Individual Assignment

Weightage: 80%

Introduction to Computer Science Part 1 (50 marks)

Construct a solution algorithm for the following problems. Your solution should contain:

- A defining problem
- A pseudo code algorithm
- A desk check of the algorithm

1. An Employee file contains the employee number, employee name, department number and their salary. The department number contains value 10, 20 and 30. Increment for the employee will be calculated depends on their department number. If the department number is 10, 20 and 10 salary increment will be 40%, 60% and 80% respectively. Construct an algorithm that will read a record on the file, calculate the salary increment for that record, and print the employee number, employee name, department number and new salary after increment.

2. A member records contains member name, gender (M or F), age (in years) and marital status (single or married) for each member. Design an algorithm that will read through the file and calculate the numbers of married men, single men, married women and single women. Print these numbers on member summary report. If any single men are over 30 years of age, print their names and ages on a separate eligible bachelors report.

3. Design an algorithm that will prompt for and receive the time expressed in 2400 format (e.g. 2305 hours), convert it to 12-hour format (e.g. 11.05 pm) and display the new time to the screen. Your program is to repeat the processing until a sentinel time of 9999 is entered.

Introduction to Computer Science Part 2 (50 marks)

Implement the three questions algorithm in Assignment 1 into Java programming.

Your code should contain appropriate validations and must focus on code optimization.

You need to submit:

- Three Java code
- For each question explain (100-150 words) how the logic works.
- Three sample output screenshot (Two normal and one error test case)

Kaplan 2021

KAPLAN SINGAPORE

MID timetable - FTDip2105_IT71/CF32_ICS (44458), FTDip2105_IT71/CF32_ICS (Wk 37)

	Monday 6 Sep 2021	Tuesday 7 Sep 2021	Wednesday 8 Sep 2021	Thursday 9 Sep 2021	Friday 10 Sep 2021	Saturday 11 Sep 2021	Sunday 12 Sep 2021
08:30AM 11:30AM							
11:30AM							
12:00PM 03:00PM		Lab, 12:00PM-03:00PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng	Lab, 12:00PM-03:00PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng				
03:15PM 06:15PM	Lab, 03:15PM-06:15PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng						
06:30PM 10:30PM							
10:30PM							

Kaplan 2021

KAPLAN SINGAPORE

MID timetable - FTDip2105_IT71/CF32_ICS (44458), FTDip2105_IT71/CF32_ICS (Wk 38)

	Monday 13 Sep 2021	Tuesday 14 Sep 2021	Wednesday 15 Sep 2021	Thursday 16 Sep 2021	Friday 17 Sep 2021	Saturday 18 Sep 2021	Sunday 19 Sep 2021
08:30AM 11:30AM							
11:30AM							
12:00PM 03:00PM		Lab, 12:00PM-03:00PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng					
03:15PM 06:15PM	Lab, 03:15PM-06:15PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng						
06:30PM 10:30PM							
10:30PM							

Kaplan 2021

KAPLAN SINGAPORE

MID timetable - FTDip2105_IT71/CF32_ICS (44458), FTDip2105_IT71/CF32_ICS (Wk 39)

	Monday 20 Sep 2021	Tuesday 21 Sep 2021	Wednesday 22 Sep 2021	Thursday 23 Sep 2021	Friday 24 Sep 2021	Saturday 25 Sep 2021	Sunday 26 Sep 2021
08:30AM 11:30AM							
11:30AM							
12:00PM 03:00PM		Lab, 12:00PM-03:00PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng	Lab, 12:00PM-03:00PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng				
03:15PM 06:15PM	Lab, 03:15PM-06:15PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng						
06:30PM 10:30PM							
10:30PM							

Kaplan 2021

KAPLAN SINGAPORE

MID timetable - FTDip2105_IT71/CF32_ICS (44458), FTDip2105_IT71/CF32_ICS (Wk 40)

	Monday 27 Sep 2021	Tuesday 28 Sep 2021	Wednesday 29 Sep 2021	Thursday 30 Sep 2021	Friday 1 Oct 2021	Saturday 2 Oct 2021	Sunday 3 Oct 2021
08:30AM 11:30AM							
11:30AM							
12:00PM 03:00PM		Lab, 12:00PM-03:00PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng					
03:15PM 06:15PM	Lab, 03:15PM-06:15PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng						
06:30PM 10:30PM							
10:30PM							

Kaplan 2021

KAPLAN SINGAPORE

MID timetable - FTDip2105_IT71/CF32_ICS (44458), FTDip2105_IT71/CF32_ICS (Wk 41)

	Monday 4 Oct 2021	Tuesday 5 Oct 2021	Wednesday 6 Oct 2021	Thursday 7 Oct 2021	Friday 8 Oct 2021	Saturday 9 Oct 2021	Sunday 10 Oct 2021
08:30AM 11:30AM							
11:30AM							
12:00PM 03:00PM		Lab, 12:00PM-03:00PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng					
03:15PM 06:15PM	Lab, 03:15PM-06:15PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng						
06:30PM 10:30PM							
10:30PM							

Kaplan 2021

KAPLAN SINGAPORE

MID timetable - FTDip2105_IT71/CF32_ICS (44458), FTDip2105_IT71/CF32_ICS (Wk 42)

	Monday 11 Oct 2021	Tuesday 12 Oct 2021	Wednesday 13 Oct 2021	Thursday 14 Oct 2021	Friday 15 Oct 2021	Saturday 16 Oct 2021	Sunday 17 Oct 2021
08:30AM 11:30AM							
11:30AM							
12:00PM 03:00PM		Lab, 12:00PM-03:00PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng					
03:15PM 06:15PM	Lab, 03:15PM-06:15PM FTDip2105_IT71/CF32_ICS (44458) Kang Leng						
06:30PM 10:30PM							
10:30PM							

Kaplan 2021

KAPLAN SINGAPORE

MID timetable - FTDip2105_IT71/CF32_ICS (44458), FTDip2105_IT71/CF32_ICS (Wk 43)

	Monday 18 Oct 2021	Tuesday 19 Oct 2021	Wednesday 20 Oct 2021	Thursday 21 Oct 2021	Friday 22 Oct 2021	Saturday 23 Oct 2021	Sunday 24 Oct 2021
08:30AM 11:30AM							
11:30AM							
12:00PM 03:00PM							
03:15PM 06:15PM							
06:30PM 10:30PM							
10:30PM							

