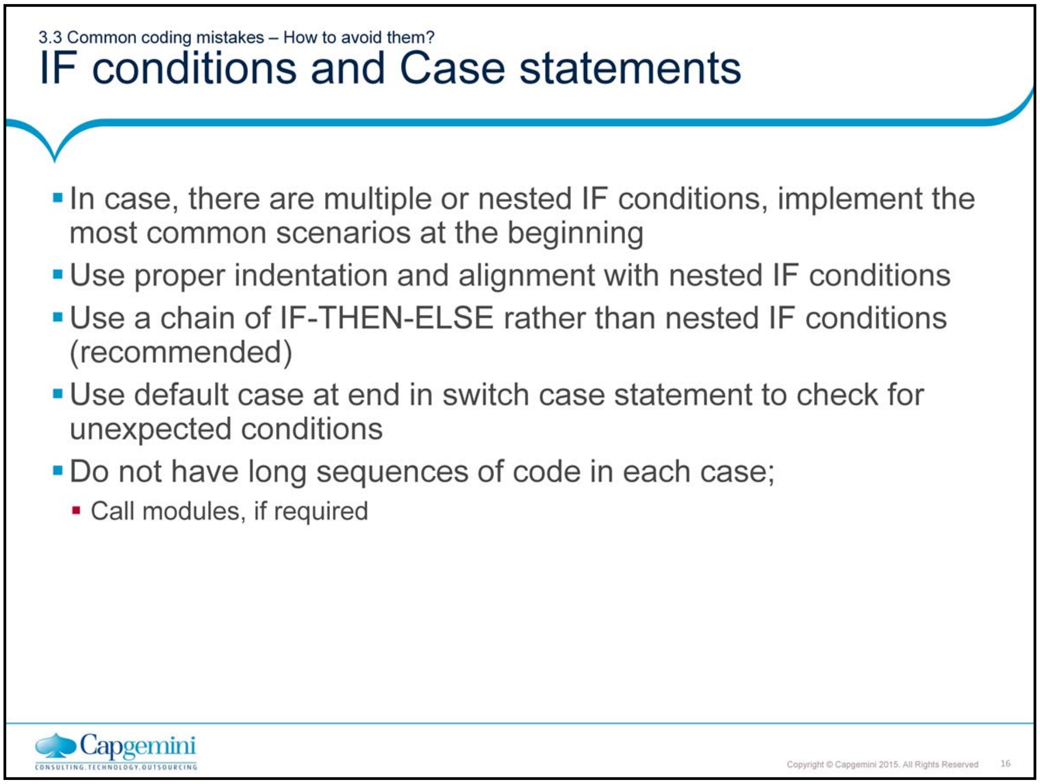


Use one variable for only one purpose.

For an example, if a variable is used for multiple different what can we provide? Consider a variable that is used to count the number of students, and count their grades. How would you name this variable: count, studentCount/gradeCount?

So create one variable for one purpose, otherwise data management related to the requirement will be complex.

- **Don’t use variable with hidden meanings** because it might not be understandable by others.

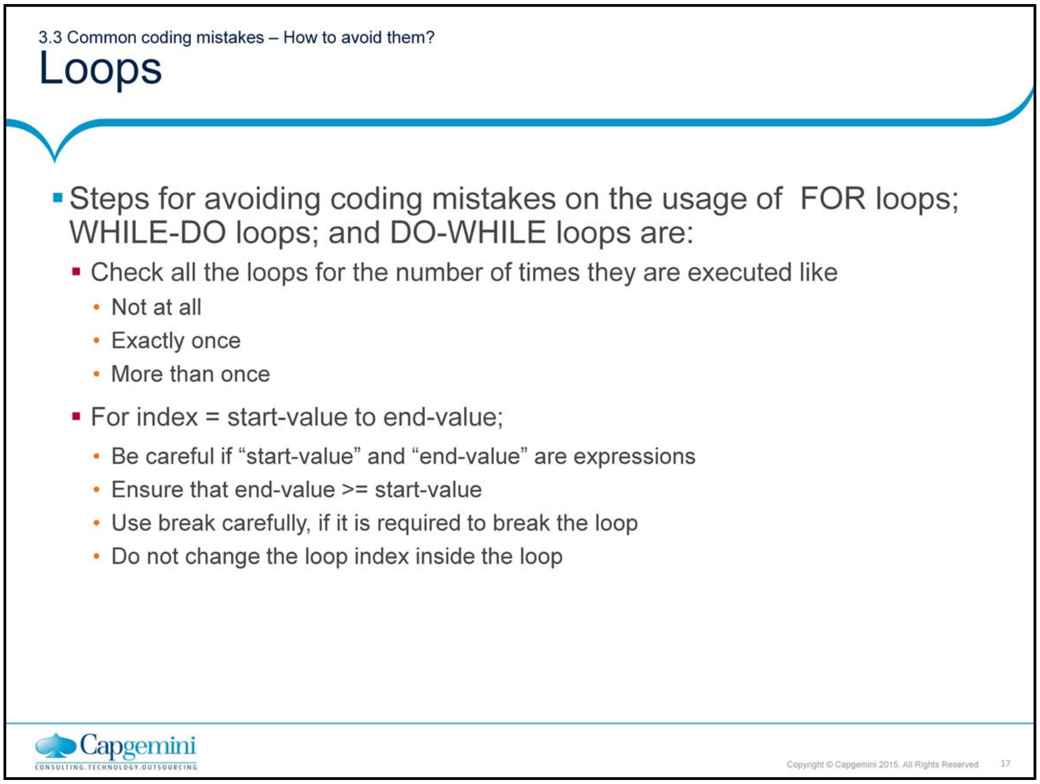


How to avoid Common Coding Mistakes while using IF and case statements:

- If there are multiple nested IF conditions, then find out the condition which has high priority and specify the same in first if condition.
- Adopt a standard indentation style for your code, and stick with it throughout the program so that program will be easily readable by other programmers.
- **For an Example:**

```
BEGIN
    DECLARE file AS FILE
    PROMPT "Enter the filename" AND STORE IN file
    IF (theFileExits) THEN
            determine the length of the file
             IF(FileLength>o) THEN
                 readFile(file); // Invoke readFile module
             ELSE
                 PRINT "File doesn't contain data"
              END IF
    ELSE
                 PRINT "File doesn't exist"
    END IF
END
```

In the above code, indentation is followed, if-else is used and fileexists condition is checked first before finding length of a file for avoiding common mistakes.



Steps for avoiding Common Coding Mistakes while using Loops are:

- Check for the number of times the loop executed: For an example, execute the below loop thrice, by considering num values as 5, 4 and 2.

```
WHILE (num<5)
DO
 ….
END WHILE
```

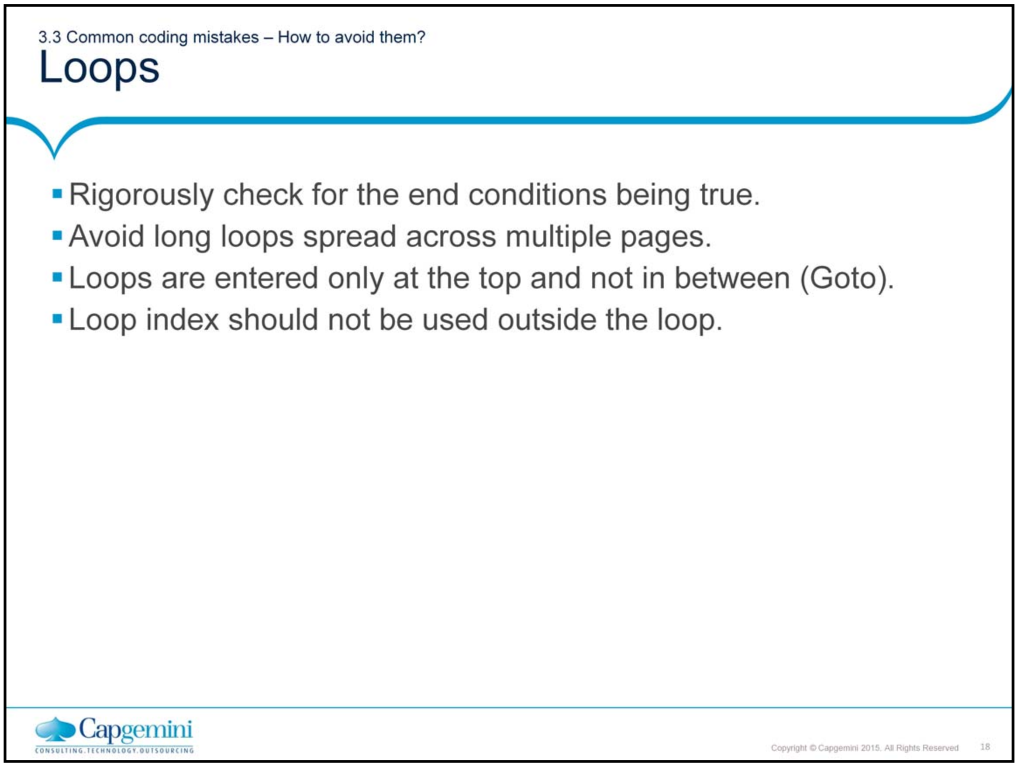
The above loop will be executed once for the value of 4, and the loop will not be executed at all for the num value of 5 and the same loop will be executed more than once if num value is 2.

- For loop:
  - Ensure that endvalue>=startvalue.

Better version	Not recommended to be used
FOR index= 0 to 10	FOR index=10 to 0
..	...
END LOOP	END LOOP

  - Do not change the loop index within the loop as shown in the below example. In the below example, loop execution will be stopped after the re-initialization of loop index inside the loop.

```
FOR index = 0 to 10
      index=13
END LOOP
```



Steps for avoiding Common Coding Mistakes while using Loops are:

Rigorously check for the end conditions being true : Rigorously avoid coding infinite loops.

Avoid long loops spread across multiple pages : The loop should be short enough to view all loop code at once.

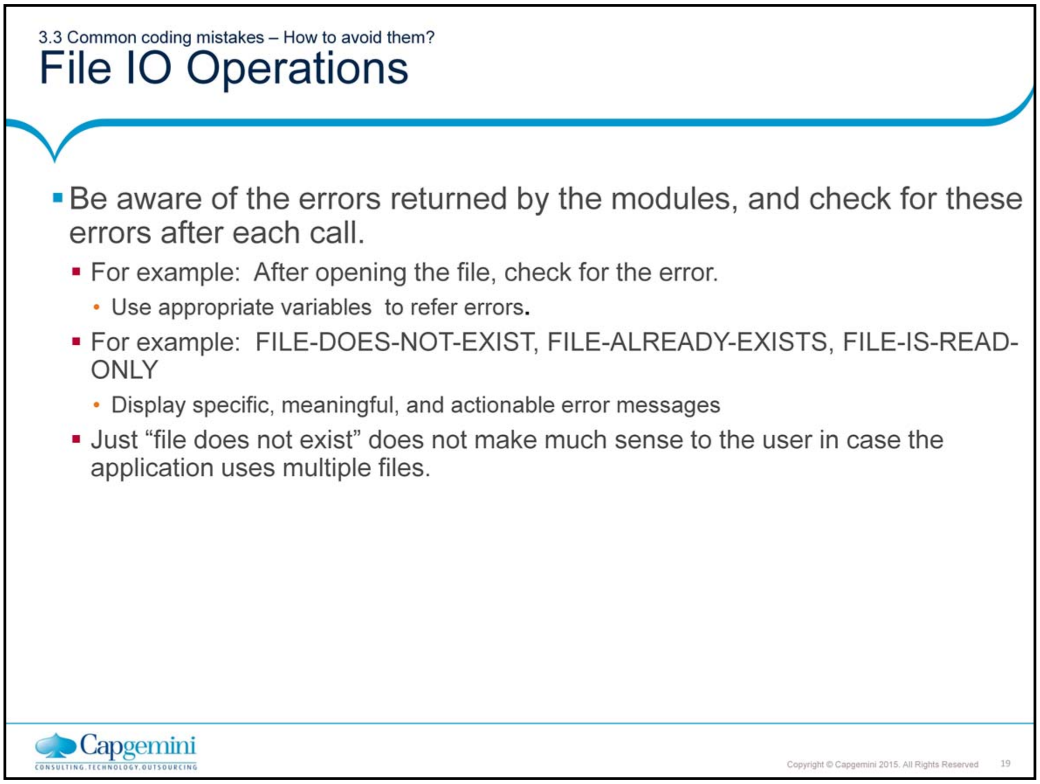
Loops are entered only at the top and not in between (Goto) : "goto" statement is always not recommended to be used inside loop as the loop execution always re start from the lower limit.

Loop index should not be used outside the loop. Consider the below code

```
FOR index=1 to 5
END LOOP
index=3;
```

Don't use index outside of the loop. Use another variable, if required as shown below:

```
FOR index=1 to 5
END LOOP
num=3;
```



Points to be considered for avoiding Common Coding Mistakes while performing file IO operations are:

- Be aware of the errors returned by the modules, and check for these errors after each call like
    - Check for the error, if file doesn't exists
    - After opening the file, check for the error.
    - Check for the error, if file size is zero.
    - Check for the error, if file is not readable/writable.
- Use appropriate variables to refer errors. For example: FILE-DOES-NOT-EXIST, FILE-ALREADY-EXISTS, FILE-IS-READ-ONLY
- Display specific, meaningful, and actionable error messages. For an Example, Just "file does not exist" does not make much sense to the user in case the application uses multiple files. Instead use the error message as "Employee.txt file does not exist", considering "Employee.txt" is a filename.

3.3 Common coding mistakes – How to avoid them?

Calls to modules

- Ensure that the number of parameters, and the sequence is correct for the module call.
- Ensure that there is no “Type mismatch” for any parameter.

Consider the below signature of a module to calculate total price.

calculateTotal(Integer price, Integer quantity)

Refer the valid and invalid statements to invoke a module

calculateTotal(3,5); //Valid

calculateTotal(4,3,4); //Invalid

calculateTotal(‘Test’,3); //Invalid

Summary

- In this lesson, you have learnt about:
  - Record is a composite data type used to store data of different datatypes
  - Record is a composite data Usage of files
  - Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behavior.
  - Use proper variable types, and initialize wherever necessary.
  - Avoid GLOBAL and STATIC variables to the best extent possible.
  - In case of multiple or nested IF conditions, implement the most common scenarios at the beginning
  - In case of loops, rigorously check whether the end conditions are true.
  - In case of files, be aware of the errors returned by the functions, and check for these errors after each call.

Summary

Review Question

Question 1: Mixing different types of calculation is a good practice.

A. True

B. False

Question 2: Which of these are characteristic benefits of refactoring

A. It enables you to add functionality to your code

B. Its main function is to optimize performance

C. It makes code easier to understand and modify

D. It makes code more flexible and reusable

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

Review Question

Question 3: Data item that make up structure can be of the different types.

A. True

B. False

Question 4: Which type of file format is used to store one character per byte?

A. Binary File

B. Text File

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23

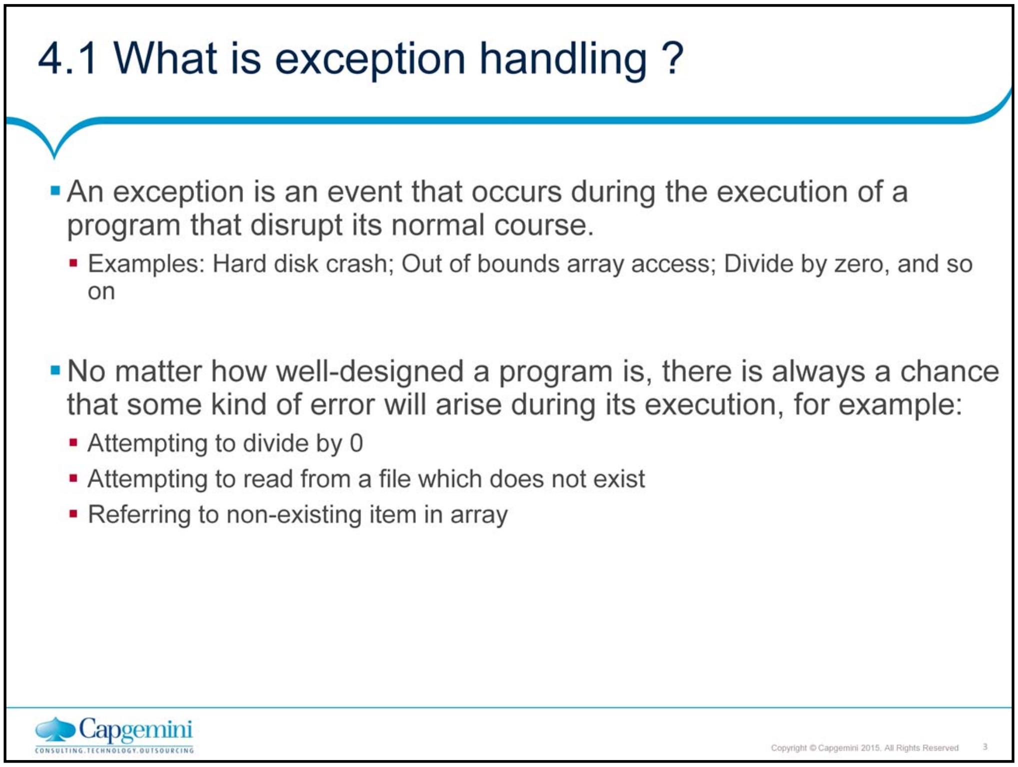
Programming Foundation

Lesson 4 : Exception Handling

Lesson Objectives

- After completing this lesson, you will be able to understand the following topics:
  - Importance of exception handling.
  - Guidelines for creating exceptions
  - Exceptional scenarios in ATM system – case study
  - Exception handling case study : Product management system
  - What is Defensive programming
  - Purpose of defensive programming
  - Techniques of defensive programming

 2



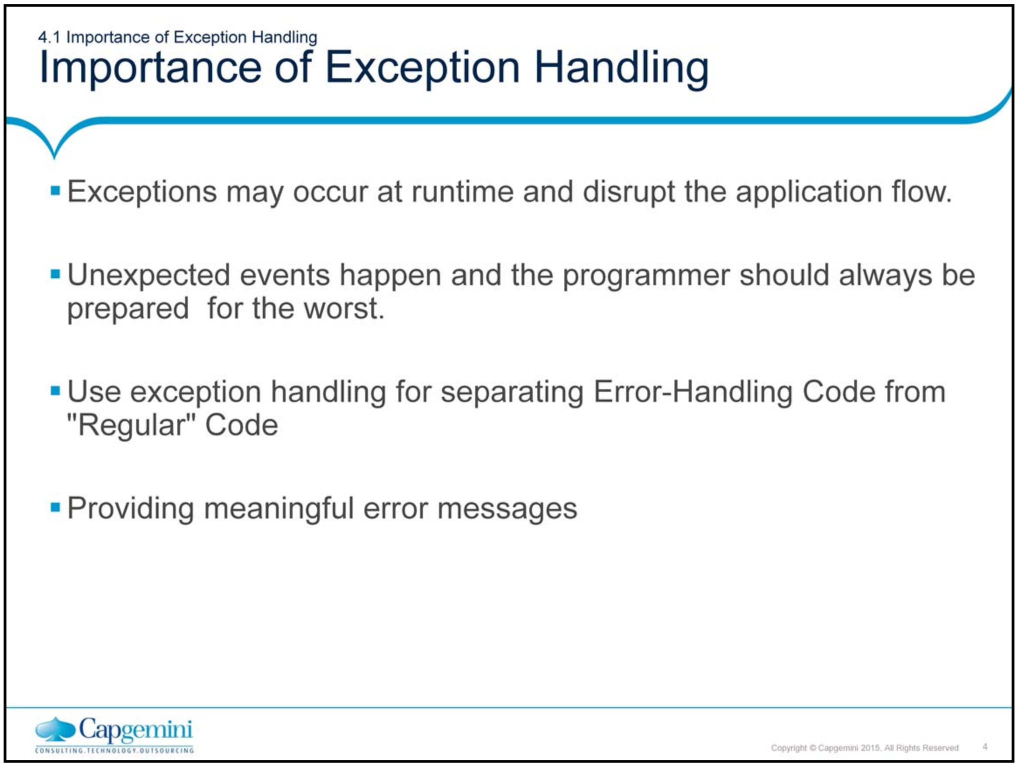
Exception Handling:

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. Exceptions are used as a way to report occurrence of some exceptional condition. Exception provides a means of communicating information about errors up through a chain of methods until one of them handles it.

> When an exception occurs, the executing method creates an Exception object and hands it to the runtime system —”throwing an exception”
>
> The runtime system searches the runtime call stack for a method with an appropriate handler, to catch the exception.

For an example, while booking a ticket in railway reservation system suddenly if the database is down and if an abrupt error page with numerous lines of exception messages displayed on the screen, then user(non-technical person) would be embarrassing and will get annoyed.

Instead of displaying an abrupt error page, if your code catches this database down scenario and displays a graceful message on screen saying "We are currently experiencing some difficulties in our system. Kindly try after some time. Thank you" , then it would be a better message for the end users.

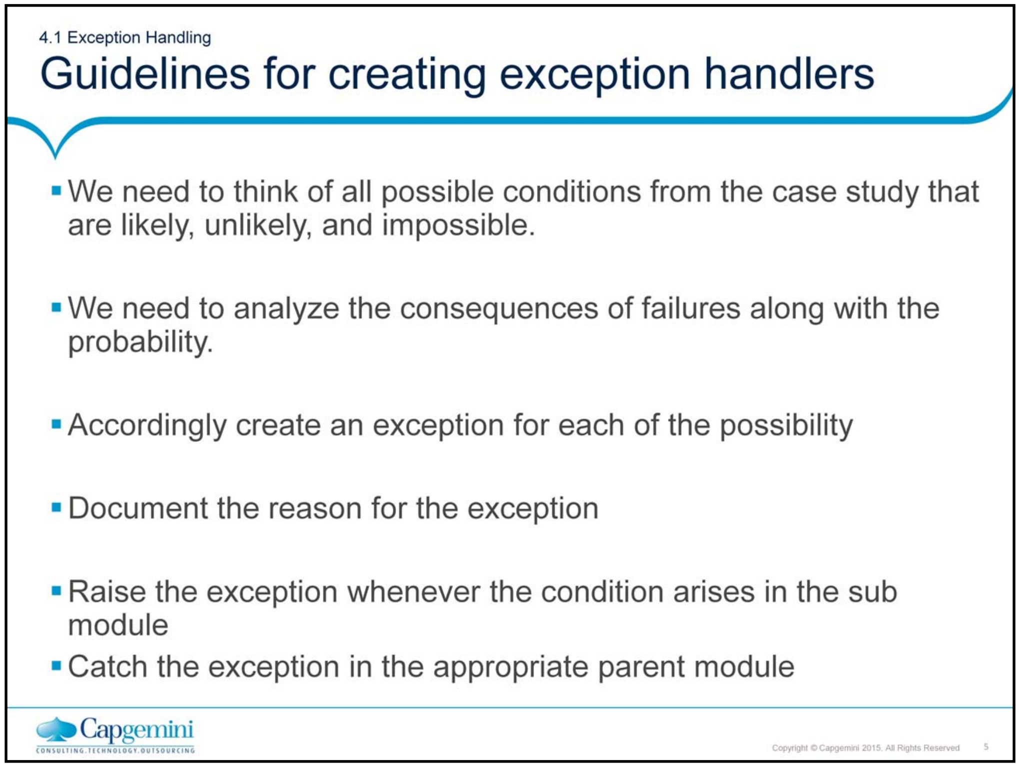


Importance of Exception Handling

- Once an exception occurs at runtime, the program execution will be stopped immediately and an exception message gets displayed which may not be understandable by all the end users.

- Whenever an unexpected input is passed to an application by the end users, program may get crashed. So in order to create an application which should be sustainable at any scenarios, exception handling is mandatory to be taken care.

- Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program. Enclose the error handling code separately in try block from regular code for making program execution to happen without any disruption even though an exception occurs.



- Pseudocode example without taking care of exception’s

SUB readFile

open the file;

determine its size;

read data from the file;

close the file;

END SUB

At first glance, this function seems simple enough, but it ignores all the following potential errors.

- What happens if the file can't be opened?
- What happens if the length of the file can't be determined?
- What happens if the read fails?
- What happens if the file can't be closed?

See the below Pseudocode with exception handling for taking care of the possible potential errors.

```
SUB readFile
        open the file;
        determine its size;
        read data from the file;
        close the file;
EXCEPTION
        WHEN fileOpenFailed THEN
                doSomething;
        WHEN sizeDeterminationFailed THEN
                doSomething
        WHEN readFailed THEN
                doSomething;
        WHEN fileCloseFailed THEN
                doSomething
END SUB
```

4.2 Case Study 1

- Case 1: Consider the transaction at a Bank ATM. An user needs to withdraw some money.
  - Analyze this transaction from the user’s point of view.
  - Think out all scenarios about what can go wrong.
  - Determine how the system should perform this transaction.
  - What irritations have you faced at an ATM?
- Scenarios:
  - User authentication: User thinks he has entered the correct password. However, the system says “invalid password”.
  - The ATM does not provide coins. However, it expects the user to enter the amount in the format 5000.00
  - User enters the amount as 15000, and the system gives 150 notes of hundred rupees
  - The system asks the user for a password, then asks the details about the transaction (like amount, requirement of a printed receipt, etc.), and then at the end it tells the user that the password is invalid.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

4.2 Exception Handling

Case Study 1

- Case 2: Now, consider the same transaction from the programmer's point of view.
  - Visualize what all can go wrong.
  - Determine how should the system respond.

- Scenarios:
  - The user may have multiple ATM cards. Although the correct password has been entered, it may be for a different card. Does the error message consider this possibility?
  - After one transaction is completed, the system says "Thank You", without checking whether another transaction is desired.
  - Does the system provide multiple alternatives or options to recover from errors, or to deal with various contingencies?
  - Notes of a specific denomination are not available. Can user enter "OK" to ask for another denomination?

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8

4.2 Exception Handling

Case Study 1

- Machine is out of cash. What options are available?
  - Allow for withdrawal of a smaller amount.
    - The system should suggest this amount, and not let the user guess by trial and error!
  - Display the location of another ATM nearby.
    - Can we double check whether that ATM has enough cash?
- What if the communication link is very slow (during “Authentication” and “Balance Check”)?
- What if we get a device error, file io error, or disk full error while writing the transaction to the database?

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 9

4.2 Exception Handling

Common Issues

- Some of the common issues in Exception Handling are:
  - On opening a file for read access, you get an exception “file does not exist”.
    - Check for this condition.
    - Display a specific error message, with the name of the file that was not found.

- Some of the common issues in Exception Handling are (contd.):
  - On opening a file for write access, you get an exception “file already exists”.
    - Check for this condition.
    - Display a specific error message, with the name of the file that already exists.
  - In either case, analyze if it is necessary to stop the program! Or is it possible to allow the user to provide the correct file, and continue execution?

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

 10

4.2 Exception Handling

Common Issues

- Some of the common issues in Exception Handling are (contd.):
  - On any DBMS operation, always check for successful completion, or error returned.
    - Complex software can fail in many different ways.
      - For example: access problem, concurrency problem, integrity problem, etc.
  - For code involving “memory allocation”, always check for failure to allocate memory.
    - This is always possible if the application has been running for many hours.
- Some of the common issues in Exception Handling are (contd.):
  - In computations, ensure that there is no “divide by zero”.
  - While using strings, ensure that they are NULL terminated.
  - In languages that support exception handling, ensure you understand how it works. Then use the appropriate constructs like Assert, Throw, Catch, etc.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

4.2 Exception Handling

Common Issues

- Some of the common issues in Exception Handling are (contd.):
  - Check for exceptions at the interfaces between two modules.
    - Is the “calling module” making any assumptions that the “called module” does not guarantee?
    - Is the “called module” making any assumptions that the “calling module” may violate?
  - For example: Suppose that a module performs a “binary search” on an array, which is given as input to the module. Does it assume that the array is sorted? Does the interface definition for this module clearly document about that?

 12

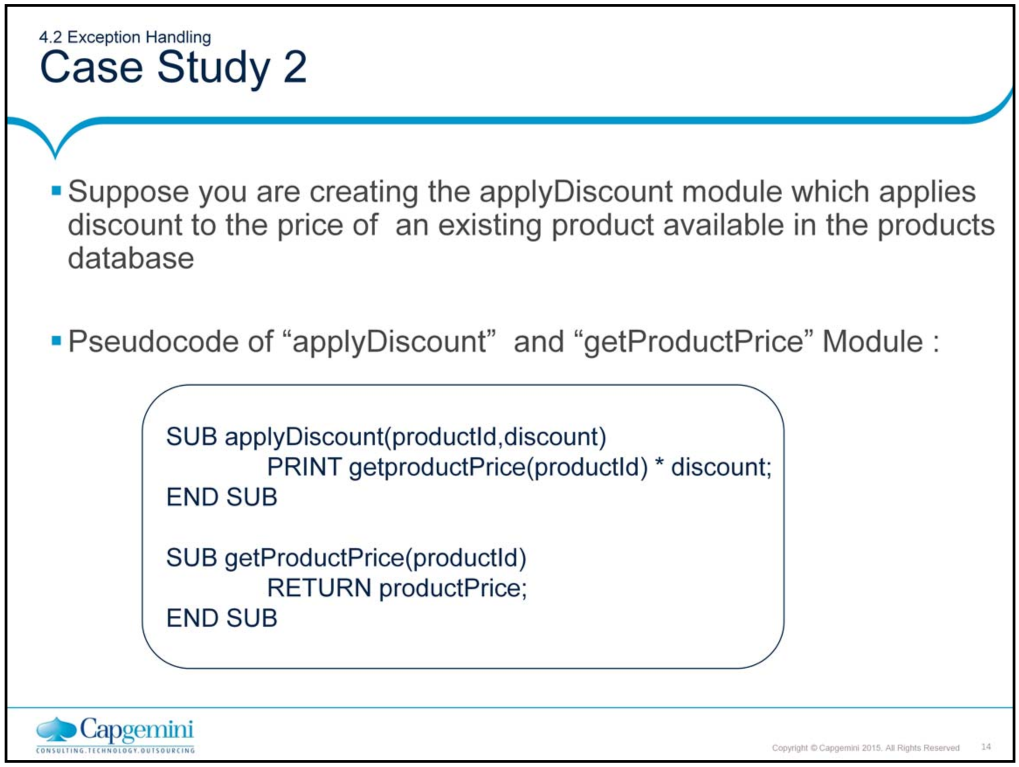
4.2 Exception Handling

Common Issues

- Some of the common issues in Exception Handling are (contd.):
  - In object oriented languages, check for exceptions with respect to the collaborations between objects.
  - Exceptions can either be “handled by a module” or “reported to a higher level module as a return value or parameter”.
    - This should be decided based on the level at which it is possible to take appropriate recovery actions.
- For example:
  - In a Railway Reservation system, if the desired seating is not available, check:
    - whether alternate seating options will be acceptable
    - whether alternate date is acceptable
    - whether alternate train or route will be acceptable

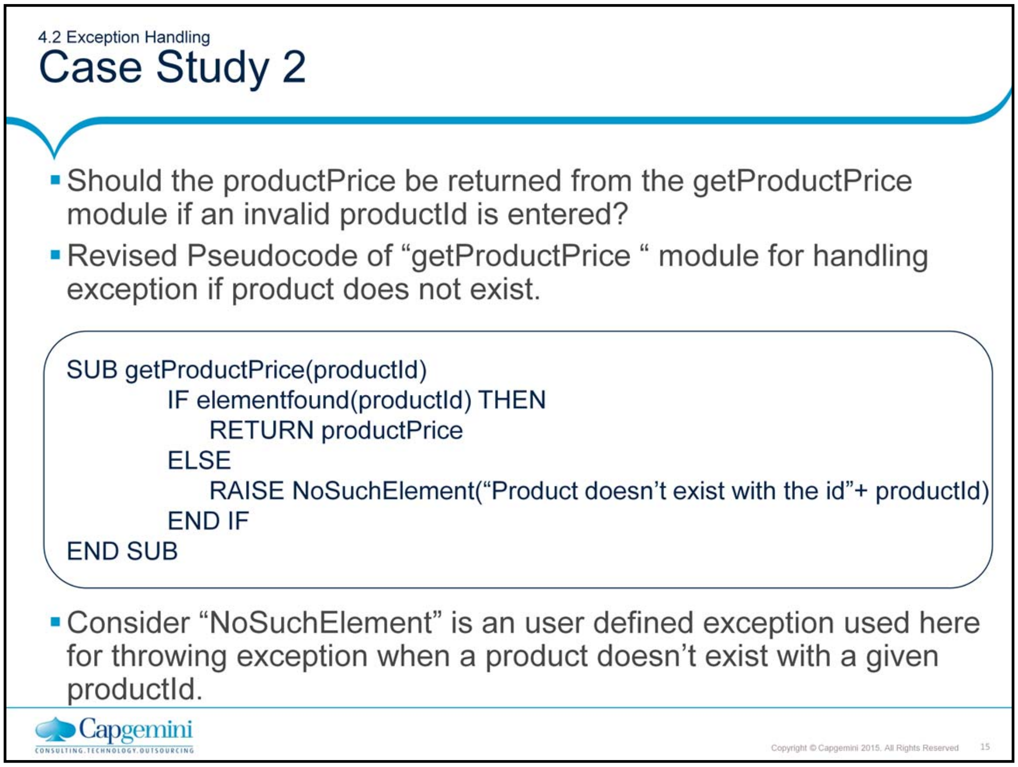
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13



applyDiscount module is used to apply discount on a productprice for a particular product.

The applyDiscount module includes a call to the getproductPrice module which searches for the required product based on the productId and return the price of a product



Description of the pseudocode given in slide

If the product id supplied to the productPrice module does not exist in the database then an exception, “Product does not exist along with productId”, should be raised from the getproductPrice module

The applyDiscount module should catch and handle the exception appropriately

Rasie is a keyword which can be used to throw an exception.

Syntax: RAISE exceptionname(message);

Once if an exception raised from a module, then catch the exception and display the exception message while invoking the module from which the exception is thrown.

Exception messages should be more meaningful along with the entered productId.