Introduction of computer science

Liew You Jie

CT0357623

Department of information Technology, Kaplan

FTDip2105_IT71/CF32_ICS

Dr. Kang Leng

Assignment Due Date 15 October 2021

Dropbox link: ICS assignment to download

https://www.dropbox.com/s/bhn1iwljklo31un/ICS_Assignment_CT0357623.zip?dl=0

Introduction of computer science

Liew You Jie

CT0357623

Department of information Technology, Kaplan

FTDip2105_IT71/CF32_ICS

Dr. Kang Leng

Assignment Due Date 15 October 2021

Google Drive Link: ICS assignment to download

https://drive.google.com/file/d/1zMlyX04A0rxr3niXDWyH73DfEuecqV9U/view?usp=sharing

KAPLAN HIGHER EDUCATION ACADEMY

KAPLAN HIGHER EDUCATION INSTITUTE

Study Guide

Diploma Programme

Introduction to Computer Science

v2.0

Message to Student

"I encourage you to be an active learner, questioning and extending what you know in order to become a contributing participant in the modern global economy."

Dear Kaplan Student,

Thank you for choosing Kaplan Singapore and entrusting your educational investment with us.

It is Kaplan Singapore's and the School of Diploma Studies' mission as your 'private education institution of choice' to commit to providing a fulfilling student journey and learning experience for your development. The underlying goal of Kaplan Singapore is to ensure that our students are able to respond the demands of the globalised economy set against a highly uncertain and complex environment.

In the 21st Century it is important that your educational experience provides with you with the skills and competencies on HOW to think and not just WHAT to think.

As such, the Kaplan Diplomas, which have a long history of being recognised as equivalent to the first year or more of undergraduate studies in high-quality international universities, aim to prepare you for work, further study, and lifelong learning.

Your role as a Kaplan student is to participate, think deeply on a topic and work with other students and your Lecturers to broaden your knowledge. I encourage you to be an active learner, questioning and extending what you know in order to become a contributing participant in the modern global economy.

However, only you can embody these attributes. You are the major determinant of your own academic success and as adult learners you will receive both the freedom and responsibility this entails. With that, I wish you well in your learning and look forward to seeing you succeed.

Krishna Rajulu
Academic Dean, Kaplan Singapore

Kaplan Desired Graduate Attributes

Through the reading of this module, Kaplan Singapore intends to:

- Instill in students the value of lifelong and self-directed learning by stimulating intellectual curiosity, creative and critical thinking and an awareness of cultural diversity;
- Assist students in developing professional attributes, ethical values, social skills and strategies that will nurture success in both their professional and personal lives;
- Foster integrity, commitment, responsibility and a sense of service to the community;
- Prepare students to meet the ever-changing needs of their communities both now and in the future; and
- Promote innovative and effective teaching.

Culminating from these institutional values and educational goals, Kaplan Singapore's Desired Graduate Attributes are:

Inquiry and criticality: Graduates will be able to critically collect, evaluate and apply information and data in order to make decisions in a wide variety of professional situations. This attribute is demonstrated when students:

- Undertake, evaluate and apply appropriate research, theories, concepts and tools to investigate problems and find solutions;
- Exercise critical thinking and independent judgement to assess situations and determine solutions; and
- Have an informed respect for the principles, methods, values and boundaries of their profession and the capacity to question these.

Ethicality and discernment: Graduates will be able to assess situations and respond in an ethically, socially and professionally responsible manner. This attributed is demonstrated when students:

- Act responsibly, ethically and with integrity in their profession;
- Hold personal values and beliefs and participate in the broad discussion of these values and beliefs while respecting the views of others;
- Understand the broad local and global economic, political, social and environmental systems and their impact as appropriate to their discipline and profession; and
- Acknowledge personal responsibility for their own judgments and behaviour

Ability to communicate well: Graduates will recognise the importance and value of communication in the learning and professional environment. This attributed is demonstrated when students:

- Create and present knowledge, arguments and ideas confidently and effectively using a variety of methods and technologies;
- Recognise the wide range of possible audiences for information and respond with communication strategies appropriate to those audiences; and
- Work collaboratively with people from diverse backgrounds and be aware of the different roles of team members and to function within that team.

Independent and reflective practitioner

- Graduates will be able to work independently and be self-directed learners with the capacity and motivation for continued professional learning and development; and
- They will be able to critically reflect on their own practice and evaluate and understand current capacity and further development needs

Embedded within the desired graduate attributes are the following skills:

- Conduct research.
- Analyse, organise and present data and information.
- Think and read critically.
- Make an oral presentation.
- Intellectual curiosity and awareness of culture and diversity.
- Develop professional ethos and practice that will foster success in career and life.
- Meet the ever changing needs of communities now and in the future.

Table of Contents

Message to Student i
Kaplan Desired Graduate Attributes ii
Table of Contents iii
About this module iv
Instructions to Students v
Scheme of Work vii
Assessment Matters ix

Topic 1
Introduction to Program Design 1

Topic 2
Program Design 11

Topic 3
Introduction to Programming using Sequence and Selection control structure 18

Topic 4
Repetition Control Structures 34

Topic 5
Array processing 44

Topic 6
Introduction to Modularisation 52

Topic 7
Introduction to Object-Oriented Programming 57

Topic 8
Introduction to JAVA Programming 65

Topic 9
Introduction to selection Control 73

Topic 10
Introduction to Repetition Control 83

About this module

In this module will be covering program design in two distinct sections. In the first section we will be studying about algorithm design in the context of traditional programming languages. The section begins with a basic introduction to program design methodology and the steps in developing a solution algorithm. Then, concept by concept you will be introduced to the syntax of pseudo code; methods of defining the problem; the application of basic control structures in the development of the solution algorithm; desk-checking techniques.

In the second section will introduce the use of Java language to perform basic computation and the concept of control structures.

On completion of this module, students will develop the skills of problem solving and program design using the Java programming language.