4.2 Exception Handling

Case Study 2

- Revised Code of applyDiscount and getProductPrice Module with exception handling

```
SUB applyDiscount(productId,discount)
        PRINT getProductPrice(productId)*discount
EXCEPTION
        WHEN NoSuchElement THEN
                PRINT errormessage //Errormessage returned from exception
END SUB
SUB getProductPrice(productId)
        IF elementfound(productId) THEN
            RETURN productPrice
        ELSE
            RAISE NoSuchElement(“Product doesn’t exist with the id”+ productId)
        END IF
END SUB
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

EXCEPTION section will be executed whenever exception occurs. Using WHEN section exception can be handled.

If the given productId exists in database, then updated price will be returned else “Product doesn’t exist with the given productId “ message will be printed.

4.2 Exception Handling

- Depending on the "degree of criticality", the developer is required to design the exception handlers:
  - to give meaningful error messages, or
  - to provide alternatives and options, or
  - to provide recovery options

 17

4.3 Defensive Programming

What is Defensive Programming

- Defensive programming – program tries its best to provide service despite errors or unexpected conditions
  - It is like defensive driving. “Should you drive on the assumption that all other drivers will follow the rules?” or, “Should you play safe by assuming they will do the unexpected at least once in a while?”
  - Similarly, good programmers do not assume that users will always do the expected thing, i.e. pass the right type of parameters, initialize variables, etc.

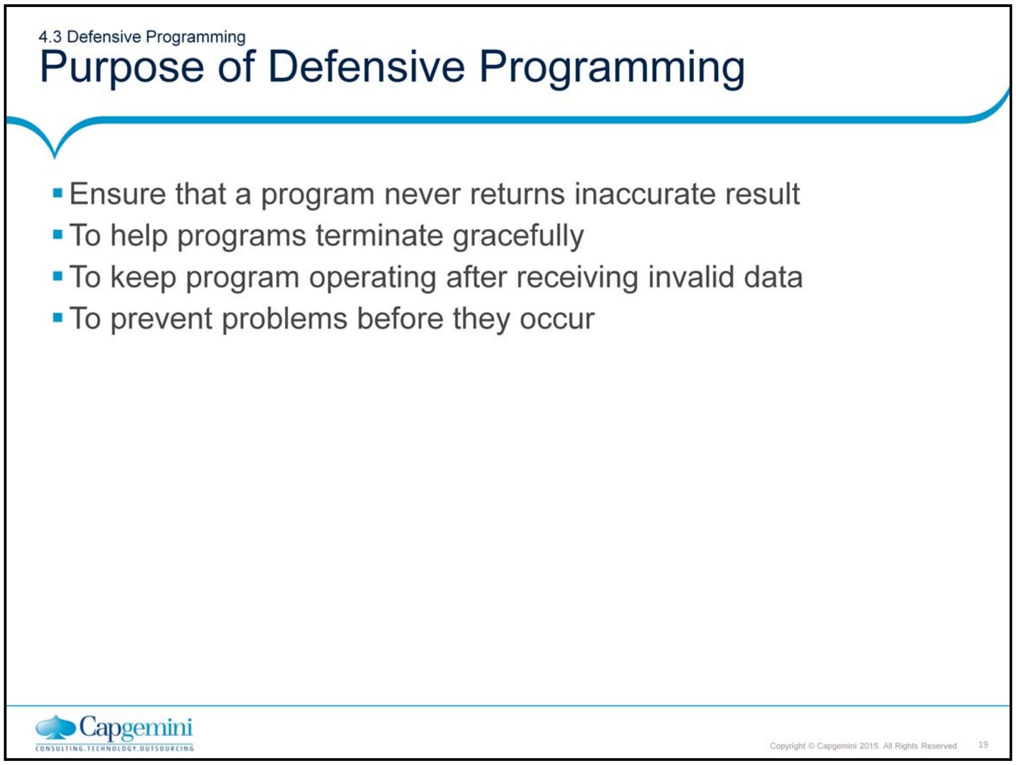
Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Defensive programming enables us to detect minor problems early on, rather than get bitten by them later when they’ve escalated into major disasters.

Defensive programming is a method of prevention, rather than a form of cure. Compare this to debugging—the act of removing bugs *after* they’ve bitten. Debugging is all about finding a cure.

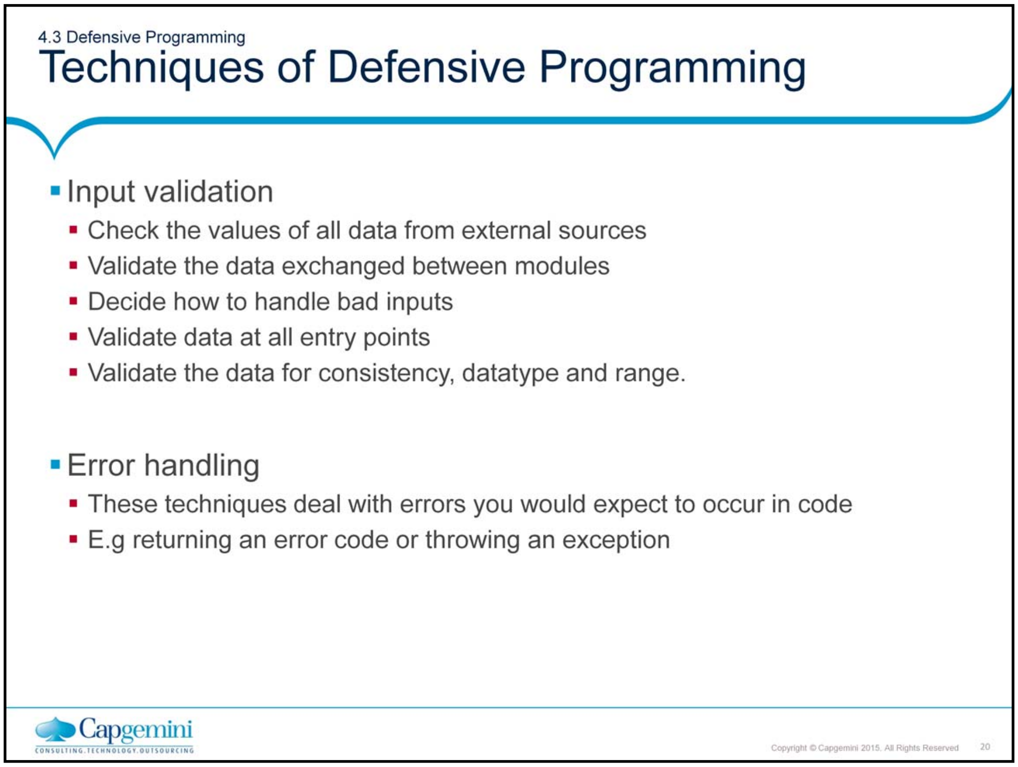
Defensive Programming is not

- Error checking
- Testing
- Debugging



Purpose of Defensive Programming

- Ensure that a program never returns inaccurate result even though valid data is passed.
- Abnormal termination of the program will be avoided
- Even though, invalid data is passed instead of abnormal termination of program execution, meaningful error messages has to be displayed.
- Thinking all the possible and impossible scenarios and take care of exception to prevent problem occurrence.

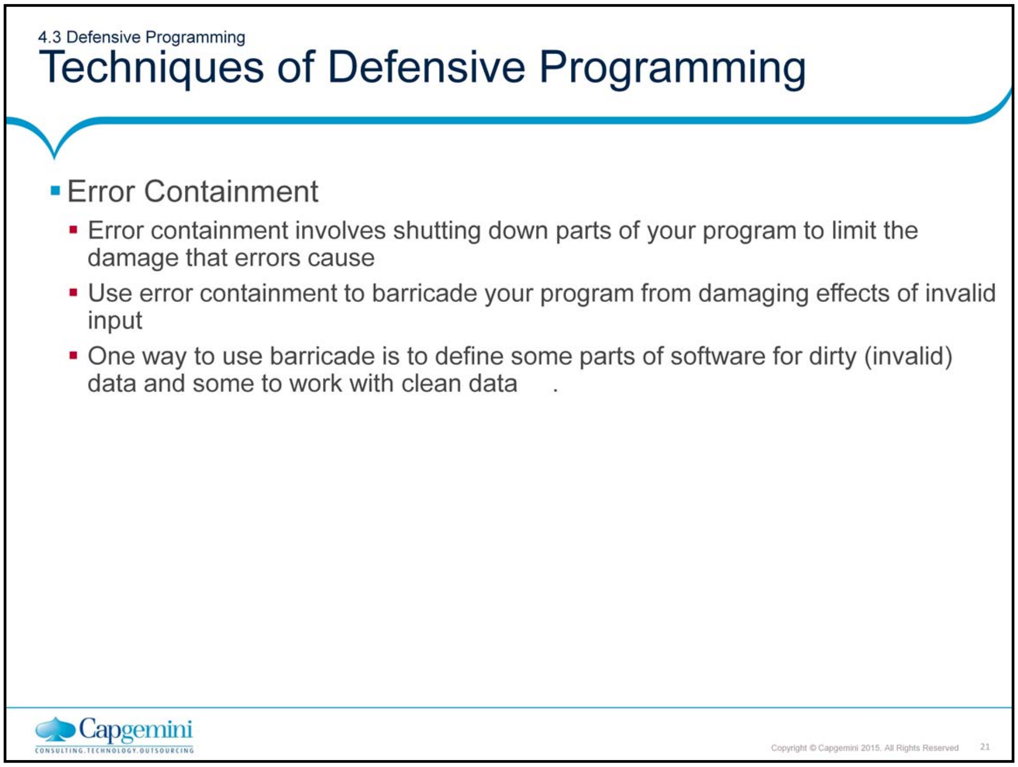


Techniques of Defensive Programming

1. **Input validation**

- **Check the values of all data from external sources :** When getting data either from a file or from a user, check to be sure that the data falls within the allowable range and correct type of value is accepted.
- **Validate the data exchanged between modules:** Checking the values of routine input parameters is essentially the same as checking data that comes from an external source
- **Decide how to handle bad inputs:** Once you have detected an invalid input, take care of necessary technique to display an error message or return error code.
- Validate the data for
    - **consistency :** The consistency check compares new data with previous data. For example, a current meter reading against past meter readings.
    - **data type :** The data type check ensures input data is of the correct data type. For example, numeric or alphabetic.
    - **range :** check ensures that input data is within a specified range.

2. **Error handling :** these techniques deal with errors you would expect to occur in code



Techniques of Defensive Programming(Contd..)

Example for Error Handling:

```
IF (fileExists)  THEN
        Read the data from the file
ELSE
        errorCode = -1; //File doesn't exists
        return errorCode
END IF
```

3. Error Containment

- Protect your code from an invalid data coming from "outside"(Data from an external system or the user or a file)
- Establish "barricades"(module where validation logic exists) for leaving dangerous data outside of the boundary(like before invoking module), everything inside of the boundary(within module) is safe. For an example, send input parameters to a module only if it is valid(Inside of the boundary is safe)
- In the barricade code(isValid Module), validate all input data (check all input parameters) for the correct type, length, and range of values. Double check for limits and bounds.
- **For an Example:**