Module Learning Outcomes

Upon successful completion of this module, the student should be able to:

- describe the steps in the program development process;
- understand about procedural and object oriented programming;
- understand the importance of control structures in programming;
- distinguish between algorithm, flow chart and pseudo code;
- apply the array data structure to programming applications;
- describe the inter-module communication that takes place within applications;

Instructions to Students

How to use this study guide

This study guide consists of written notes that form the main treatise of the subject matter of this module. You are strongly advised to **study these notes carefully and thoroughly, as well as, examine the sources that have been cited.**

Written quiz and examination will not test beyond the scope of the contents found in the study guide. However, in order to fully address the assessment requirements of the assignment, you will need to research beyond the confines of the study guide. Nevertheless, the materials herein are still a sound basis from which to build the assignment.

Further supporting materials

The study guide is supplemented by the following:

- Reproduced PowerPoint slides used by the lecturers
- Activity sheets

PowerPoint Slides

The PowerPoint slides are meant for the lecturers to signpost the flow of the lesson and for you to have a visual focus when in class. Outside of class, they can also serve to help you recall the activities that took place during the respective lessons so that you might be reminded of key learning points.

However, the PowerPoint slides must **NOT replace the need for you to read the written notes** in the study guide. The slides alone are **INSUFFICIENT for you to gain the necessary understanding of the subject matter.** As such, they **will NOT prepare you adequately** for the various summative assessment components.

Activity Sheets

It is imperative that you sincerely attempt all the activities in class and document your responses faithfully. **These activity sheets are specially designed to scaffold your learning; working through the tasks is an integral part of developing the desired skills.**

Also, by making your thinking visible through the activity sheets, it is then possible for your lecturer to provide **you with growth producing feedback** so that you may improve your performance or have your doubts clarified.

Overview of Learning Resources

Recommended Reading:

Other Suggested Reading:

Other Sources:
See Proquest and Newslink databases linked to your Elearn LMS homepage. The National Library Board on North Bridge Road (databases are for Singaporean/PR only).

Scheme of Work

SESSION		CURRICULUM
FT	PT	TOPICS
1	1	Topic 1: Introduction to Program Design • Introduction • Stages of a System Life Cycle • Steps in program development • Algorithm, Pseudo Code and Flowchart • Data Types used in Program • Data Structures used in Program
2		Topic 2: Program Design • Introduction • Six basic computer operations • Pseudo code Keywords for computer Basic operations • Flow Chart Symbols • The Structure Theorem
3	2	Topic 3: Introduction to Programming using Sequence and Selection control structure • Introduction • Program Design • Sequence control Structure • The selection control structure • Algorithms using selection
4		Topic 4: Repetition Control Structures • Introduction • Repetition using the DOWHILE structure • Counted repetition • Repetition using the REPEAT…UNTIL structure • Algorithm using repetition
5	3	Topic 5: Array processing • Introduction • Array processing • Initialising the elements of an array • Searching an array • Searching an array • Declaring a Two-Dimensional Array
6		Topic 6: Introduction to Modularisation • Introduction to Modularization • Module • Benefits of Using Modules • Top-Down Design • Hierarchy Charts • Steps in modularisation

Scheme of Work

SESSION		CURRICULUM
FT	**PT**	**TOPICS**
7	4	Topic 7: Introduction to Object-Oriented Programming • Introduction • Generations of Programming Languages • Object-oriented programming (OOP) • Introduction to Programming • Access modifiers • Programming Errors
8		Topic 8: Introduction to JAVA Programming • Introduction • Why Java? • How to write, compile, and execute our first java program. • Operators used in Java Program • Java Program
9	5	Topic 9: Introduction to selection Control • Introduction • The program development cycle • Simple Selection with null false branch (null Else statement)
10		Topic 9: Introduction to selection Control • Simple selection (simple IF statement • nested if selection structure • switch…case structure
11	6	Topic 10: Introduction to Repetition Control • Introduction • Loop Design Strategies • DOWHILE structure
12		Topic 10: Introduction to Repetition Control • Counted loop structure • REPEAT…UNTIL structure
13	7	Revision Topics 6-10 Assignment Presentation (FT)
14		

Assessment Matters

Assessment Overview

Assessment 1: Continuous Assessment (Quiz)

Weighting: 20%
Date: To be confirmed
Duration: 10 minutes per quiz
Test format: 5 MCQs per topic

Assessment 2: Individual Assignment

Weighting: 80%
Date: Session 14 for FT
Session 7 for PT
Word Limit: To be confirmed
Citation format: APA
References: Module Specific

Important Policies

Penalties for Plagiarism

Plagiarism in any form is not tolerated by Kaplan Singapore. That said, direct quotations and general similarities of common terms and language mean the E-Learn LMS will often pick up every small similarity so the likelihood of a Turnitin Similarity report recording a result of 0% is unrealistic. After all, no technology is perfect and there is the need for some direct quotation (provided you reference using APA guidelines, of course) and to use commonly accepted terms and language.

By way of quality control, Kaplan Singapore has imposed an absolute cut-off of 25% in proprietary programmes (a zero grade for the assessment). In all cases, the lecturer is the true determinant of any result. To be certain you should always reference according to the APA style required by the institution.

TOP TIP:
The surest way to succeed is to ensure all work is correctly referenced. Keep a copy of the Kaplan Singapore Academic Works and APA Guide handy when you are typing your assignments and use it to guide you as to correct referencing, citation and other aspects of academic writing.

APA GUIDE: http://bit.ly/igd-apa

Penalties for late submissions

Kaplan Singapore prepares students for the realities of the workforce and further education by requiring students to meet deadlines and submit all work on time. As such, students are required to seek approval and penalties will be imposed on late assignment submissions in accordance with the table below and cited in the Programme Handbook:

No of days late	Penalty
1 – 5 days	10% deduction per day from the marks attained by students.
After 5 days	Assignments that are submitted more than 5 days after the due date will not be accepted and it will be deemed as "No Submission". Student will be required to re-module.

Source: Kaplan School of Diploma Studies Student Handbook (V4.5, 2017), Section 9

Assignments and Kaplan Learning Management System

Kaplan Singapore School of Diploma Studies requires you to submit Assignments through the Learning Management System (E-Learn LMS). When submitted, your assignment is checked for plagiarism by software called Turnitin linked to the E-Learn LMS. The software is intended to provide one more tool to improve the quality of academic writing and as such will be compulsory for use. It is important to note that this is merely one of many tools available to you and that final decisions about the quality of your work rest with your lecturer.

Assigment Submission: How to Use E-Learn LMS for Assignment Submission

1. You will be enrolled by the School of Diploma Studies Programme Management into the E-Learn LMS system only after your fee payment is confirmed.
2. You will be sent your USER NAME and PASSWORD via email.
3. Reset your password as prompted.
4. Enter the site at the following address: https://elearn-diploma.kaplan.com.sg
5. To submit assignment please refer to the LMS Manual

Please refer to your Student Handbook for more details on Penalties for Plagiarism, Misconduct, Examinations Rules and Regulations. Should you have any queries, please contact diploma.sg@kaplan.com

This page intentionally left blank

Study Guide

Topic 1: Introduction to Program Design

Chapter Objectives

- The concept of a system
- The various stages of a system life cycle
- The activities in each stage
- To describe the steps in the program development process
- Data types and Data structures
- Differentiate between Flowchart, algorithms and pseudo code

Introduction

Systems analysis and design is a method used by organisations to create and maintain information systems that perform basic business functions such as keeping track of customer information, processing orders, and paying employees.

An Information system is a collection of interrelated components that collect, process, store, and provide as output the information needed to complete business tasks. For example, Human Resource Management (HRM) system – a system that supports such employee-related tasks as payroll, benefits, hiring, and training.

System Life Cycle is a method for developing Information System which consists of either a program or a collection of programs. Therefore, a system life cycle describes how programs are developed.

In this chapter, we shall study the phases of System Life Cycle before learning Program Design. System Life Cycle describes the stages, starting from when an idea is conceived, to the stage when the completed system is reviewed. After that we will discuss about the Program Design.

1.1 Stages of a System Life Cycle

- Planning
- Analysis
- Design
- Construction
- Implementation
- Support

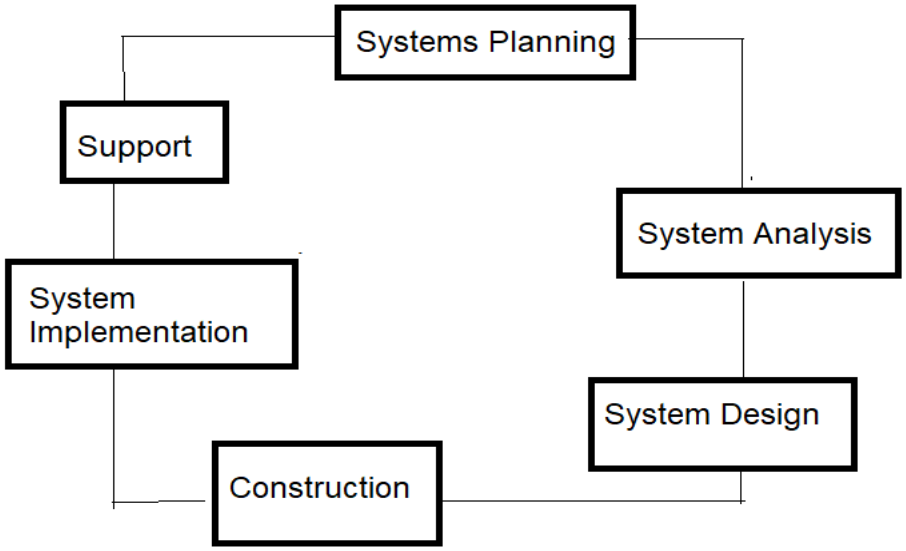


Figure 1-1: The stages of a system life cycle

1.1.1 System Planning

Any new information system must be initiated by users of the system or management. Depending on the organisation's request, the information system development team will check the background of the request to see if it is possible to provide a solution. After the background check of the existing system, the System Analyst will submit a report to management. This report is called a feasibility report. This feasibility report will describe the following:

- Define existing problem and scope
- Produce project schedule
- Confirm project feasibility
- Staff the project (resource management)
- Launch project

Detailed work will only commence if the report is approved by management.

1.1.2 System Analysis

The System Analyst gathers information to understand the problem using fact finding techniques. The fact finding techniques are interviewing, observation, questionnaires and sampling to find out the new system's requirements. Then the System Analyst will produce a report of the findings and provide recommendations to the user.

An acceptance of the system specification by the user will commence in the next stage called Design.

Requirement Analysis activities are:

- Gather information to understand problem domain
- Define system requirements
- If necessary, build prototypes for discovery of requirements
- Prioritise requirements
- Generate and evaluate alternatives
- Review recommendations with management

1.1.3 System Design

System design is the process of defining the components, modules, interfaces, and data for a system. System development is the process of creating models to develop a system.
The program specification consists of:

- Program/module description
- Input file(s) specification
- Output file(s) specification
- Processing requirement

Common methods are Pseudo Code, Flowchart and Structured Chart.

1.1.4 Construction

Once the design is approved, the development stage begins. In this stage, the models or designs are converted to programming code.
Construction activities are:

- Construct the program
- Test the program

Testing is to ensure there are no errors or problems in the program.
During Testing, the following documents are developed:

- Test log
- Test plan
- Test data
- Test results (both expected and actual)

1.1.5 System Implementation

After System Testing, installation of the developed system takes place. During the installation stage, the newly developed or modified system is handed over to the user.
Implementation stage activities are:

- Providing user training of the new system
- Data conversion for the new system
- Control procedures for the changeover

Implementation uses one of the following implementation methods:

- Direct changeover
- Parallel changeover
- Phased/Gradual changeover

1.1.6 Support

When a new system is in operation in a live environment, users start to use the system, they may require maintenance or modification due to the following reasons:

- Errors may be detected during the live usage.
- Changes in the user requirements due to a misunderstanding between the user and the System Analyst.
- Discovery of new requirements which have not been included in the requirement specification.
- External influences like changes in government policies which require modifications to the program.
- The system has been used for a sufficient length of time, maintenance may be required to improve the system
- Management may have decided that part or the whole of the system needs to be re-designed, and hence re-developed.

If redesigning needs to be done, the system planning will come in again, thus there is a need to start the entire System Life Cycle once more.