```
    IF(data is invalid) THEN
        PRINT error message
  ELSE
        Invoke a module to execute logic
  END IF
```

Summary

- Exceptions are powerful error handling mechanism
- Developer is required to design the exception handlers:
  - to give meaningful error messages, or
  - to provide alternatives and options, or
  - to provide recovery options
- Defensive programming is used to ensure the continuing function of a piece of software under any circumstances.
- Techniques of Defensive Programming
  - Input validation
  - Error Containment
  - Error handling

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

Review Questions

Question 1 : What is the main purpose of input validation ?

A. To ensure that data exists in a field
B. To ensure that input data is of correct datatype
C. To ensure that input data is within a specified range
D. To ensure that pre input check data is correct

Question 2 : What is the purpose of defensive programming ?

A. Ensure that a program never returns inaccurate result
B. To help programs terminate gracefully
C. To keep program operating after receiving invalid data
D. To prevent problems before they occur

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Review Questions

Question 3 : What is the purpose of exception handling ?

A. To provide meaningful error messages

B. To provide alternatives

C. To separate error code from regular code

D. All of the above

Question 4 : An exception is an event that occurs during the execution of a program that disrupt its normal course.

A. True

B. False

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

 24

Programming Foundation

Lesson 5: Software Reviews and Testing

Lesson Objectives

- To Understand the following concepts
  - What is software Testing?
  - What is Debugging?
  - Software Testing Principles
  - TestCase
  - Exhaustive Testing
  - Testing Techniques
  - Static Testing
  - Dynamic Testing
  - Testing Approaches
    - Unit Testing
    - Integration Testing
    - System Testing
    - Verification and Validation testing
    - Acceptance Testing
    - Regression testing

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

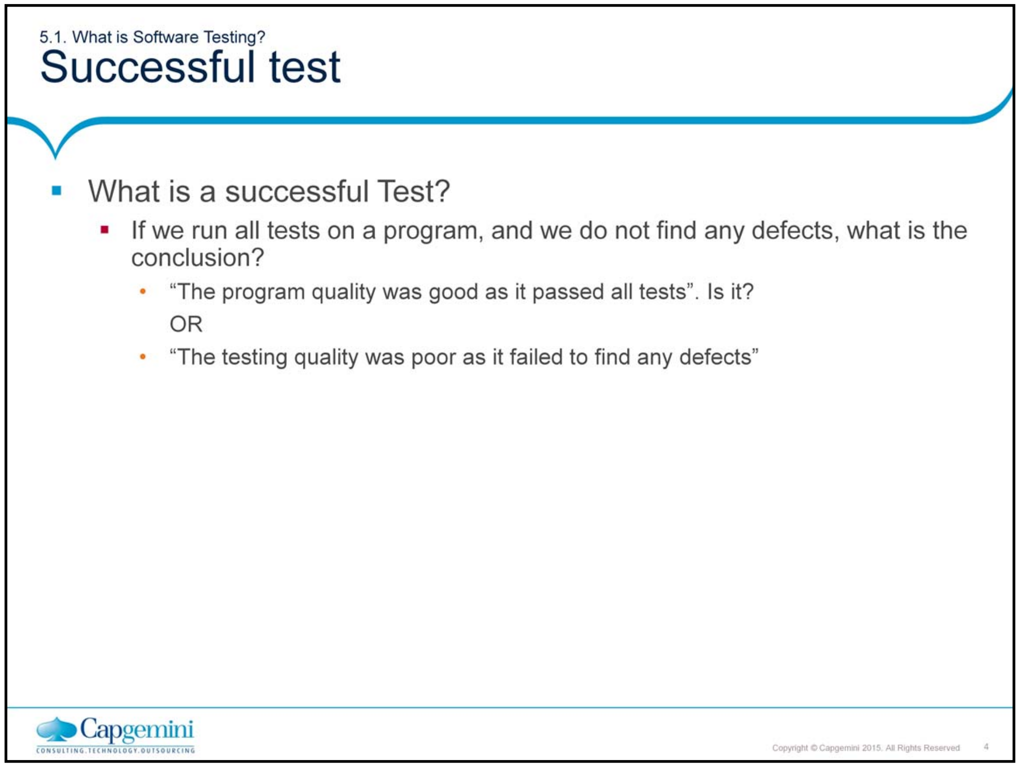
5.1. What is Software Testing?

- Testing is the process of executing a program with the intent of finding errors
- Testing is a process used to help identify the correctness, completeness and quality of a developed computer software
- Testing helps is Verifying and Validating if the Software is working as it is intended to be working

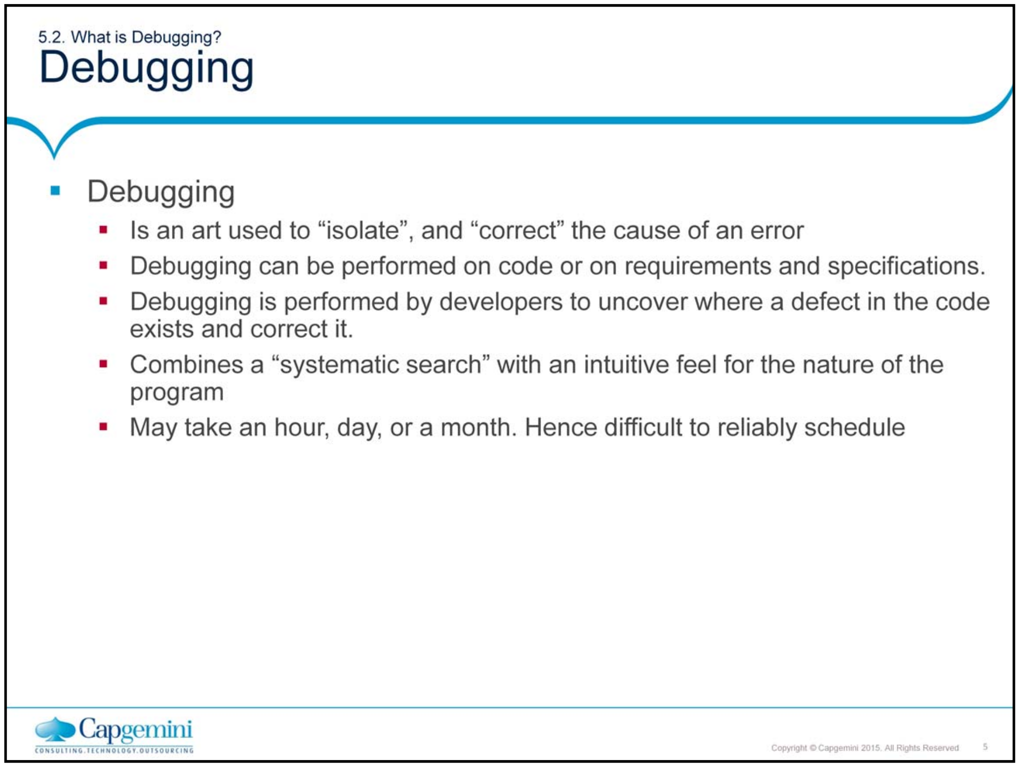
What is software testing?
Exercising (analyzing) a system or component with

- defined inputs
- capturing monitored outputs
- comparing outputs with specified or intended requirements

To maximize the number of errors found by a finite no of test cases.
Testing is successful if you can prove that the product does what it should not do and does not do what it should do.



Testing is in a way a destructive process and a successful test case is one that brings out an error in the program . Detection of an error/failure is a success



Objective of debugging is to find and correct the cause of the software error. Debugging is not testing.

The symptom may appear in one part of a program, while the cause may actually be located at a site that is far removed.

The symptom may be caused by round off in accuracies.

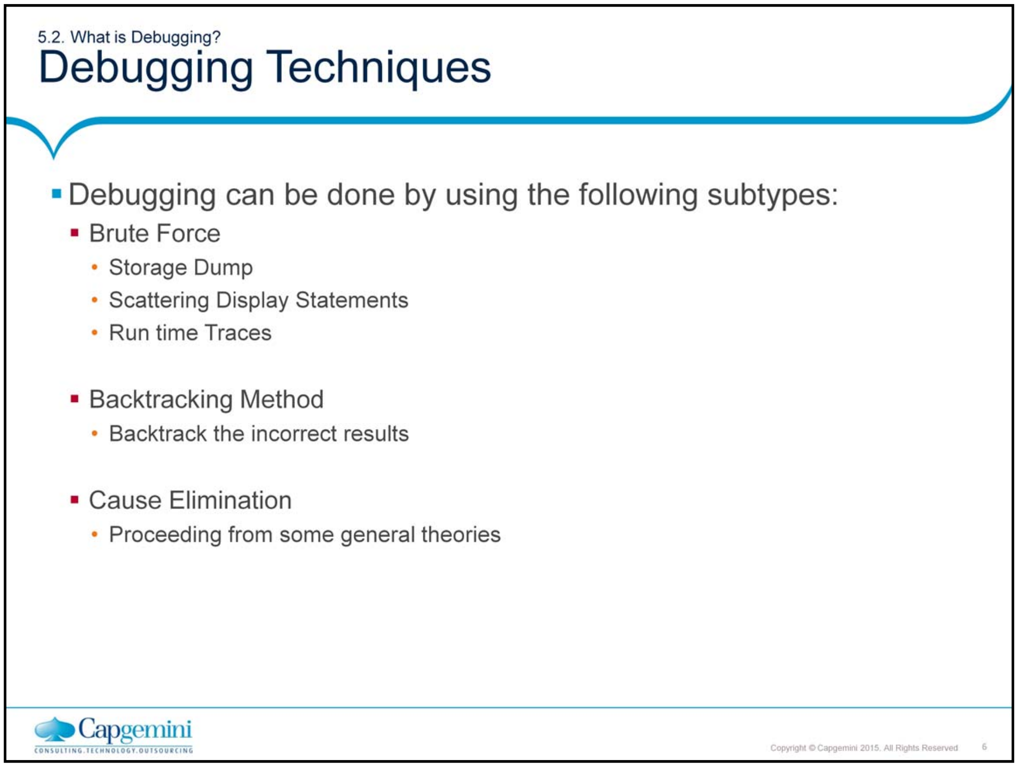
Intermittent problems in embedded systems because hardware is tightly coupled with software

As the consequences of an error increase, the amount of pressure to find the cause also increases. Often, pressure forces a software developer to fix one error while at the same time introducing two more.

The debugging process has one of the two outcomes:

1. The cause will be found, corrected and removed.
2. The cause will not be found.

In the second outcome, the person performing debugging may suspect a cause, design a test case to help validate his or her suspicion and work toward error correction in iterative manner



Brute Force: It is the most common and least efficient method for isolating the cause of a software error. We apply brute force debugging methods when all else fails.

Example of debugging by Brute Force are

1. By studying Storage Dumps I.e. usually a crude display of storage location
2. by invoking run-time traces
3. by scattering print statements
4. by use of automated debugging tools

Backtracking is a common debugging approach. It is basically used in small programs. The program code is manually tracked beginning at the place where the symptom is uncovered, the source code is traced backward until the site of the cause is found.

Cause elimination

Debugging by Induction

1. Locate data about what program did correctly/incorrectly
2. Organize data
3. Device a hypothesis about the cause of the error
4. Prove the hypothesis

Debugging by deduction

1. Enumerate the causes of error
2. Eliminate each cause of error

5.2. What is Debugging?

Comparison

- Purpose of Testing
  - To show that a program has a bug
- Purpose of Debugging
  - To find and correct the cause of an error

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

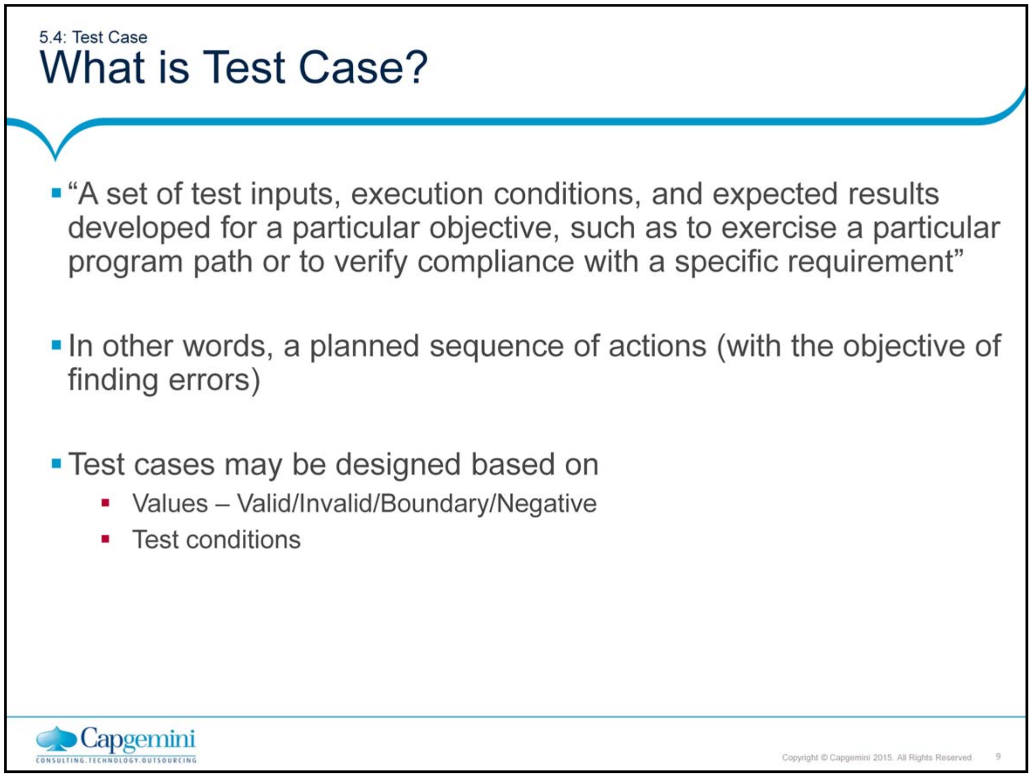
 7

5.3Testing Principles

- A necessary part of a test case is a definition of the “expected output” or “result”
- Test cases must be written for “invalid and unexpected” as well as “valid and expected” input conditions
- The probability of finding “more defects” in a module is proportional to the “number of defects already found” in that module
- In most systems, 20% of the modules account for 80% of the defects found
- A programmer should not be the only person to test his or her own program

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8



A Test case is a planned sequence of actions.

Characteristics of a Good Test:

They are: likely to catch bugs
not redundant
not too simple or too complex.

5.4: Test Case

Example

- Problem:
  - Given the lengths of three sides of a triangle, determine whether a valid triangle is formed.
  - If valid, determine the type of triangle – equilateral, isosceles, or scalene.
  - Develop the code for the above problem.
  - Identify all the test cases required to test the code, by using the following headers:
    - Test Case #, Test Case description, Expected result

While testing the code consider valid expected and invalid unexpected scenarios.

5.4: Test Case

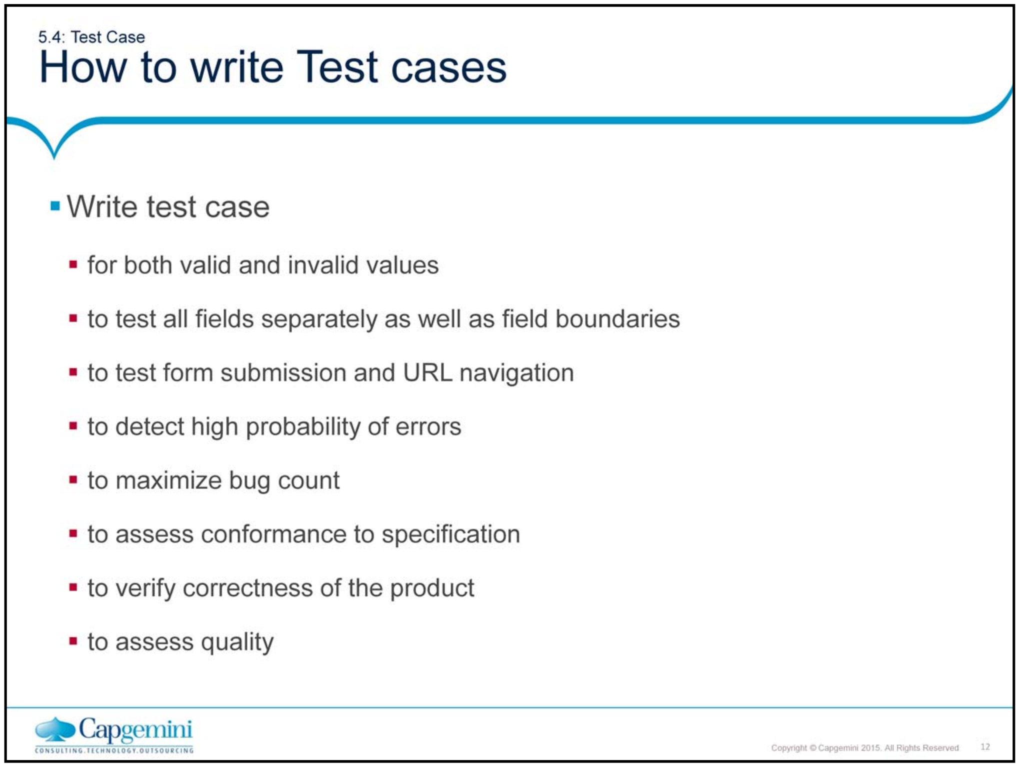
Test cases - Example

- Valid Test cases:
  - Scalene 3,4,5; 4,5,3; 5,3,4;
  - Isosceles 3,4,4; 4,3,4; 4,4,3;
  - Equilateral 3,3,3; 4,4,4 5,5,5
- Invalid test cases
  - Scalene 3,3,a 3,4,-1 1,2,0
  - Isosceles 3,-1,3 1,2,3 3,4,0
  - Equilateral 0,0,0; -1,-1,-1

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

If you observe the examples given on the slide, Some of the input values are checking for valid values for scalene, isosceles, equilateral triangle. and some of the input values are checking invalid values like

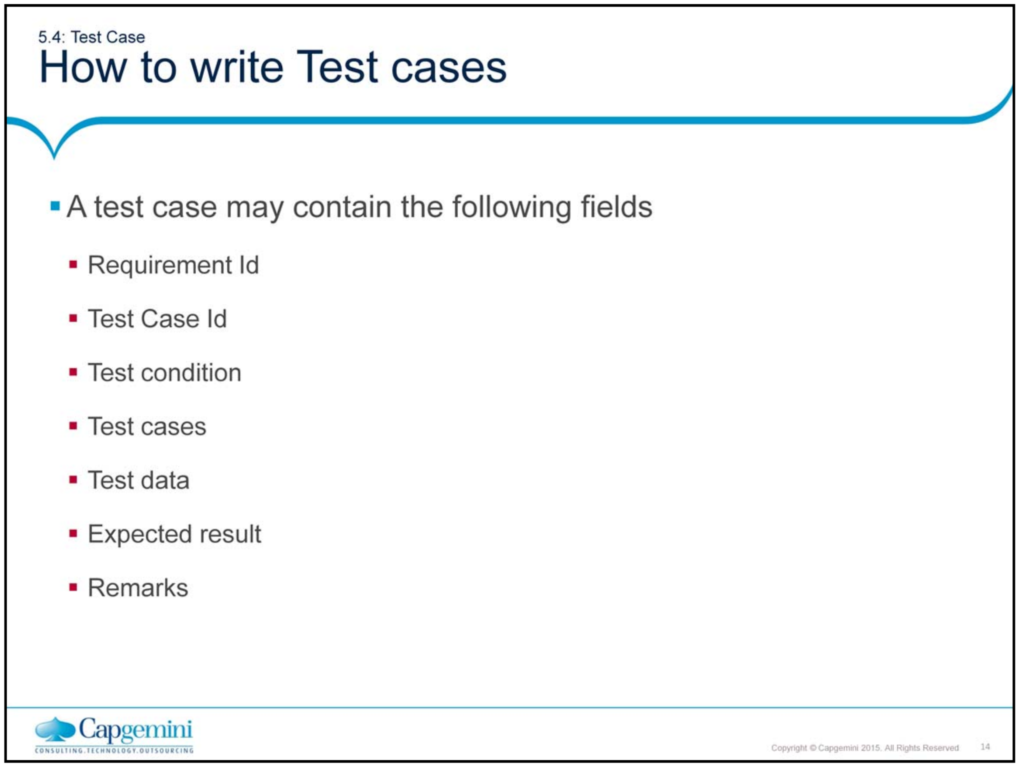
- Length of a triangle cannot be negative
- Length of triangle cannot be zero
- Length of a triangle cannot be alphabet etc



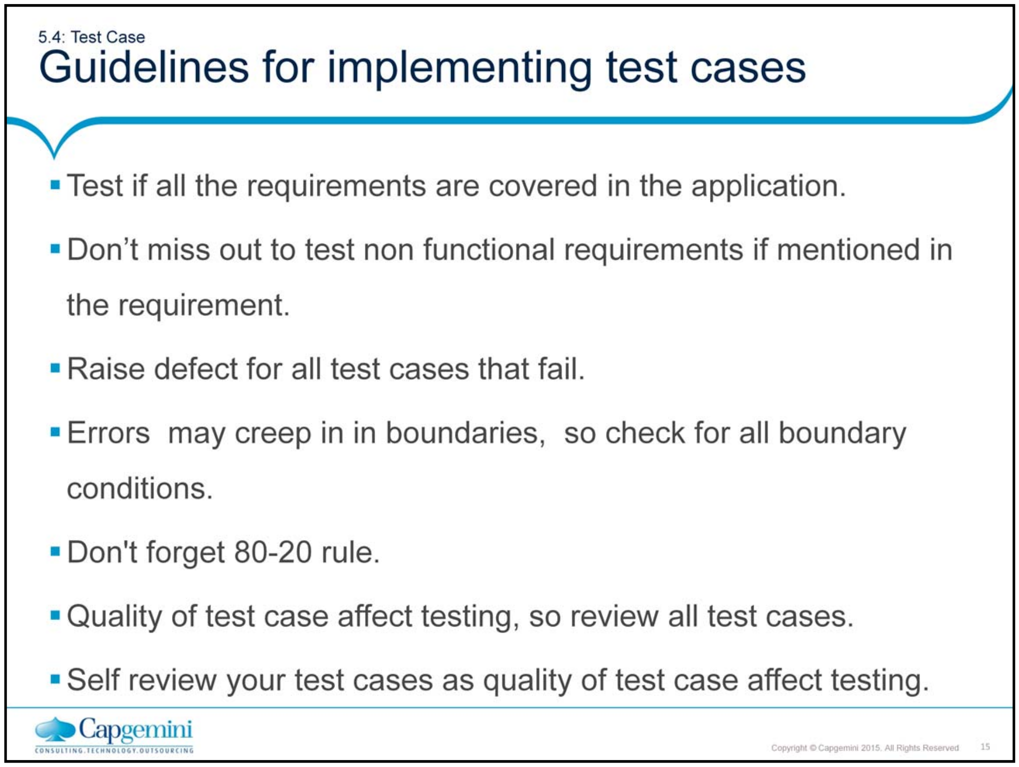
How to write test case?

- Write test case for both valid and invalid values. For an Example, if you want to validate the salary, then consider the below mentioned test cases.
  - write test case with valid input data as "10000".
  - write test case with invalid input data as "Test".
- Write test case to test all fields separately as well as field boundaries. For an Example, consider a login form contains two fields like username and password. Write both valid and invalid test cases for all the fields(i.e, Username and Password) available in the page.
- Write test case to test form submission and URL navigation. For an Example, consider a login form contains two fields like username and password. If you want to navigate to the next page, while clicking on submit button after entering valid username and password, then prefer writing test case for form submission.
- Defect high probability of errors: This is the classic objective of testing. A test is run in order to trigger failures that expose defects. Generally, we look for defects in all interesting parts of the product.
- Maximize bug count: The distinction between this and “find defects” is that total number of bugs is more important than coverage. We might focus narrowly, on only a few high-risk features, if this is the way to find the most bugs in the time available.

- Assess conformance to specification. Any claim made in the specification is checked. Program characteristics not addressed in the specification are not (as part of this objective) checked.
- Verify correctness of the product. It is impossible to do this by testing. You can prove that the product is not correct or you can demonstrate that you didn't find any errors in a given period of time using a given testing strategy. However, you can't test exhaustively, and the product might fail under conditions that you did not test. The best you can do (if you have a solid, credible model) is assessment--test-based estimation of the probability of errors. (See the discussion of reliability, above).
- Assure quality. Despite the common title, quality assurance, you can't assure quality by testing. You can't assure quality by gathering metrics. You can't assure quality by setting standards. Quality assurance involves building a high quality product and for that, you need skilled people throughout development who have time and motivation and an appropriate balance of direction and creative freedom. This is out of scope for a test organization. It is within scope for the project manager and associated executives. The test organization can certainly help in this process by performing a wide range of technical investigations, but those investigations are not quality assurance. Given a testing objective, the good test series provides information directly relevant to that objective. Different types of tests are more effective for different classes of information.



Test case has to be written to validate the testing coverage of the application.
Requirement Id: The ID of the requirement this test case relates/traces to.
Test Case Id: Unique ID for each test case. Follow some convention to indicate types of test. E.g. ‘TC_UI_1′ indicating ‘user interface test case #1′.
Test condition: Describes what to test for i.e. the condition which is being tested for example, Validate name
Test cases: Step-by-step procedure to execute the test.
Test data: Use of test data as an input for this test case. You can provide different data sets with exact values to be used as an input
Expected result: What should be the system output after test execution? Describe the expected result in detail including message/error that should be displayed on screen
Remarks: Any comments on the test case or test execution.



Guidelines for implementing test cases:

- Write test case for all the requirements specified in the application
- Take care of writing test case for non functional requirements like security, performance, etc..
- If any test case fails, log the failed test cases as defect in defect tracking sheet.
- Check for all boundary conditions.
- **80-20 Rule:** In most systems, 20% of the modules account for 80% of the defects found. The probability of finding defect in a module is directly proportional to the number of defects already found in the module.
- Do self review and peer review for all test cases as quality of test case affects testing.

Demo : Test Case creation

- Test case example

Refer Hotel Bookings Management System.doc and Test_Case-HotelBookingsSystem.xls file to understand test case example.

5.5 Exhaustive and Economics testing

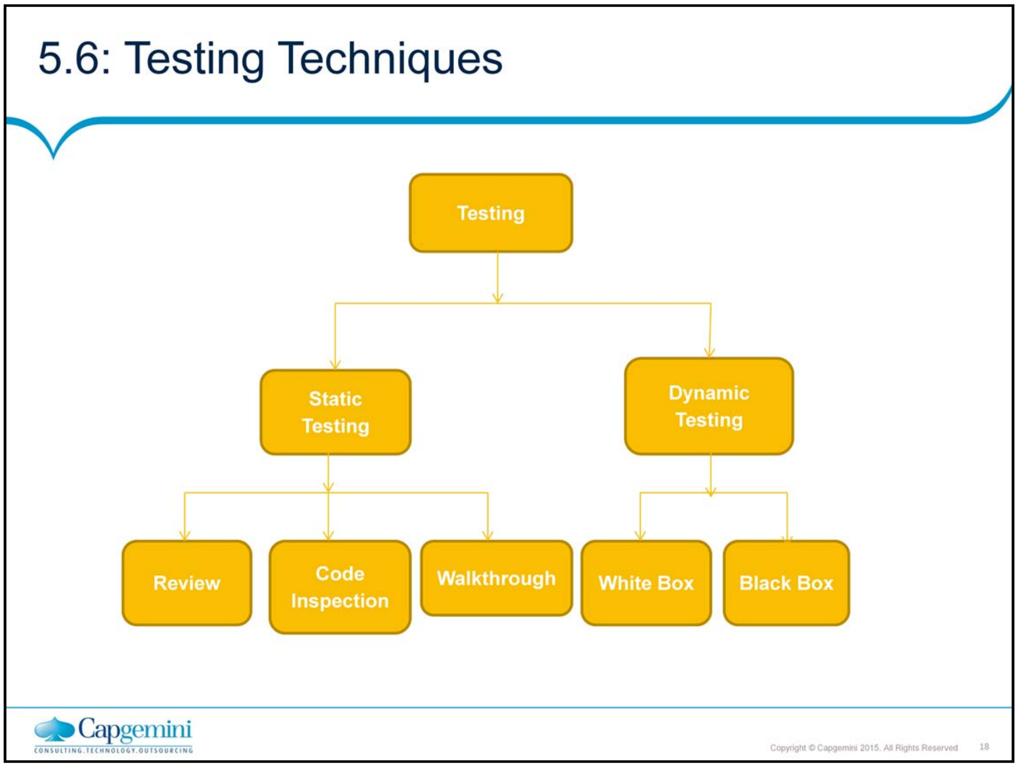
- Exhaustive Testing
  - Exhaustive Testing involves testing for every possible input, and every possible output
  - For example: Online railway reservation system
    - It is impossible to test the ticket booking for all possible combination of sources and destinations
    - Hence we test the code with some sample values
  - However, “Exhaustive Testing” is impractical, and not economically viable
  - Therefore, the objective of testing is to find “maximum errors” with a finite number of test cases

- Economics of Testing
  - It is both the driving force and the limiting factor
  - Driving - Earlier the errors are discovered and removed in the lifecycle, lower the cost of their removal.
  - Limiting - It is infeasible to test exhaustively all possible combinations.

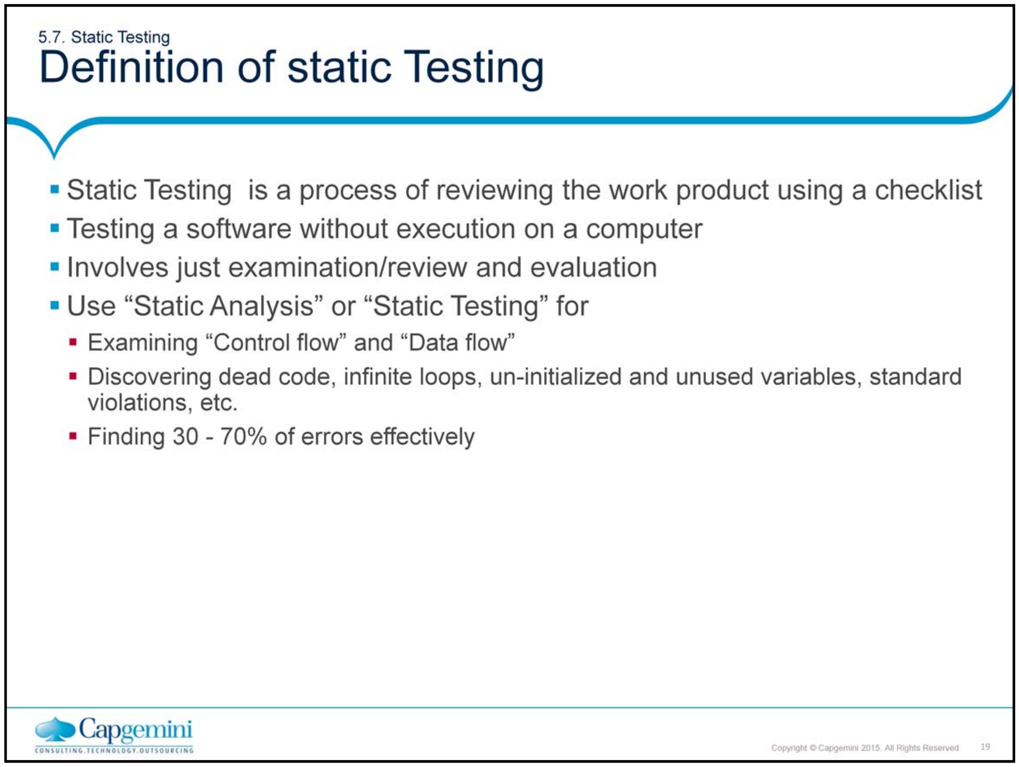
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Exhaustive testing is impossible.
For E.g. COBOL Compiler
•Impossible to create test cases to represent all valid cases.
• Impossible to create test cases for all invalid cases
The compiler has to be tested to see that it does not do what it is not supposed to do E.g. to successfully compile a syntactically incorrect program

But writing Cobol programs to include all possible syntactical errors that compiler check is impractical. It is time consuming and hence not suggestible.

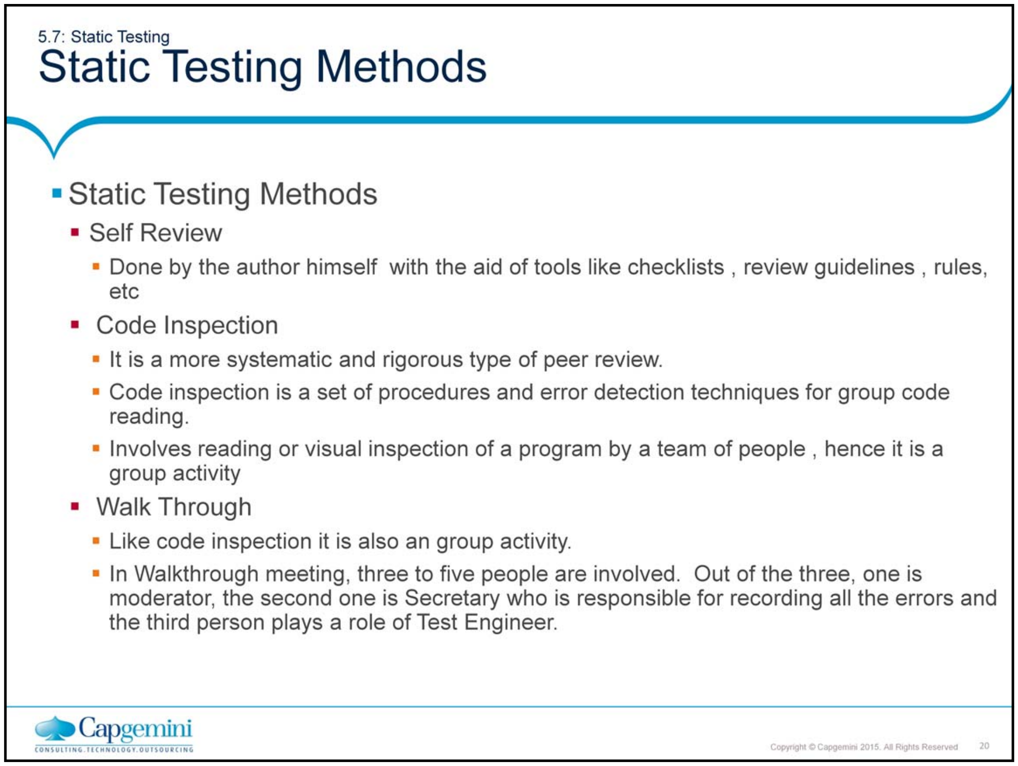


- Static Testing: Testing a software without execution on a computer
    - Review : Review the created artifacts using checklist
    - Code Inspection : Code inspection is a set of procedures and error detection techniques for group code reading.
    - Walkthrough : Like code inspection it is also an group activity.
- Dynamic Testing techniques exercise the software by using sample input values
    - WhiteBox testing : Used to test the internal structure of the code
    - BlackBox Testing : Test the functionality of application by providing input and getting expected output



Static Testing:

- Static Testing is a process of reviewing the work product using a checklist.
- No need to execute a program for performing static Testing.
- Static testing may be conducted manually or through the use of various software review tools like PMD, Checkstyle. Specific types of static software testing include code analysis, inspection, code reviews and walkthroughs.
- Through static testing, errors in the control flow/data flow can be identified at the earlier stage.



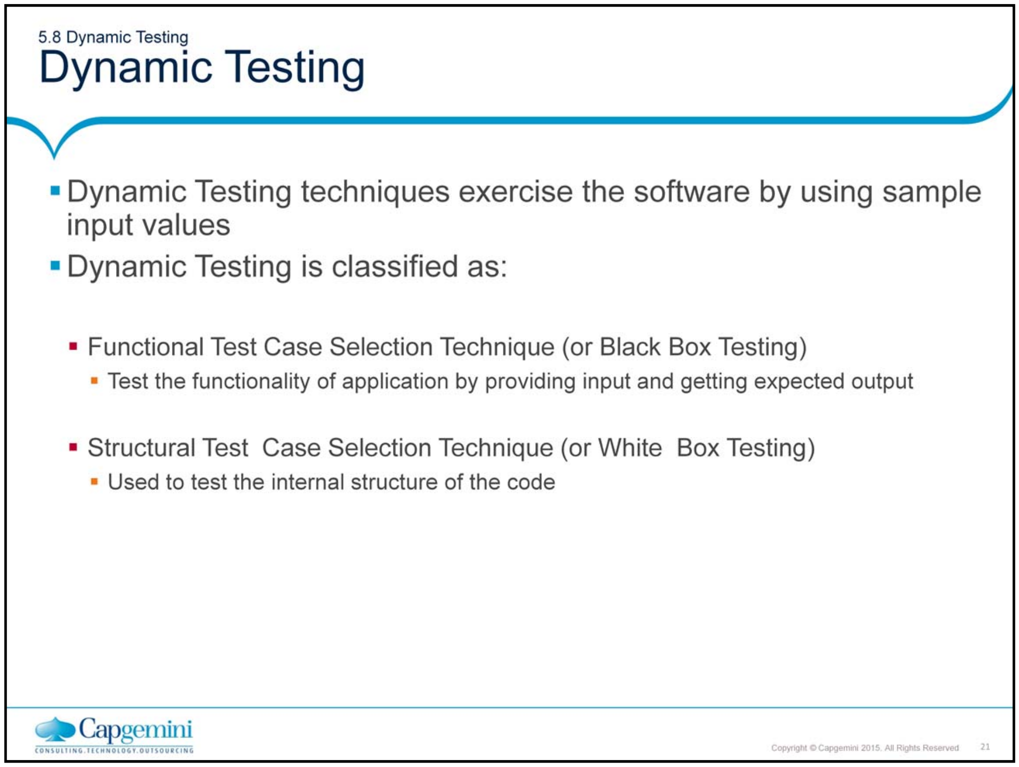
Review Process

Input : Work Product , Specifications, Checklists, Guidelines, Historical Data

Process

- Prepare for Review
- Conduct Reviews
- Analyze Deviations
- Correct Defects

Output : Review Form, reviewed work product



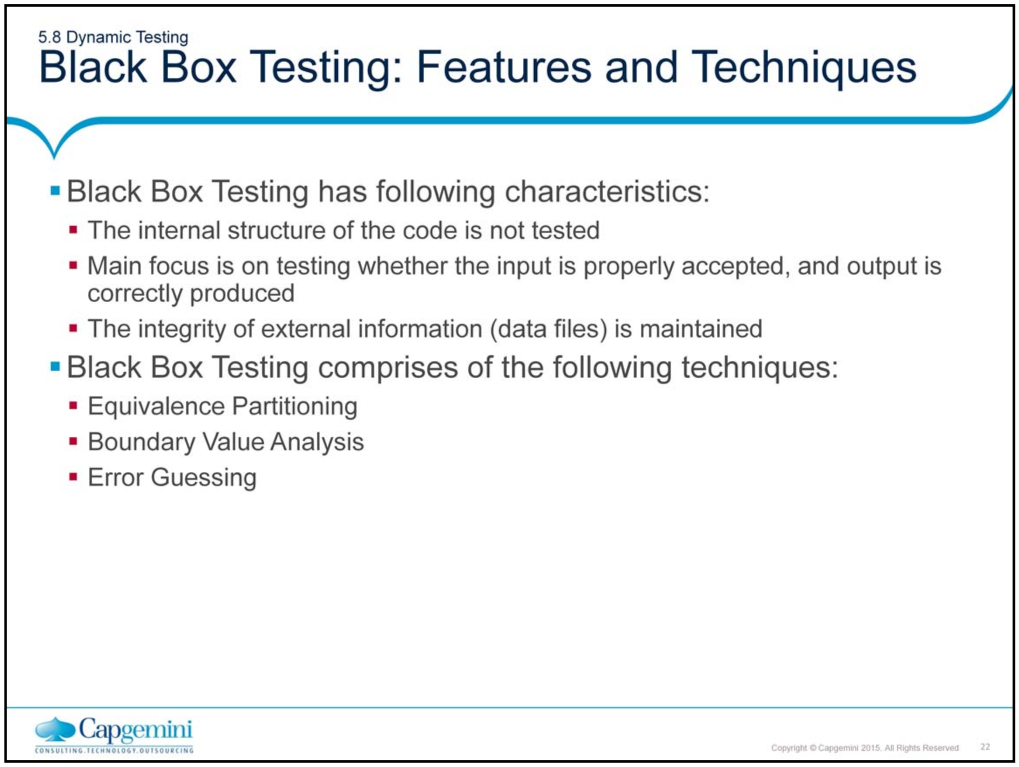
Dynamic Testing

- Dynamic Testing involves the testing of a software by executing the system.
- Using Dynamic Testing, internal structure and functionality of an application will be tested.
- Perform white box testing for testing the internal structure of the code
- Test the functionality of an application using Black box testing.

For an Example, if you want to validate all the fields in the login page to accept valid data, then perform black box testing.

Steps to be performed for black box testing are:
1. Execute an application
2. Type the input
3. Validate the actual result against the expected result mentioned in the test plan.
4. If actual result and expected result matches, then the test case result is pass else the result is fail.

For an Example, if you want to check whether the logic(code) of login functionality is working fine, then perform white box testing.



E.G
While testing washing machine, you need not be aware of internal structure of washing machine.
We need to know how to use interface and give instructions to washing machine.

Hence in black box testing we need to know what is input and what is output.

5.8 Black Box Testing

Equivalence Partitioning

- Equivalence Partitioning is a Black Box Testing method
  - It divides the input domain of a program in to classes of data
  - Test cases can be derived from these classes
- An ideal test case:
  - Single handedly uncovers a “class of errors”, which might otherwise require many cases to be executed
  - It thereby reduces the number of test cases that must be developed

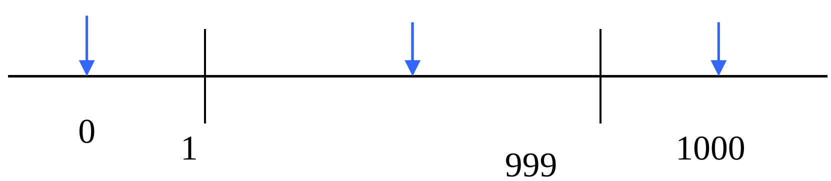
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Examples

If an input condition specifies that a variable, say ***count,*** can take range of values(1 - 999),

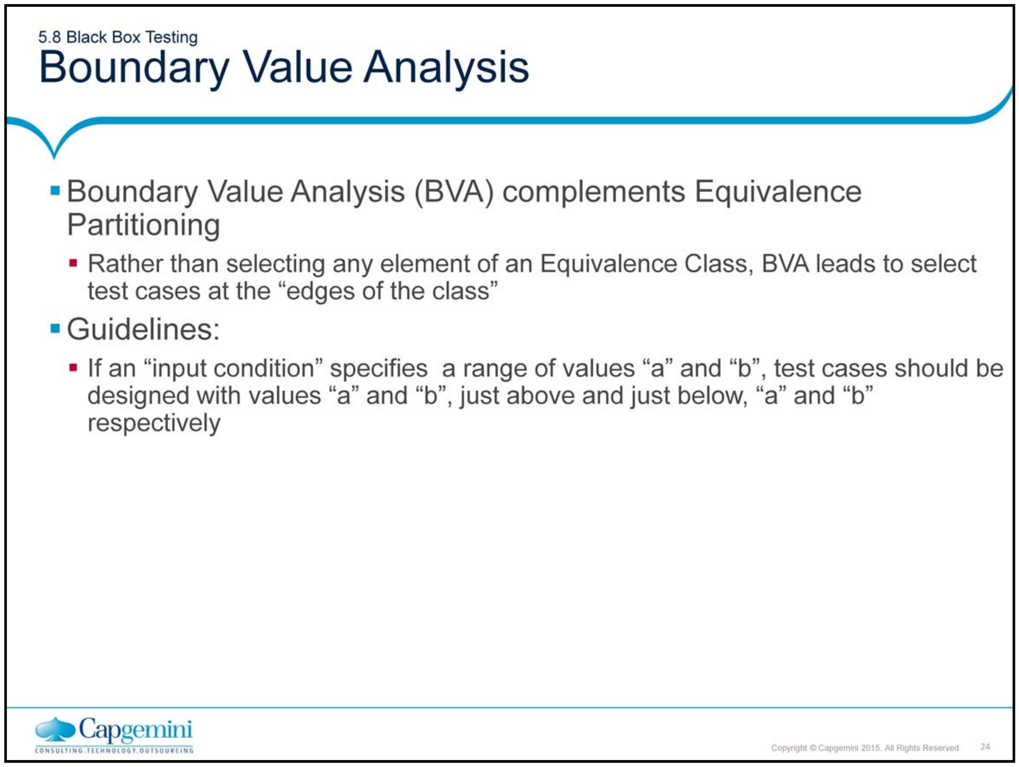
Identify - one valid equivalence class (1 < ***count*** < 999)

- two invalid equivalence classes (***count*** < 1) & (***count*** >999)

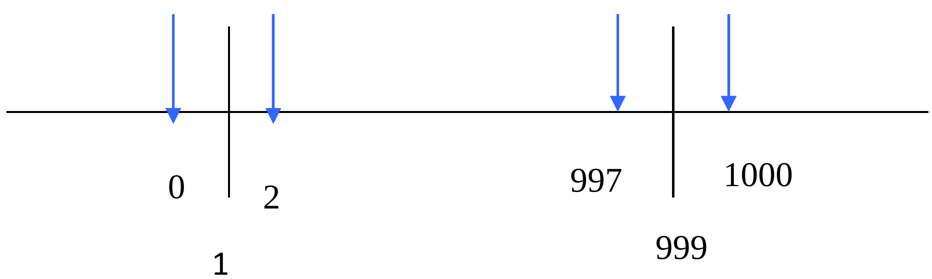


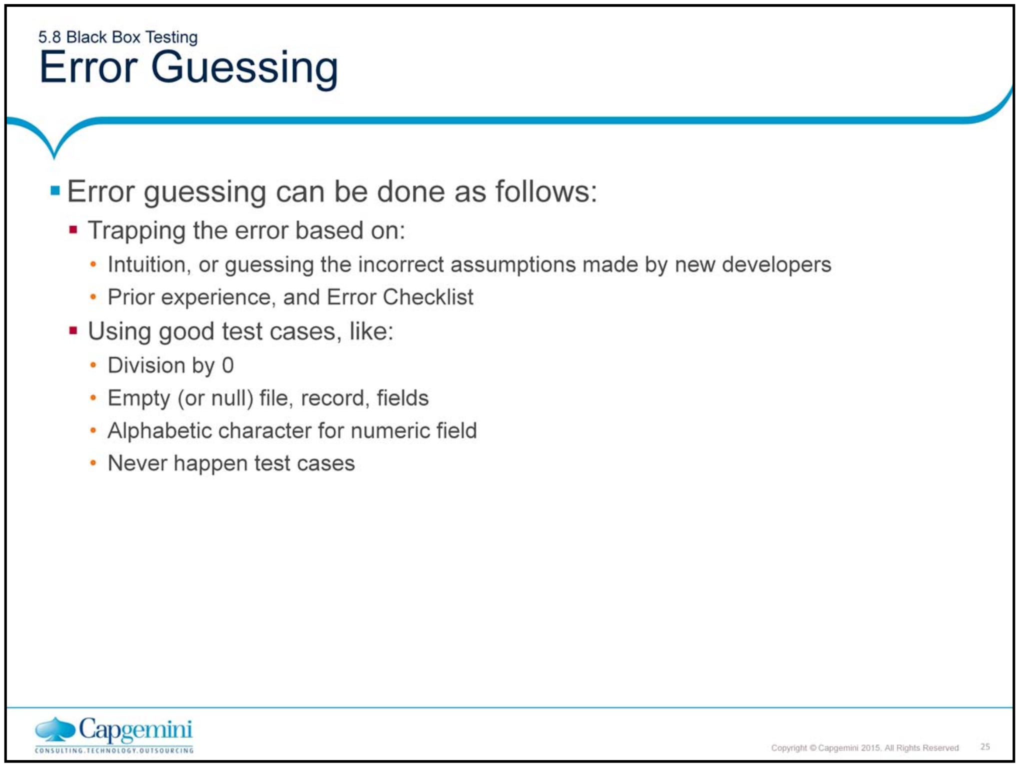
Equivalence classes may be defined according to the following guidelines.

1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
2. If an input condition requires a specific value, one valid and two invalid EC are defined.
3. If an input condition specifies a member of a set, one valid and one invalid EC are defined.
4. In an input condition is Boolean, one valid and one invalid class are defined.



From **previous example,** we have the valid equivalence class as (1 < ***count*** < 999).
Now, according to boundary value analysis, we need to write test cases for ***count=0, count=1, count=2, count=995, count=999 and count=1000 respectively***





Examples

Suppose we have to test the login screen of an application. An experienced test engineer may immediately see if the password typed in the password field can be copied to a text field which may cause a breach in the security of the application.

Error guessing testing for sorting subroutine situation

- The input list empty
- The input list contains only one entry
- All entries in the list have the same value
- Already sorted input list

5.8 White Box Testing

White Box Testing : Features

- White Box Testing focuses on the internal structure of the software. It determines the “predicted outcome” for a “given input”
- White Box Testing examines:
  - existence of non-executable paths
  - presence of infinite loops
  - consistency of logic on true and false sides
  - validation of internal data structures
- White Box Testing comprises of the following techniques:
  - Control Structure Testing
    - Coverage
    - Loop Testing
  - Path Testing
  - Data Flow Testing

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 26

5.8 White Box Testing

Control structure Testing

- Control structure testing is a group of white-box testing methods
- Control Structure Testing comprises of the following two techniques:
  - Coverage
    - Statement Coverage
    - Decision Coverage
    - Conditional Coverage
  - Loop testing

“Coverage” is a measure of how thoroughly a program is exercised

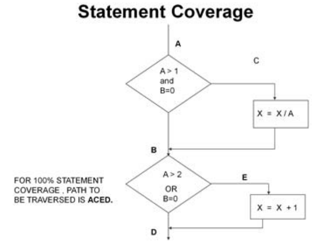
5.8 White Box Testing

Statement Coverage

- Statement Coverage

Test Case: A=2,B=0,
•Every statement will be executed once.
•But only path ACE will be covered and path ABD,ACD,ABE will not be covered.

Statement Coverage

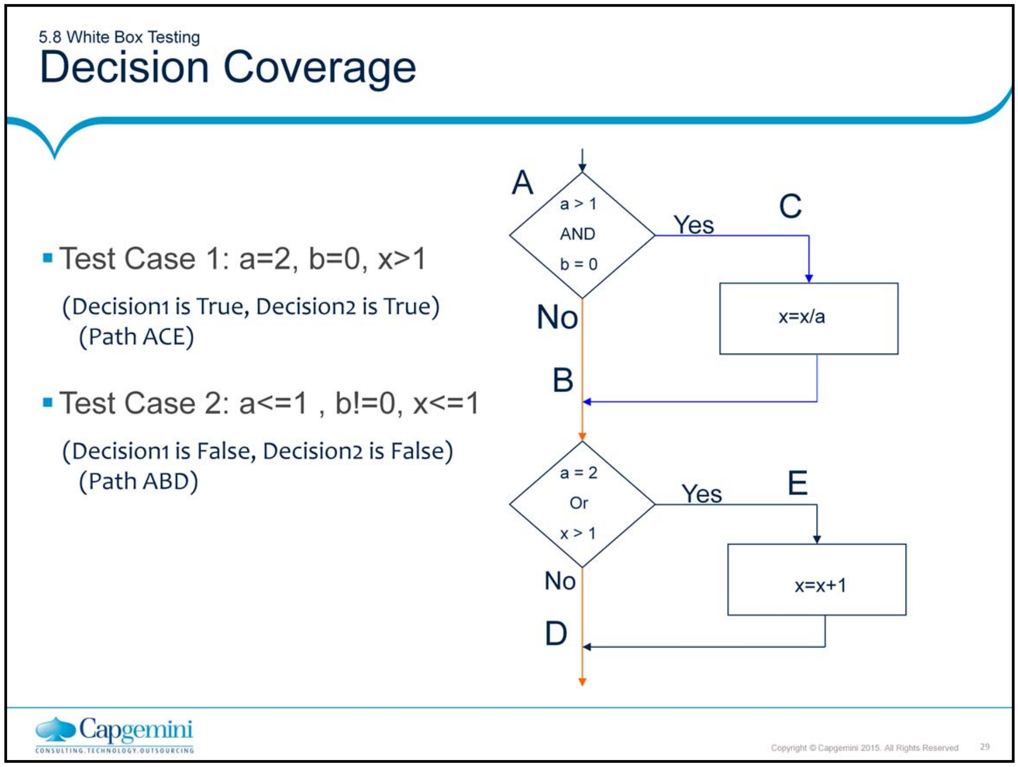


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 28

Statement Coverage:
Every statement can be executed by writing a single test case. This case covers only ACE path.
This criteria is weak one. Since it is not considering other paths to traverse. So the path ABD, ACD, ABE would go undetected.

```
BEGIN
        PRINT "Enter 2 numbers"
        READ a and b
        If (a>1) && (b=0) THEN
                x=x/a;
        ELSE IF (a=2 || x>1) THEN
                x=x+1;
        END IF
        PRINT x
END
```



Decision Coverage:

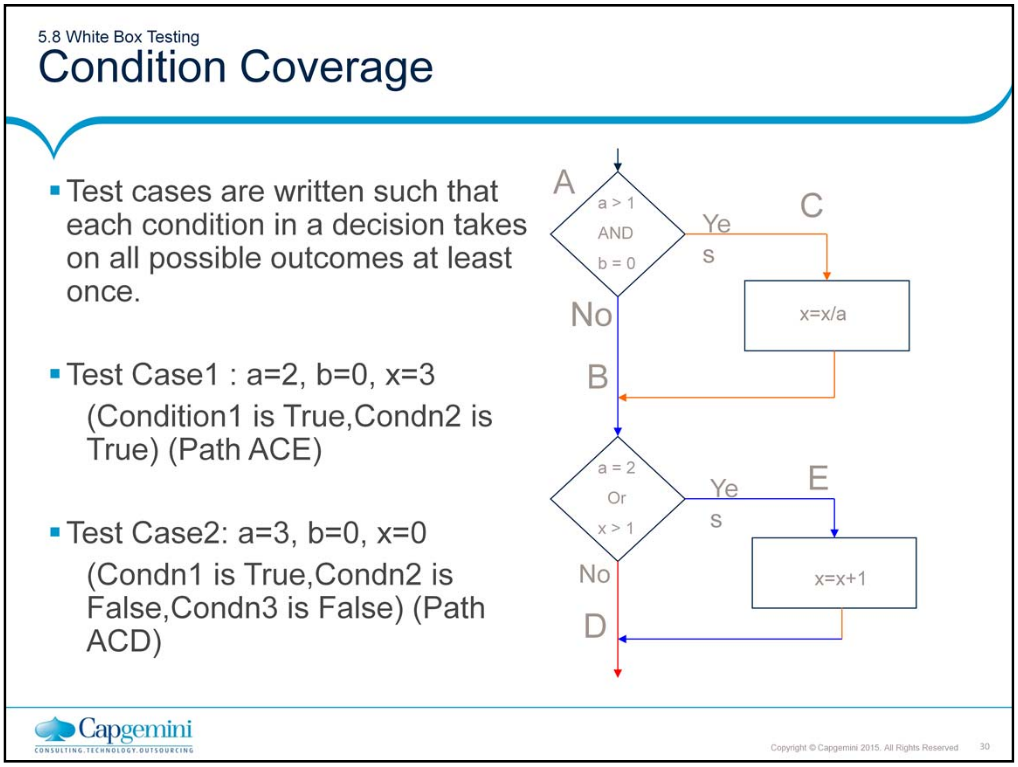
- Decision means any predicate. i.e the statement which returns true or false.
- In decision coverage test cases should be designed in such a way that each decision will be tested for true and false value.