1.2 Introduction to Program Design

In this module we are going to learn Program design in detail. Programming is really all about solving problems...but we use a computer to do that for us.

Steps in program development

There are seven steps used in program development. The steps are shown below:

1. Define the problem.
2. Outline the solution.
3. Develop the outline into an algorithm.
4. Test the algorithm for correctness.
5. Code the algorithm into a specific programming language.
6. Run the program on the computer.
7. Document and maintain the program.

Step 1: Define the problem

Define the problem refers to separating the problem into three components:

- Inputs
- Processing
- Outputs

Inputs: The data that is input to the system
Outputs: What is expected from the system?
Processing: How to transform the given inputs to get desired outputs.

For e.g. you can use three numbers (3, 2, 5) with any mathematical operations to give a result of 25.
Inputs: 3, 2, 5
Outputs: 25
Processing: (3+2) * 5

Step 2: Outline the solution.

Decompose the problem in small steps. If the problem is too complicated it is better to decompose them into smaller steps. For e.g. to design a website, it is easy to break down the tasks one by one and try to find solutions individually.

Step 3: Develop the outline into an algorithm

What is an algorithm? – A set of precise steps that describes exactly the tasks to be performed and the order in which they are to be carried out. Algorithm is a solution to a problem.

Step 4: Test the algorithm for correctness

One of the important steps in the development of a program is testing. This is often ignored or forgotten. The main purpose of this stage is to identify for logic errors. We need to go through the algorithm step-by-step to make sure we get desired results.

Step 5: Code the algorithm into a specific programming language

Algorithm to Program: Once the algorithm is written the next step is to transform them into any specific programming language. If the algorithm is not correct, the program will never be correct.

Step 6: Run the program on the computer

Execute the program to check for Syntax, logic or runtime errors.

Syntax errors: Error in the rules or keywords of a specific programming language.
E.g. forget to put a semicolon at the end of the statement.
Logic errors: Error in the calculations or using incorrect operators or symbols.
E.g. using a greater than symbol (>) instead of less than (<). Logic errors will execute the program but the outputs will not be the same as expected.
Runtime errors: Errors during the execution of program.
E.g. Accessing 6th element from an array where the last index is 5.

Step 7: Document and maintain the program

Documentation is an ongoing task from the initial phase to the last stage. It involves external documentation and internal documentation.

Algorithm, Pseudo Code and Flowchart

Algorithm: Set of detailed instructions that describe the processes necessary to follow in order to produce a desired output. An algorithm is like a recipe. Follow the set of instructions step-by-step.

- Properties of an algorithm:
  - An algorithm must be very clear, precise and self-explanatory.
  - An algorithm must return correct results for all different types of inputs.
  - It doesn't matter how many times the algorithm executes but eventually it must end.

Pseudo code: Structured English language like instructions in solving a problem. They resemble high level computer language instructions. E.g. IF, ELSE etc.

A **flow chart** is a pictorial representation of the logical steps it takes to solve a problem. It uses the standard symbols in designing input, process, and output of a program. The program flow should be read from top to bottom, left to right.

Example: Program to convert the given length in feet to centimeter:

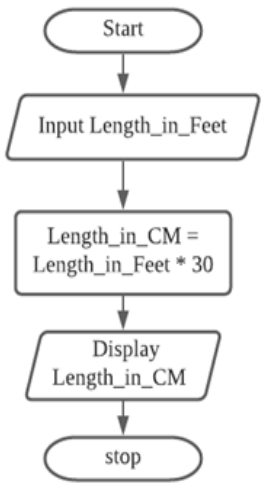
Algorithm:

- Step 1 : Input Length in feet
- Step 2 : Length in centimetre = Length in meter * 30
- Step 3: Display Length in centimeter.

Pseudocode:

1. Prompt for Length_in_Feet
2. Get Length_in_Feet
3. Length_in_CM = Length_in_Feet*30
4. Display Length_in_CM
5. END

Flowchart



1.3 Program Data

Using keyboard, we can input two types of data: Numeric and Non-numeric.

- Numeric data
  - Purely numbers are used as data. For example 5, -23, 4.5, -56.34, 0.4567, etc.
- Non-numeric
  - Non numbers such as strings of characters are used as data. For example 'William', 'S234567H' and 'test@gmail.com, etc.

Data Types used in Program

Character

Numeric Data

Divided into two main groups: Integer and Floating point number.

- Integer
  - These are whole numbers, e.g. 10, 5, 56, 123 …
- Examples of data include quantity, age of a person, number of employees, etc.
- Floating point number
  - Numbers with decimal point, example 1.23, 0.567, 23.4567….
  - Can be positive or negative.
  - Examples are height, weight, price, etc.,

Non-Numeric data

- Character
  - Character refers to single character input. e.g. 'Y' or 'N'.
  - To represent such character data type, quotes (') are needed.

- Logical / Boolean data
  - Consists of 2 values: True or False.

Identifiers

All data items used in a program need to be declared before usage. This means to state down the name given to each data item storage cell known as "identifier". We will refer to these identifiers for their content/value.

- An identifier may be numeric or non-numeric.
- The data type that can be contained in the memory location will depend on the declaration of the data.
- This memory location can be a *constant* value or *variable* value.

Constants

- It is a "Read-only" variable which contains fixed data values that cannot be changed during the program execution.
- Constant: As the name suggests, a constant is a value that remains the same throughout the execution of the program.
  e.g. pi = 3.14r.

Variables

- The value that is stored in memory which changes during the execution of a program. During programs/algorithms we always use variables to store values.
- A variable can store only one value at a time.
- e.g. num, sum, income etc.

Meaningful names

- It is essential to have unique names for variables or objects in programs.
- Names should be meaningful and must be easy to understand.
  E.g. age is better than a
- Names should be transparent to adequately describe variables (Integer1, int2, etc.).
- Programming language does not tolerate a space between variable names if multiple words are used.
- In such instances, an underscore is used (income_tax or number_of_employees).
- Another method is to use capital letters as a word separator (incomeTax or numberOfEmployees).

Summary of rules:

1. Variable names should always start with an alphabet
2. Variable names are case sensitive
3. No spaces between variable names if using multiple words
4. No special characters in variable names
5. Keywords are not supposed to be used as a variable name

1.4 Data Structures used in Program

- Data items grouped logically together to give information about an individual.
- For example, an address book is a data structure containing data items (names, addresses, phone numbers, etc.).
  - Array
  - String
  - File
  - Record

♦ Array: Collection of homogenous data items. All elements must be of the same data type.
  For e.g. 23, 35, -12, -4, 88
  For e.g. ‘a’, ‘e’, ‘i’, ‘o’, ‘u’

♦ String: is a Collection of alpha-numeric characters.
  e.g. kaplan123*

♦ File
  - A file is defined as a group of related data records handled as a logical unit.
    For example, the stock master file contains data records of all stock items in the warehouse, the payroll master file contains data records of all employees currently on the payroll of the organisation.

- Record
  - A record is made up of various data items.
  - e.g. student record contains studentID, Name, Address, Mobile
  - A record is defined as a group of related data items called fields, forming a logical unit.

    For example, the stock master record may contain data items like ITEM-CODE, DESCRIPTION, UNIT-PRICE, QUANTITY-ON-HAND and so on. These data items make up the stock master record, and they are called FIELDS.

REFERENCES

Robertson, L. (2007). Simple program design (5th ed.). Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving (7th ed.). Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology.

Topic 2: Program Design

Chapter Objectives

- Introduce keywords when writing pseudo code
- Introduce flowchart symbols used for different functions
- To define the three basic control structures as set out in the Structure Theorem
- To illustrate the three basic control structures using pseudo code and flowchart

Introduction

Program is set instruction to computer to perform certain task. That means, the system accepts input from the user as data, processes the data and displays the output. For example, customer food order details are entered into the program as input. After processing, an output in the form of a receipt format will be produced by the program.

Computers require data to perform tasks. In order to develop any program to process and produce a result, a programmer needs to know data type and structure of the data. In this chapter, we will discuss meaningful names for variables, basic computer operations and how to write pseudo code and flow chart for these operations.

2.1 Six basic computer operations

A computer performs six basic operations: it receives input, displays output, can perform various arithmetic calculations. It can assign a value to a variable or memory location. It also does various logical comparisons. If an instruction needs to be repeated many times, the computer can perform these type of instructions.

The six basic computer operations are:

1. A computer can receive information
2. A computer can put out information
3. A computer can perform arithmetic
4. A computer can assign a value to a variable or memory location
5. A computer can compare two variables and select one of two alternate actions
6. A computer can repeat a group of actions

2.2 Pseudocode

The word pseudo means fake, so pseudocode is a fake code. It is an informal language that has no syntax rules, and is not meant to be compiled or executed. Instead, programmers use pseudocode to create models, or “mock-ups” of programs. Because programmers don’t have to worry about syntax errors while writing pseudocode, they can focus all of their attention on the program’s design. Once a satisfactory design has been created with pseudocode, the pseudocode can be translated directly to an actual code.

Pseudo code keywords for computer basic operations

There are many other keywords used, but the most popularly used keywords are shown below:

Computer Basic operation 1: A computer can receive information

To receive information, the following keywords are used:
- GET
- READ

Get is used to receive input from the user or keyboard. Get will receive a value from the user and store it in a variable.

For e.g. Get num → prompts the user to enter a value for num; once entered it will be stored in num.
Read is used to read input data values from a file. Prompt and Get are also used in pseudocode to retrieve information.

Computer Basic operation 2: A computer can put out information

To output information, the following keywords are used:
- DISPLAY
- PRINT

The verbs Print, Write, Put, Output or Display are used in pseudocode when a computer is required to supply information or output to a device.
Print is used when the output is to be sent to a printer.
Put, Output or Display are used when the output is to be written to the screen.

Computer Basic operation 3: A computer can perform arithmetic

A computer performs arithmetic operations such as addition, subtraction, multiplication, division and modulus calculations. In pseudo code, the following keywords or operators are used to represent arithmetic operations.

- Add (+)
- Subtract (-)
- Product (*)
- Divide (/)
- Remainder (%)

- Orders of operation
  - BIDMAS/BODMAS
  - Brackets Index Division Multiplication Addition Subtraction

Examples of order of operation:
5 + 2 * 8 = 21
(5 + 2) * 8 = 80

Computer Basic operation 4: A computer can assign a value to a variable or memory location

To assign a value to a variable we use SET as the keyword.
Example:

```
SET Sum to zero
SET COUNT = 0
SET message to Blank
```

Computer Basic operation 5: A computer can compare two variables and select one of two alternate actions

For comparison of two variables we use IF THEN ELSE ENDIF as the keywords.

Example:

```
IF (number < 0) THEN
        Display 'Negative Number'
ELSE
        Display 'Positive Number'
ENDIF
```

Condition (number < 0) can be true or false but cannot be both. The output will be either Negative number or Positive number but not both.

Computer Basic operation 6: A computer can repeat a group of actions

For repetitions we use the following keywords:

1. DOWHILE…ENDDO
2. DO …. ENDO
3. REPEAT….UNTIL

Example:

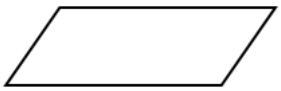
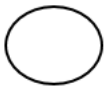
```
DOWHILE (condition)

    -   Statement block

ENDDO
```

The above block of statements will repeat as long as the condition is true. Once the condition becomes false, it will automatically stop the DOWHILE loop and execute statements (if any) after the ENDDO.

2.3 Flow Chart Symbols

- Input / Output  (parallelogram)
- Processing (rectangle)
- Terminal (racetrack)
- Connector  (circle)
- Program Flow

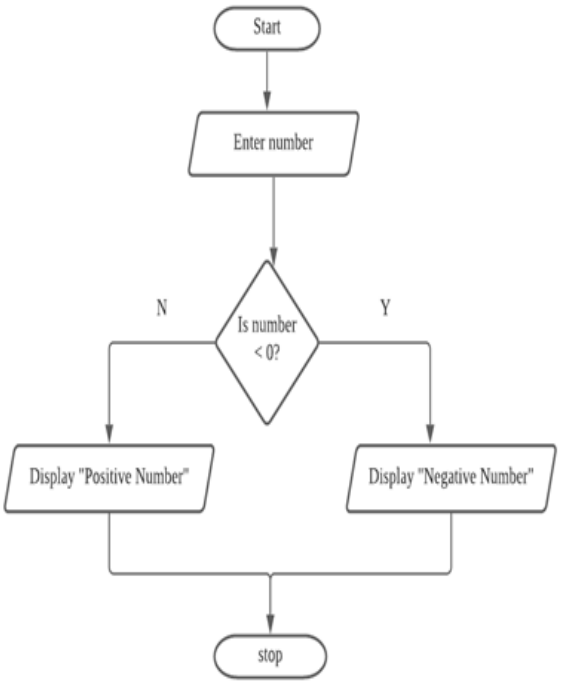
2.4 The Structure Theorem

Programs produced using only the three kinds of control structures called sequential, selection and repetition, are called **structured programs**. Structured programs are easier to test, modify, and are also easier for other programmers to understand.