```
BEGIN
            PRINT "Enter 2 numbers"
            READ a and b
            If (a>1) && (b=0) THEN
                        x=x/a;
            ELSE IF (a=2 || x>1) THEN
                        x=x+1;
            END IF
            PRINT x
END
```

Example: In the above example there are 2 decisions

1. a>1 and b=0
2. 2. a=2 or x>1

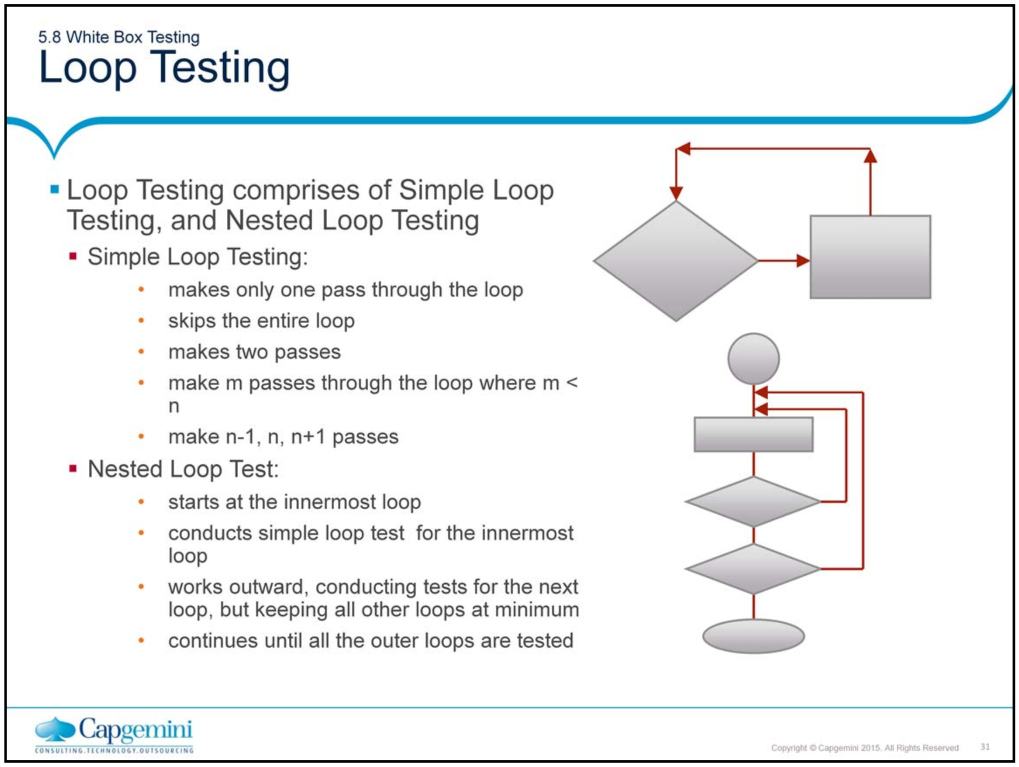
- So decision coverage can cover two test cases covering paths ACE and ABD. Even if the above test cases satisfy decision coverage it still does not cover the path ACD and path ABE. Hence decision coverage though stronger criteria than statement it is still weak. There is only 50 percent chance that we would explore the path.



Condition testing is a test case design method that exercises the logical conditions contained in a program module. A simple condition is a Boolean variable or a relational expression. Relational operator is one of the following <, <=, =, not =, > , =>.
A compound condition is composed of two or more simple conditions, Boolean operators, and parentheses.

Condition coverage focuses on testing each condition in a program. The purpose of the condition testing is to detect not only errors in the conditions of a program but also other errors in the program.