1. **Sequence structure**

   - Step by step execution of the program
   - The first four steps of the computer operations represents a sequence

```
1. Prompt for number
2. Get number
3. IF (number < 0) THEN
        Display 'Negative Number'
   ELSE
        Display 'Positive Number'
   ENDIF
   END
```



2. Selection structure

- Asks a question, then takes one of two possible courses of action based on the answer. It is also called a **decision structure** or an **if-then-else.**
- The fifth step of the computer operations represents selection

- Depending on the condition, it will select one of the options to be executed.
  General pseudocode Syntax is as show below:
  IF (condition) THEN
  - Execute statements

  ELSE
  - Execute statements

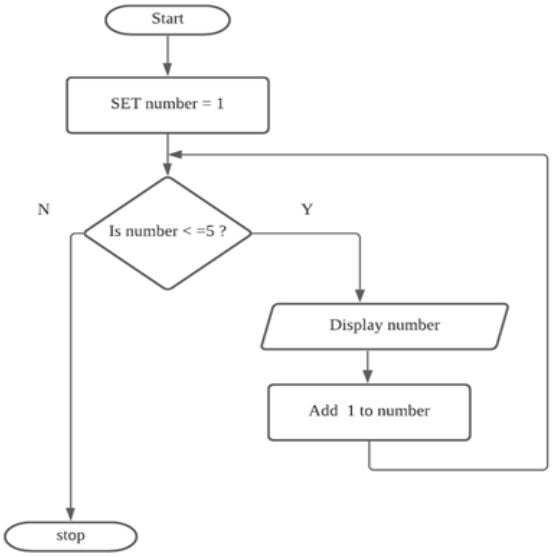
  ENDIF

  Condition can be true/false; cannot be both.

3. Repetition structure

- Repeats a set of actions based on the condition. Also called **loop** or **iteration.**
- The statement block will repeat if the condition is true. When the condition becomes false, it will stop the DOWHILE loop and come outside the ENDDO and will execute statements (if any) after the ENDDO.
- The sixth step of the computer operations represents repetition.

```
1.  Set number to 1
2.  DOWHILE (number < =5)
3.          Display number
4.          Add 1 to number
    ENDDO
    END
```



REFERENCES

Robertson, L. (2007). Simple program design (5th ed.). Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving (7th ed.). Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology, ISBN: 0-7600-

Topic 3: Introduction to Programming using Sequence and Selection Control Structure

Chapter Objectives

At the completion of this chapter, you would have learnt:

- To introduce methods of analysing a problem and developing a solution using Pseudo Code and Flowchart
- To introduce methods of manually checking the developed solution
- To develop simple algorithms using the sequence control structure
- To develop simple algorithms using the selection control structure

Introduction

Programs must be carefully designed before they are written. During the design process, programmers use tools such as pseudo code and flowcharts to create models of programs. Program Design consists of the steps a programmer should do before they start coding the program in a specific language. In this chapter, we shall look into those steps.

Flowcharts and Pseudocode are used in analysing, designing, documenting or managing a process or program in various fields. These design tools often facilitate communication between programmers and business people.

3.1 Program Design

Program design must be extremely structured, having the ultimate intentions of performing a specific calculation efficiently with attractive, understandable, efficient programs.

Three broad areas of activities in program design: (1) Understanding the Program, (2) Using Design Tools to Create a Model and (3) Develop Test Data.

Program Design is divided into three sections:

A. defining the program
B. designing a solution algorithm
C. desk checking the solution algorithm

A. Defining the program

It is the first step in program development. There is a need to identify the input required for the program, expecting output of the program and processing steps.

Any given program should be divided into three separate components:

1. Input: list of data required for the program to run
2. Processing: a list of actions needed to produce the required outputs
3. Output: list of expected outputs from the program.

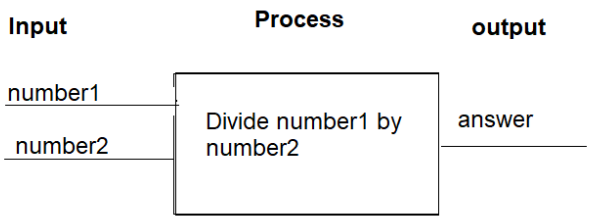
Input	Processing	Output
List all the inputs	Explain what needs to be done NOT how to be done.	List all the outputs

In some problems, the inputs, processes and output are not clearly defined. In such cases, it is best to concentrate on the outputs required.

B. Designing a solution algorithm

Designing a solution algorithm is the most challenging task in the program development. First attempt at designing a solution algorithm usually does not result in a finished product. Pseudo code is useful in the trial-and-error process where it is relatively easy to add, alter or delete instructions.

The example below shows the solution algorithm as Pseudo code and flow chart.



It is important for you to understand that the statements in this program execute in the order that they appear, from the top of the program to the bottom. This is shown in Figure 1. If you translated this pseudocode into an actual program and ran it, the first statement would execute, followed by the second statement, and followed by the third statement, fourth, fifth and sixth statement. If the sequence of 1, 2, and 3, 4 is reversed, there will be an error. This is called sequence control structure.