```
BEGIN
        PRINT "Enter 2 numbers"
        READ a and b
        If (a>1) && (b=0) THEN
                x=x/a;
        ELSE IF (a=2 || x>1) THEN
                x=x+1;
        END IF
        PRINT x
END
```



Guidelines for identifying test cases for white box testing

Look at each line of code, and check if a test is needed.

Look at computations – check if positive, negative, 0 test is needed.

Look at IF conditions – ensure that both sides of the conditions are tested.

Look at Loops – test for 0, 1, n, and (n+1) iterations (remember Induction!)

Look for special operations like file IO, memory / string manipulations.

5.8 White Box Testing

Path Testing

- Path Testing is a type of White Box Testing
  - It enables the test case designer to derive a logical complexity measure of a procedural design
  - It guarantees to execute every statement in the program at least once during the testing
  - The starting point for Path Testing is a “Program Flow Graph”
  - The upper bound on the number of independent paths that comprise the basis set can be calculated by the “Cyclomatic Complexity” of the “Flow Graph”

Flow graph:

The “Flow Graph” is the “main tool” for test case identification.

A “Flow Graph analysis” is concerned with statically determining the number of different paths by which the flow of control can pass through an algorithm.

It shows the relationship between “program segments”.

> A program segment is a sequence of statements. The program segment has a property that when the first member of the sequence is executed, all the other statements in that sequence get executed, as well.

Nodes represent one program segment.

> Nodes bounded by edges and nodes are called “regions”.
>
> Areas bounded by edges and nodes are called “regions”.

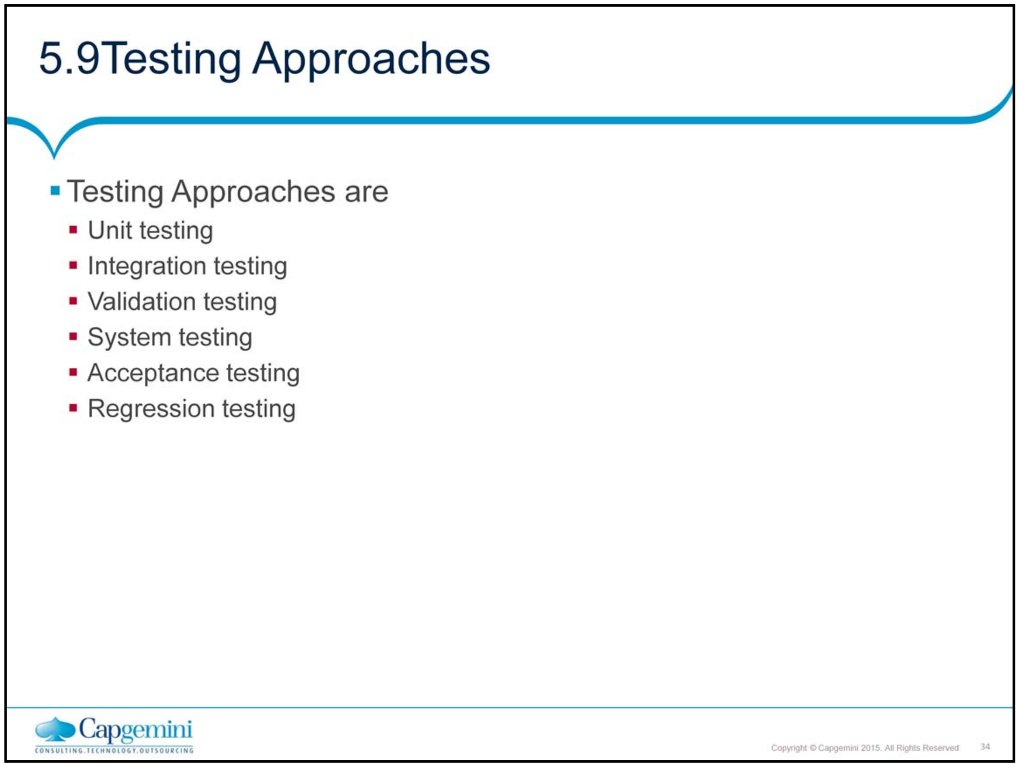
An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.

An independent path must move along at least one edge that has not been traversed before the path is defined.

5.8 White Box Testing

Path Testing

- Data Flow Testing uses the “sequence of variable access” to select points from a “control graph”
  - It is basically used to view the value produced by each and every “computation” by each and every “variable”
  - Data Definition faults are nearly as frequent as 22% (Control Flow faults are 24%)



Testing Approaches:

- Unit testing is code-based and performed primarily by developers to demonstrate that their smallest pieces of code execution works properly.

- Integration testing demonstrates that two or more units or other integrations work together properly.

- Validation Testing can be used for performing validation of software typically includes evidence that all software requirements have been implemented correctly and completely and are traceable to system requirements.

- System testing demonstrates that the system works end-to-end in a production-like environment to provide the business functions specified in the high-level design(both functional and not functional requirement)

- Acceptance testing is conducted by business owners and users to confirm that the system does, in fact, meet their business requirements.

- Regression Testing is the testing of software after a modification has been made to ensure the reliability of each software release.

5.9 Testing Approaches

Unit Testing

- Module Testing
- Done by Programmers
- Discover discrepancies between the unit's specification and its actual behavior
- Testing a form, a component or a stored procedure can be an example of unit testing

What is a Unit?

Synonyms are “component” and “module.”

The IEEE glossary says (for module):

- A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.
- A logically separable part of a program.

Unit Testing:

- The most 'micro' scale of testing to test particular functions, procedures or code modules. Also called as Module testing.
- Typically done by the programmer and not by Test Engineers, as it requires detailed knowledge of the internal program design and code.
- Purpose is to discover discrepancies between the unit's specification and its actual behavior.
- Testing a form, a class or a stored procedure can be an example of unit testing

5.9 Testing Approaches

Integration Testing

- Integration Testing focuses on testing a combination of two or more modules
- The different Integration Testing strategies are:
  - Big Bang approach
  - Incremental approach
    - Top-Down approach
    - Bottom-Up approach
    - Sandwich approach
- The terminologies related to Integration Testing
  - Stub:
    - Simulation of a subordinate module
  - Driver:
    - Simulation of a super-ordinate module

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

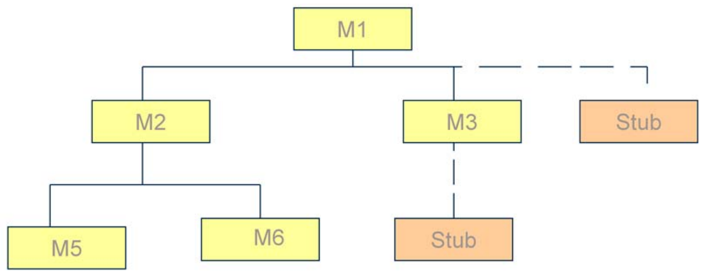
Non-incremental Testing (Big Bang Testing)

Each Module is tested independently and at the end, all modules are combined to form a application.

5.9 Integration Testing

Top Down Integration Testing

- Top Down Incremental Module Integration:
  - Topmost module is tested first. Once testing of top module is done then any one of the next level modules is added and tested. This continues till last module at lowest level is tested.



The main control module is used as a test driver. Stubs are substituted for all components directly subordinate to the main control module. Depending on the approach subordinate stubs are replaced by actual components.

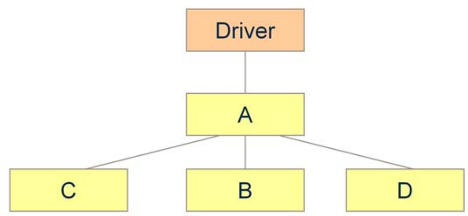
Disadvantages:
Many tests are delayed until stubs are replaced by actual modules.
Time taken to develop stubs to perform the functions of the actual modules.

Advantage :
Fast

5.9 Integration testing

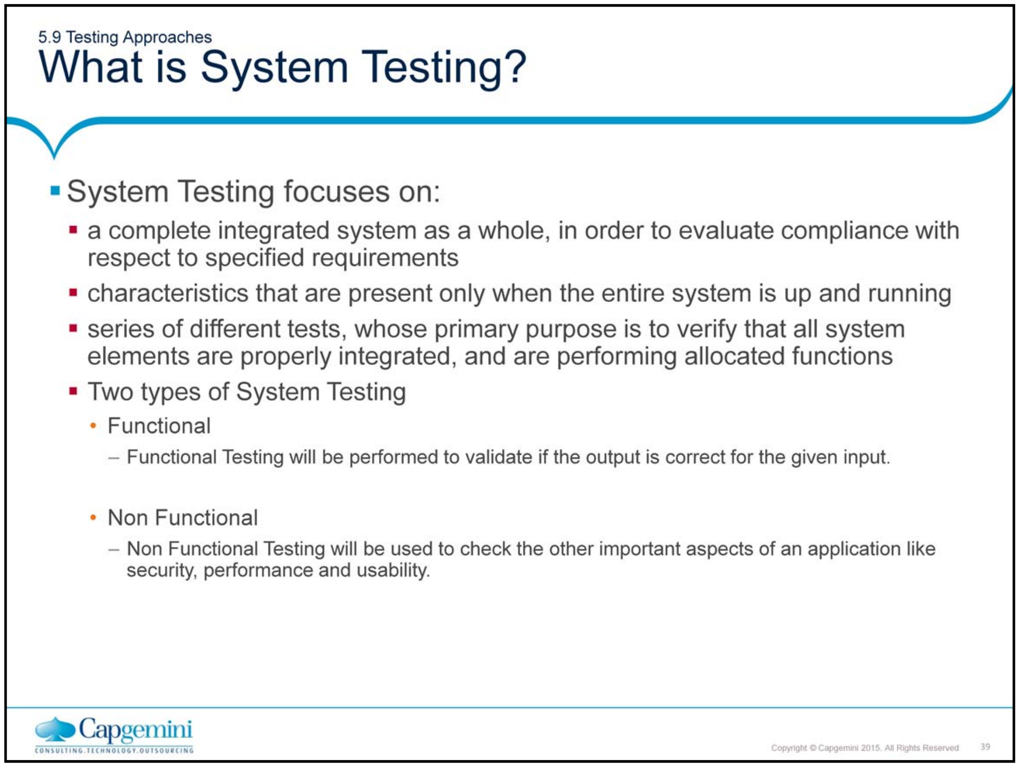
Bottom Up Integration Testing

- Bottom Up Incremental Module Integration:
  - Firstly module at the lowest level is tested first. Once testing of that module is done then any one of the next level modules is added to it and tested. This continues till top most module is added to rest all and tested



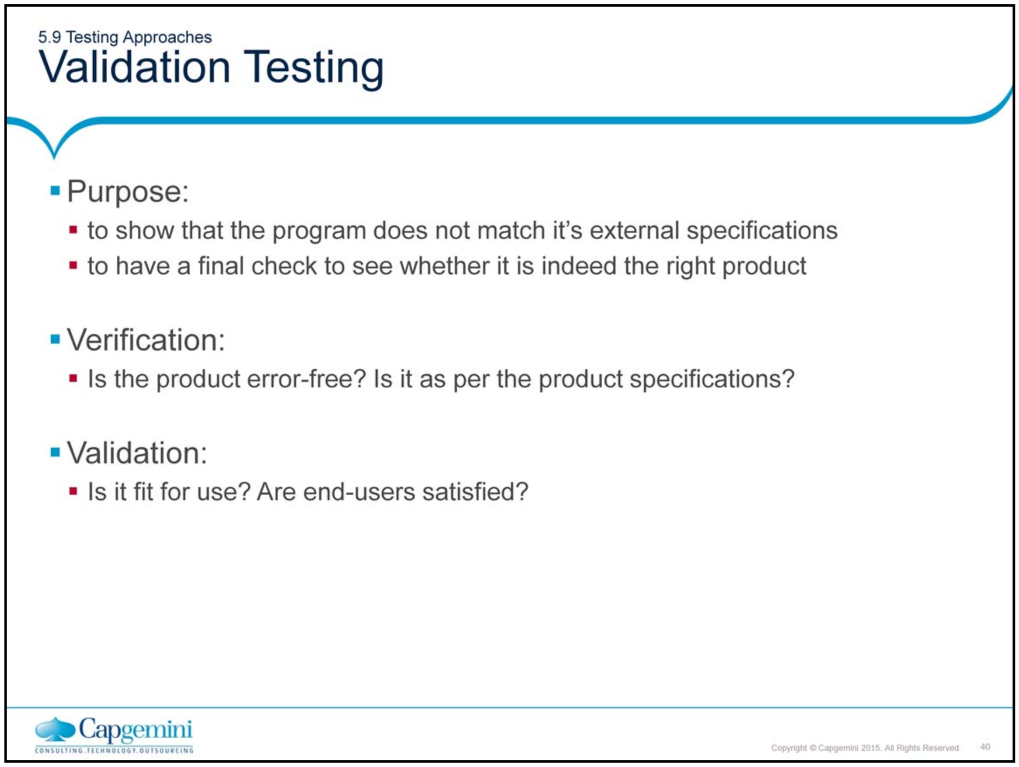
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

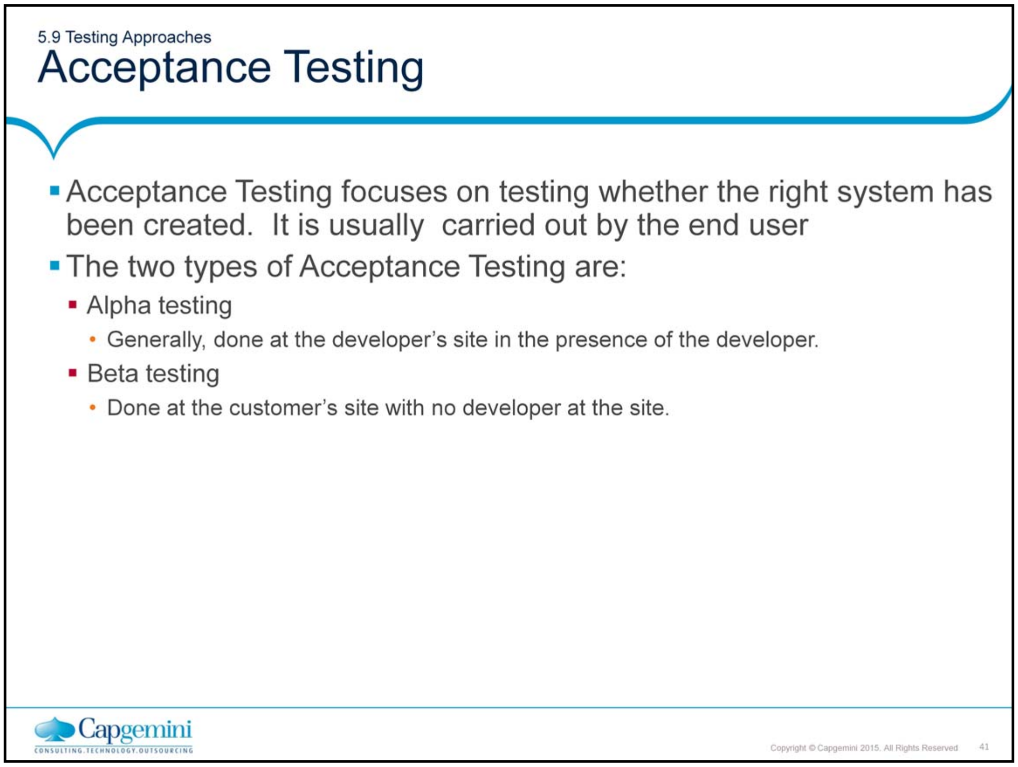
Low-level components are combined into clusters (builds) that perform a specific sub function. A driver is written to coordinate test case input and output. Drivers are removed and clusters are combined moving upward in the program structure.



Types of System Testing:

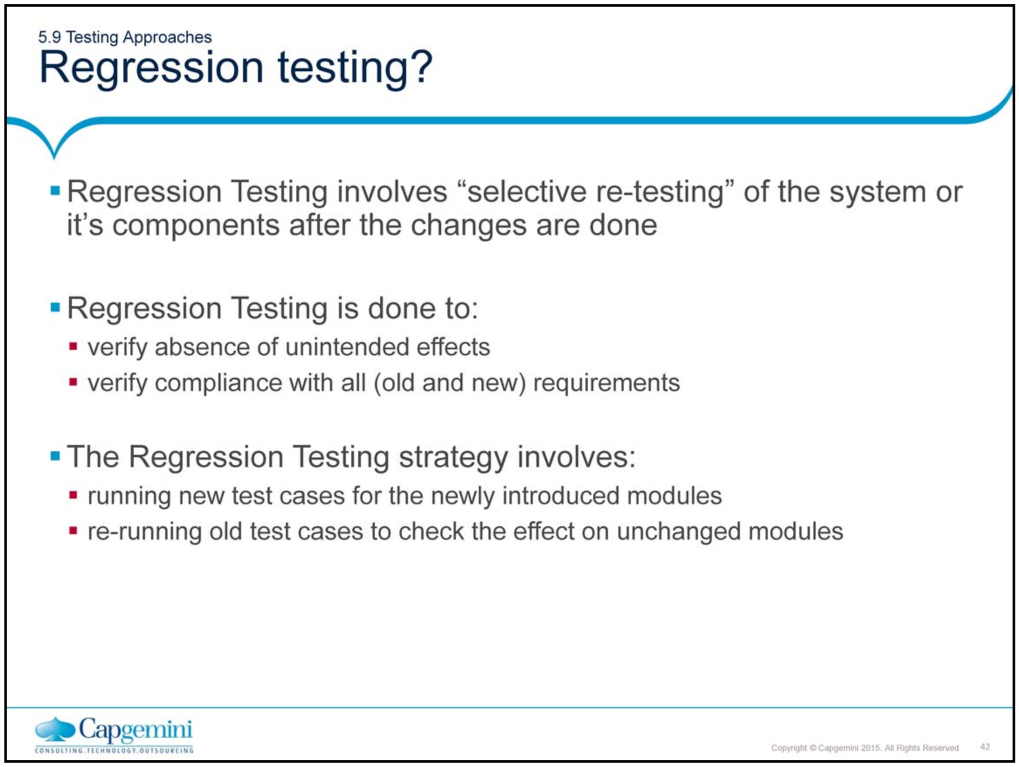
- Functional Testing will be performed to validate if the output is correct for the given input.
- Non Functional Testing will be used to check the other important aspects of an application like security, performance and usability.


5.9 Testing Approaches
Validation Testing
Purpose:
to show that the program does not match it's external specifications
to have a final check to see whether it is indeed the right product
Verification:
Is the product error-free? Is it as per the product specifications?
Validation:
Is it fit for use? Are end-users satisfied?
Copyright © Capgemini 2015. All Rights Reserved
40



Acceptance Testing:

A test executed by the end user(s) in an environment simulating the operational environment to the greatest possible extent, that should demonstrate that the developed system meets the functional and quality requirements.



Regression Testing:

- Regression Testing is the testing of software after a modification has been made to ensure the reliability of each software release.
- Testing after changes have been made to ensure that changes did not introduce any new errors into the system.
- It applies to systems in production undergoing change as well as to systems under development
- Re-execution of some subset of test that have already been conducted is required

Lesson Summary

- In this lesson, you have learnt about
  - Testing is a process to identify errors
  - A successful test case is one that brings out an error in the program
  - Exhaustive testing
  - Testing Techniques like black box and white box.
  - Testing approaches like
    - Unit Testing
    - Integration Testing
    - Verification and Validation Testing
    - System Testing
    - Acceptance Testing
    - Regression Testing

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 43

Review - Questions

Question 1: Alpha Testing is -----------

A : done at the developer's site in the presence of the end user.

B: done at the developer's site in the presence of the developer.

C: done at the end user's site in the presence of the developer.

D: done at the end user's site in the presence of the end user.

Question 2 : Which of the following are true for testing

A: Test cases should be designed for valid and expected values only

B: While testing you should check for each and every possible value

C: White Box Testing focuses on the internal structure of the software

D: The purpose of testing is to find and correct the cause of the error

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 44

Review - Questions

Question 3: In simple loop testing minimum -------- test cases need to be designed

A: 5

B:6

C:7

D: None of the above

Question 4 : In top down approach of integration testing we replace modules with -------

A: driver

B: stub

C: control module

D: All of the above

 45

Review – Match the Following

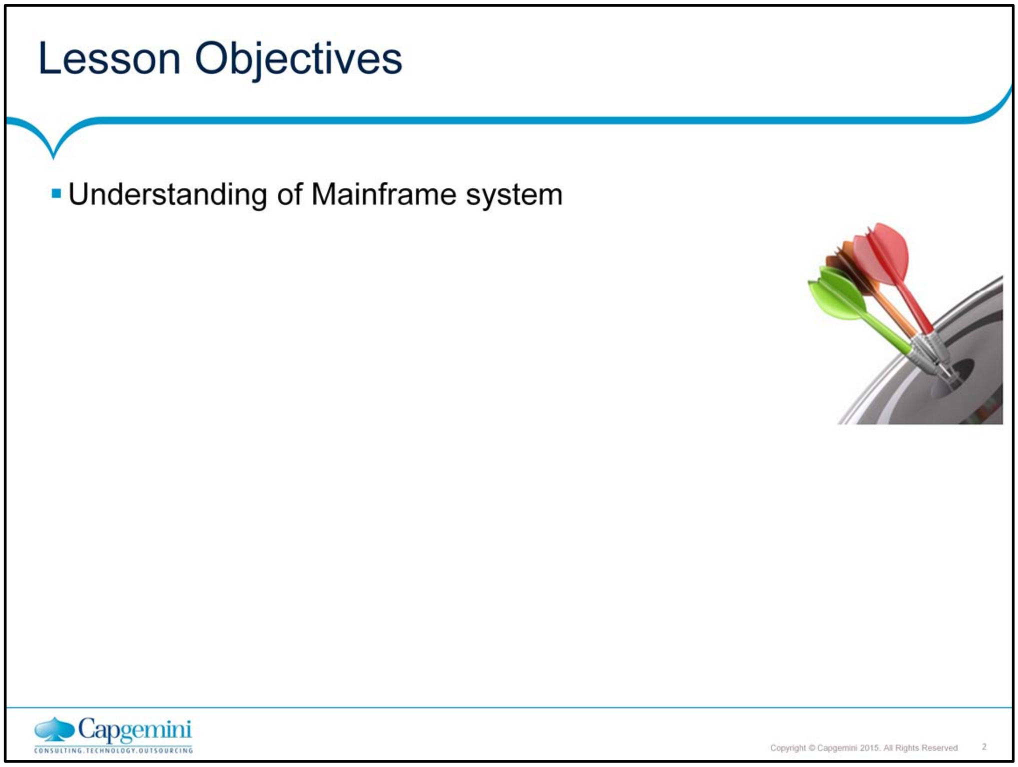
Question 5 :

1. Acceptance testing	A. Big Bang approach
2. Regression testing	B. Exhaustive testing
3. Integration testing	C. Beta testing
4. Use every possible input condition as a test case	D. Selective retesting
5. Debugging	E. Maximize bug count
	F. Storage dump

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 46

Familiarization to MF


Lesson Objectives
Understanding of Mainframe system
Copyright © Capgemini 2015. All Rights Reserved
2

Day 1

- Lesson 1: Introduction to program development with pseudocode
- Lesson 2: Good Programming Practices
- Introduction to Mainframe & ISPF region
- Demo on ISPF
  - Logon on to Mainframe terminal
  - Navigation of screens/options
  - Usage of ISPF Primary Option 3 (Utilities)
- Introduction to COBOL
  - Cobol coding format
  - Cobol program structure
- Write a pseudocode to accept the name & display them
  - Perform a self review with the help of pseudocode check list
  - Perform group review to correct the format & logic
  - Support the team to convert the pseudocode into COBOL programming

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

Day 1

- Compile the application program
  - Introduction to JCL (theory)
    - Structure of JCL statement
    - JOB Statement
    - EXEC Statement
    - DD statement
  - Steps required for compiling the application program
    - Compilation
    - Execution

Day 1

- Execute the program availble on the "DSRP042.IGATE.MF.FIRSTPRG (FIRSTPGM)
  - It would be a normal execution
- Execute the program available on the "DSRP042.IGATE.MF.FIRSTPRG (SECPGM)
  - It would be a normal execution
  - The difference is with the program. In this program group variables are used

Day 1

- Execute the program available on the "DSRP042.IGATE.MF.FIRSTPRG (THIRDPGM)
  - At the time of compilation it would throw an compilation error of **MAXCC = 8.** Debug this error & proceed for Execution.
  - At the time of execution it would throw an abend.
    - Discuss with abends & their types

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

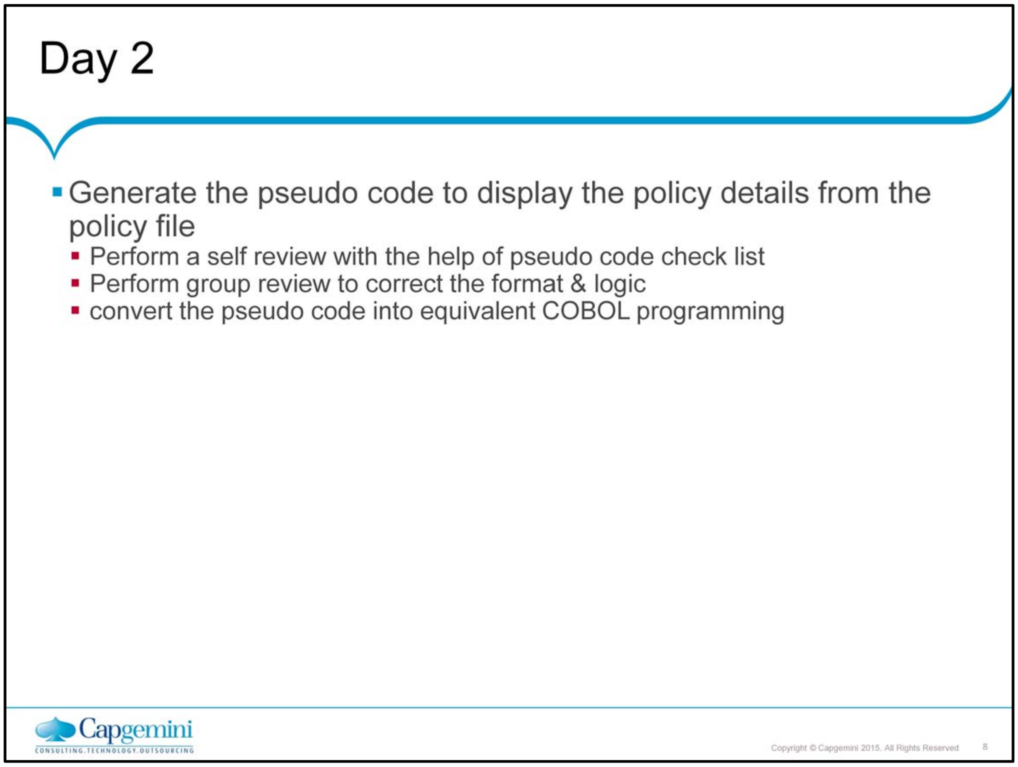
Day 2

- **Lesson 3: File Handling and Refactoring**

- Understand the insurance domain
  - Insurance Policy document
- Write a pseudo code to sort a policy file in the form of ascending order depending on policy number (use the file structure provided with the insurance domain document)
  - Perform a self review with the help of pseudo code check list
  - Perform group review to correct the format & logic
  - convert the pseudo code into equivalent COBOL programming
  - Compare the derived program with the program uploaded on the dataset "USERID.IGATE.MF.SORTING(SORTING)"
  - Compile the program using appropriate JCLs
  - Test & debug with the execution

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7


Day 2
Generate the pseudo code to display the policy details from the policy file
Perform a self review with the help of pseudo code check list
Perform group review to correct the format & logic
convert the pseudo code into equivalent COBOL programming
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
8

Day 2

- Implement brain storming to bring down COBOL Programming for the generated pseudocode (derived on day1)
  - Compare the derived program with the program uploaded on the dataset "USERID.IGATE.MF.READING(READING)"
  - Compile the program using appropriate JCLs
  - Test & debug with the execution
- Generate the Pseudocode for displaying all the new customer who have taken polices for that particular month & derive an equivalent COBOL program for the same
  - Compare the derived program with the program uploaded on the dataset "USERID.IGATE.MF.SORTING(SORTING)"
  - Compile the program using appropriate JCLs
  - Test & debug with the execution

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Day 3

- Lesson 4 : Exception Handling
- Lesson 5: Software Testing
- Generate the Pseudocode for adding new customer into the policy file & derive an equivalent COBOL program for the same
  - Compare the derived program with the program uploaded on the dataset "USERID.IGATE.MF.WRITING(SORTING)"
  - Compile the program using appropriate JCLs
  - Test & debug with the execution
- Generating a report of new customers on monthly basis.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

Day 3

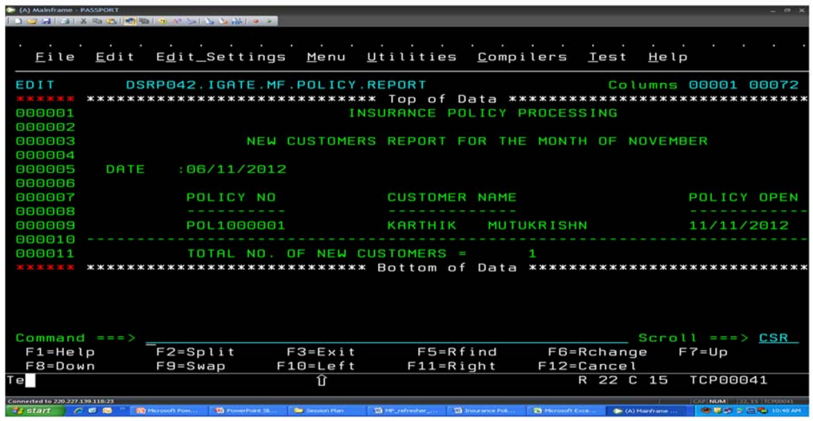
- Report1
  - Write a Pseudocode to generate a report the all the policy details of the customer, who has taken the policy for that particular month
  - Perform a self review with the code using the Checklist
  - Perform a peer to peer review with your team members
  - Derive an equivalent COBOL program for the code that you have generated

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

Day 4 : Normal execution

- Compare the derived program with the program available on "USERID.IGATE.MF.POLICY.MF.PGMS(NEWPOL)"
- Compile & execute them



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Day 4 : Compilation error

- Compile & execute the program available in "USERID.IGATE.MF.POLICY.PGMS.ERROR(NEWPOL)"
- Check for error during compilation

Day 4 – Execution error

- Compile & execute the program available in “USERID.IGATE.MF.POLICY.PGMS. FILEERR(NEWPOL)”
- When compiled, it would be normal a compilation
- During execution, it would throw an error

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

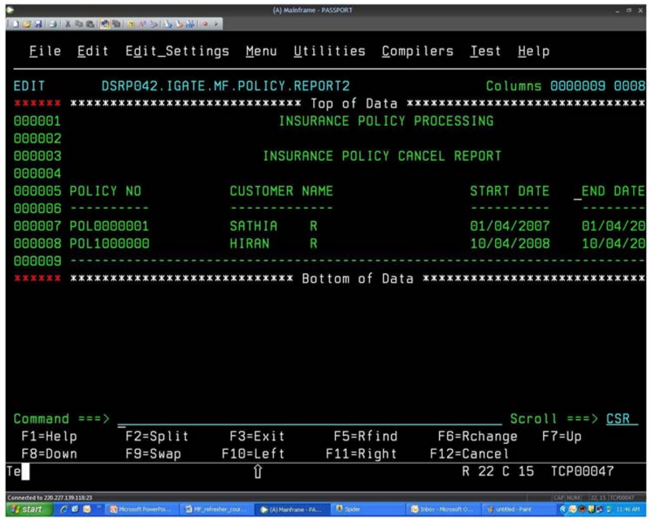
Day 4 – Report 2

- Generating a report with note of cancellation. Refer to the below link for report format
- Cancellation report
  - Write a Pseudocode to generate a report the all the policy details of the customer, who has taken the policy for that particular month
  - Perform a self review with the code using the Checklist.
  - Perform peer to peer review with your team members
  - Derive an equivalent COBOL program for the code that you have generated
  - Compare the derived program with the program available on "USERID.IGATE.MF.POLICY.REPORT2.PGMS(CANPOL)"
  - Compile & execute
  - Report generated would be as shown in next slide

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

Day 4– Report 2



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

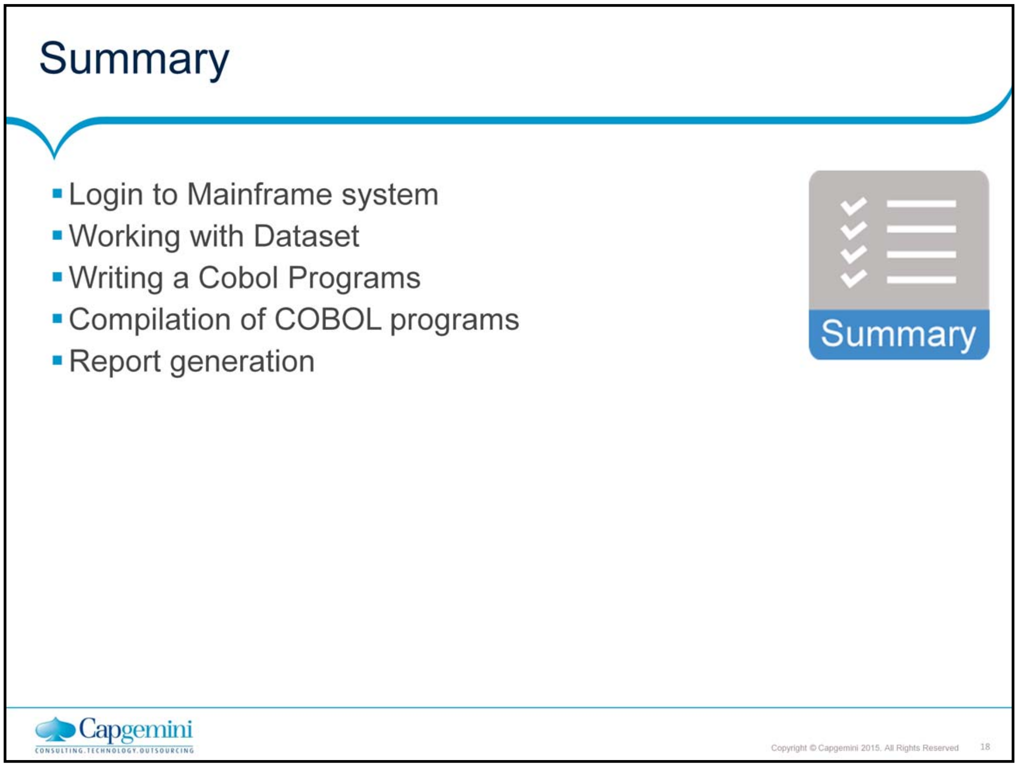
Day 4

- Modify the above program by including the footer as mentioned below

----

Reason for Cancellation: Policy has been expired

- Participants have to code the complete program to achieve the above mentioned task.
- Only an empty PDS dataset named "USERID.IGATE.MF.POLICY.REPORT3.PGMS" & "USERID.IGATE.MF.POLICY.REPORT2" would be shared


Summary
Login to Mainframe system
Working with Dataset
Writing a Cobol Programs
Compilation of COBOL programs
Report generation
Summary
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
18

Introduction to ISPF region & navigation of screens on Mainframe

- Refer to MVS Lab1
- Refer to MVS Lab4

USERID refers to your TSO login id. For eg TSO id's such as DSRP001,DSRP100,DSRP042

Introduction to COBOL

- Refer to COBOL class book Lesson 1

1. Write a Pseudo Code to get the name from the user & display them (To do)
2. Perform a self review with the code using the document shared (Checklist – Pseudo code) (To do)

Checklist - Pseudocode.xlsx

3. Perform a group review with your trainer (To do)
4. Derive an equivalent COBOL program for the code that all of us have generated, seek the help of your trainer (To do)

Execution of a COBOL program

- Refer to JCL Lab1 for executing the COBOL Program

5. Compile your first COBOL program which is stored in "USERID.IGATE.MF.FIRSTPRG(FIRSTPGM)".
   And the compiler is available at "USERID.IGATE.MF.FIRSTPRG(COBCOMP)". (To do)
6. Execute your first COBOL program which you have compiled. (To do)
7. Compile & execute your Second COBOL program which is stored in "USERID.IGATE.MF.FIRSTPRG(SECPGM)". (To do)
   a. Check if there are any errors in the program & list the difference.

8. Compile & execute your third COBOL program which is stored in "USERID.IGATE.MF.FIRSTPRG(THIRDPGM)" (To do)

Sorting Technique

- Refer to COBOL Class Book lesson 9
- Refer to the Insurance Policy Processing System document

1. Understand the insurance domain. For any quires seek the help from your facilitator
2. Write a Pseudo Code to sort a policy file in the form of ascending order depending on policy number. (To do)
3. Perform a self review with the code using the Checklist. (To do)
4. Perform a group review with your trainer (To do)
5. Derive an equivalent COBOL program for the code that all of us have generated, seek the help of your trainer (To do)
6. Compare the derived program with the program available on "USERID.IGATE.MF.SORTING(SORTING)" (To do)
7. Compile & execute the program. (To do)

Note: Unsorted records are available on "USERID.IGATE.MF.UNSRTED.FILE". And the sorted records should be loaded on "USERID.IGATE.MF.SRTED.FILE"

Reading all the records from the policy file

- Refer to the Insurance Policy Processing System document.
- Refer to COBOL Class Book lesson 8

1. Write a Pseudo Code to display the all the records from the policy file. (To do)
2. Perform a self-review with the code using the Checklist. (To do)
3. Perform a peer review with your co-team. (To do)
4. Derive an equivalent COBOL program for the code that you have generated. (To do)
5. Discuss with your facilitator with the COBOL code that you have generated.
6. Compile & execute them

Reading all the records from the policy file who have taken the policies for that particular month

- Refer to the Insurance Policy Processing System document.

1. Write a Pseudo Code to display the all the policy details of the customer, who has taken the policy for that particular month. (To do)
2. Perform a self review with the code using the Checklist. (To do)
3. Perform a peer review with your co-team. (To do)
4. Derive an equivalent COBOL program for the code that you have generated. (To do)
5. Discuss with your facilitator with the COBOL code that you have generated. Compile & execute them

Adding new records on the policy file

1. Write a Pseudo Code to add a new record into the policy file. (To do)
2. Perform a self review with the code using the Checklist. (To do)
3. Perform a peer review with your co-team. (To do)
4. Derive an equivalent COBOL program for the code that you have generated. (To do)
5. Discuss with your facilitator with the COBOL code that you have generated. Compile & execute them

1. **Generation of first report:** Generating report for new customers on monthly basis

➢ Refer to the Insurance Policy Processing System document – report1

a. Write a Pseudo Code to generate a report the all the policy details of the customer, who has taken the policy for that particular month. (To do)
b. Perform a self review with the code using the Checklist. (To do)
c. Perform a peer review with your co-team. (To do)
d. Derive an equivalent COBOL program for the code that you have generated. (To do)
e. Discuss with your facilitator or compare the derived program with the program available on “USERID.IGATE.MF.POLICY.MF.PGMS(NEWPOL)”. (To do)
f. Compile & execute them
g. Compile & execute “USERID.IGATE.MF.POLICY.PGMS.ERROR(NEWPOL)”
   When compiled & executed, identify the errors

h. Compile & execute "USERID.IGATE.MF.POLICY.PGMS.FILEERR(NEWPOL)"
When compiled & executed, identify the errors

Note: after each & every execution, view the generated report on "USERID.IGATE.MF.POLICY.REPORT"

2. **Generation of Second report:** Generating report where the status of the policy is been cancelled

- Refer to the Insurance Policy Processing System document – report2

a. Write a Pseudo Code to generate a report the all the policy details of the customer, who has taken the policy for that particular month. (To do)
b. Perform a self review with the code using the Checklist. (To do)
c. Perform a peer review with your co-team. (To do)
d. Derive an equivalent COBOL program for the code that you have generated. (To do)
e. Discuss with your facilitator or compare the derived program with the program available on "USERID.IGATE.MF.POLICY.REPORT2.PGMS(CANPOL)". (To do)
f. Compile & execute them. Refer to the report available on "USERID.IGATE.MF.POLICY.REPORT2".
g. Modify the above mentioned program, by providing the footer as

```
-----------------------------------------------------------------

Reason for Cancellation: Policy has been expired

-----------------------------------------------------------------
```

3. **Generation of Third report:** Generate report where the status of the policy is been expired.

- Refer to the Insurance Policy Processing System document – report2
- For reference, you can use the program available at "USERID.IGATE.MF.POLICY.REPORT2.PGMS(CANPOL)".

a. Write a Pseudo Code to generate a report the where the status of the policies is been expired. (To do)
b. Perform a self review with the code using the Checklist. (To do)
c. Perform a peer review with your co-team. (To do)

d. Derive an equivalent COBOL program for the code that you have generated. (To do)
e. Code your program on "USERID.IGATE.MF.POLICY.REPORT3.PGMS(EXPYPOL)". (To do)
f. Compile & execute them.
g. The final layout of the report should be generated on "USERID.IGATE.MF.POLICY.REPORT3".

Insurance Policy Processing System

Design Document

TABLE OF CONTENTS

1) Key Objectives

Developing the Insurance Policy Processing System (**IPPS**) would introduce the developer to do the following.

- Online Processing
- Batch Processing

2) Insurance Terminology

Policies - Insurance Policies can be of different types depending on what is being insured. Some samples are Auto policies, Fire Policies and Life Policies

Premium - Premium is the amount to be paid by the insured to the Insurance Company. It is derived based on the coverages chosen and other major factors of the insured. For different kind of policies different factors will have major impact on the premium calculation.
For example; Age, sex and past accidents by the driver for the Auto vehicle Insurance premium

Claims – Claims are raised by the insured on mishap and processed by the Insurance Company.

Receivables – the following transactions are posted online for billing and are called 'Receivable' in general.
NB - New Business
CH - Change Request
CN - Cancellation
IR - Insured Request Cancellation
NP - Non-Pay Cancellation
RU - Reinstatement (Taking consideration the cancelled requests)
RN - Renewal

Statements – Next act for Customers with reference to their account/policy is intimated through Statements. Some of the well known statements are

NOC - Notice of Cancellation.
OOF - Out of Force.
Bill - Statement Monthly Payable Bills. Etc...

Cash - Amount payable by the customer with reference to his/her account/policy is cash. Cash can be of different type like Agent cash and Electronic Fund Transfer cash (EFT).

3) Requirements Analysis

Online Processing:

This is the initial step which will allow the Agent/Customer to create a new policy for the vehicle or the customer can edit his existing policy. CICS would be used to create this OLTP (Online Transaction Processing) System.
Note: its only for your awareness.

Batch Processing:

In this Processing, the user has to fetch the data which is been processed by the OLTP system & generate the report. IT will have the following requirements.

1) Create a report that will have the new customers who have taken policies in the current month.
2) Create a statement report which can be sent to the customers when the policy is cancelled (Notice of Cancellation).

4) File Design

In this file assigned is a sequential file & the file description is as provided below;

COLUMN NAME	DESCRIPTION
POLICY_NUMBER	CHAR(10)
POLICY_TYPE	CHAR(10)
START_DATE	CHAR(10)
EXPIRY_DATE	CHAR(10)
AGENT_ID	CHAR(20)
RENEWAL_DATE	CHAR(10)
YEAR_OF_MANUFACTURE	NUMERIC(4)
REGISTRATION_NO	CHAR(20)
MAKE_MODEL	CHAR(20)
POLICY_PREMIUM	DECIMAL(9,2)
PREMIUM_PAID	DECIMAL(8,2)
POLICY_PERIOD	NUMERIC(2)
POLICY_STATUS	CHAR(1)
CUSTOMER_FIRST_NAME	CHAR(10)
CUSTOMER_LAST_NAME	CHAR(10)
ADDRESS_LINE	CHAR(25)
CITY	CHAR(20)
STATE	CHAR(20)
DESCRIPTION	CHAR(30)

Insurance Policy Processing System

Batch Processing

1) Design - Batch Process

1) Create a report that will have the new customers who have taken policies for the particular month

```
                 INSURANCE POLICY PROCESSING SYSTEM
                 NEW Customers Report for the month July 2012

 Date: MM/DD/YYYY

     Policy No          Customer Name          Policy Open Date
     ------------       ------------------     ---------------------



    -------------------------------------------------------------------------------
                    Total No. of New Customers = ZZZZ9
```

Description:

Create the report by unloading the data from the CUSTOMER File

Main Process:

1) The header of the report should be in the format

INSURANCE POLICY PROCESSING SYSTEM NEW Customers Report for the month July 2012

2) Initialize the Customer counter.
3) Read the file and display the records in the format

Policy No	Customer Name	Policy Open Date
------------	------------------	---------------------

4) Increment the customer counter.
5) Repeat the steps 3 and 4 until end of the file.
6) Display the Customer counter calculated in the below format.

-- Total No. of New Customers = ZZZZ9

Insurance Policy Processing System

Batch Processing

2) Design - Batch Process

2) Create a statement report which can be sent to the customers when the policy is cancelled (Notice of Cancellation).

INSURANCE POLICY PROCESSING SYSTEM
Insurance Policy Cancel report

Policy No	Customer Name	Start Date	Expiry Date
------------	------------------	----------------	----------------

----

Description:

Create the report by unloading the data from the CUSTOMER File

Main Process:

1) The header of the report should be in the format

INSURANCE POLICY PROCESSING SYSTEM
Insurance Policy Cancel report

2) Initialize the policy cancel counter.
3) Read the file and display the records in the format

Policy No	Customer Name	Start Date	Expiry Date
------------	------------------	----------------	----------------

4) Increment the policy cancel counter.
5) Repeat the steps 3 and 4 until end of the file.
6) Display the remarks of cancellation as a trailer in the below format

----
Reason for Cancellation: **