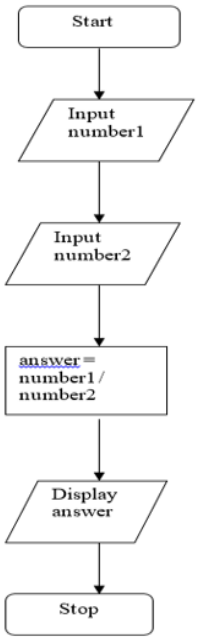
Pseudo code	Flowchart
1. Prompt for number1 2. GET number1 3. Prompt for number2 4. GET number2 5. answer = number1 / number2 6. DISPLAY answer END	

Figure 1

If you try to visualise the way this program's output would appear on the screen, you should imagine something like that shown in Figure 2. Each display statement produces a line of output.

Output:

```
Enten number 1 :
12
Enten number 2 :
3
answer = 4
```

Figure 2

C. Desk checking the solution algorithm (Testing)

Desk checking is a simple debugging process for locating hard-to-find errors in a program. Testing is necessary for finding major logic errors that occur during the development of the algorithm.

Desk checking is going through each of the program's statements one by one. As we carefully look at each statement, we record the contents of each variable on a sheet of paper after the statement executes. This process is often helpful in finding mathematical mistakes and other logic errors. Desk checking helps detect errors early and allows the user to become familiar with the way the program runs.

Steps in desk checking an algorithm:

1) Choose simple input test cases that are valid
2) Establish the expected result for each test case
3) Make a table on a piece of paper of the relevant variable names within the algorithm
4) Walk the first test case through the algorithm
5) Repeat the walk-through process using other test data
6) Check the expected result established in Step 2 matches the actual in Step 5

Sequence control structure:

Let us learn to design the program using the following example:

Example 1:

Write a program that will prompt a user to input two integer values and display their sum, difference, product and quotient.

Three broad areas of activities in program design: (A) Understanding the Program, (B) Using Design Tools to Create a Model and (B) Develop Testing.

A. Defining the problem (understanding the program):

Any given program should be divided into three separate components as shown below:

Input	Processing	Output
- value1 - value 2	- prompt for value1 and value2 - Get value1 and value2 - Calculate sum, difference, product and quotient - Display sum, difference, product and quotient	- sum - difference - product - quotient

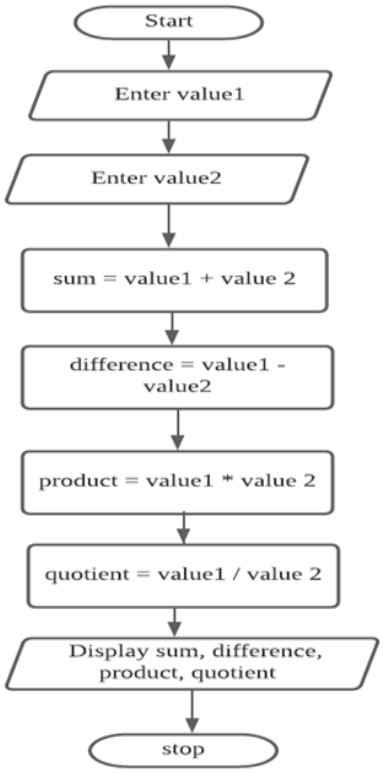
B. Using Design Tools to Create a Model

Solution Algorithm (Pseudo code)

Display_Results

```
1  prompt for value1, value 2
2  Get value1, value 2
3  sum = value1 + value2
4  difference = value1 - value2
5  product = value1 * value2
6  quotient = value1 / value2
7  Display sum, difference, product, quotient
   END
```

Flowchart



The solution algorithm shows the steps of the statements in this program execute in the order that they appear, from the top of the program to the bottom.

C. Desk checking (Testing)

We need to choose simple test data based on the requirement of the program specification. To desk check the algorithm, only a few simple test cases are required to follow the major parts of the algorithm logic.

(i) **Input data:** You can choose any two sets of input data for the checking of the algorithm.

	First data set	Second data set
value1	10	-20
value2	5	7

(ii) Expected result : You can write your expected results from the program as shown below:

	First data set	Second data set
sum	15	-13
difference	5	-27
product	50	-140
quotient	2	-2

(iii) Desk check table

Statement No.	value1	value2	sum	difference	product	quotient
First Pass						
1, 2	10	5				
3			15			
4				5		
5					50	
6						2
7			display	display	display	display
Second pass						
1,2	-20	7				
3			-13			
4				-27		
5					-140	
6						-2
7			display	display	display	display

Create a table on a piece of paper with the relevant variable names used in the algorithm. Walk through the algorithm steps from 1 to 7 using the first test data set. This is called First Pass. Repeat the walk-through process using second test data set. This is called Second Pass.

Check the expected result matches with the actual result in the desk check table. If the expected result and actual result are same, then the program is correct. Then the design part is completed.

3.2 The selection control structure

The condition in the IF statement is based on a comparison of two items and it is expressed with the following relational operations:

< less than
> greater than
= equals to
<= less than or equal to
>= greater than or equal to
<> or != not equal to

There are a number of variations of the selection structure as follows:

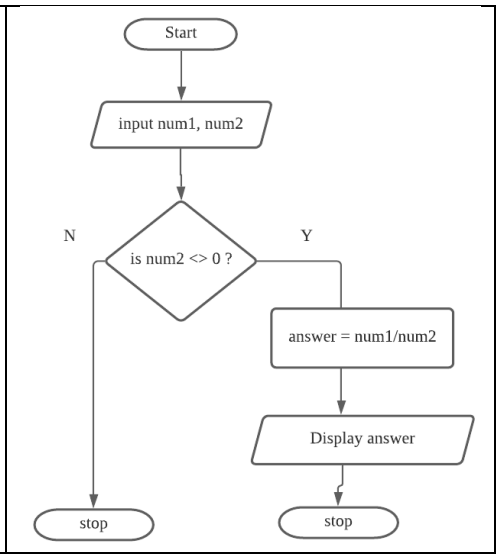
1. Simple selection with null false branch (null ELSE statement)
2. Simple selection (simple IF statement)
3. Combined selection (combined IF statement)
4. Nested selection (nested IF statement)

1. Simple selection with null false branch (Null Else Statement)

In simple selection structure, a specific action is performed only if a certain condition exists. If the condition does not exist, the action is not performed. The flowchart shown below shows how the logic of an everyday decision can be diagrammed as a decision structure. The diamond symbol represents a true/false condition.

If the condition is true, we follow one path, which leads to an action being performed. If the condition is false, we follow another path, which skips the action.

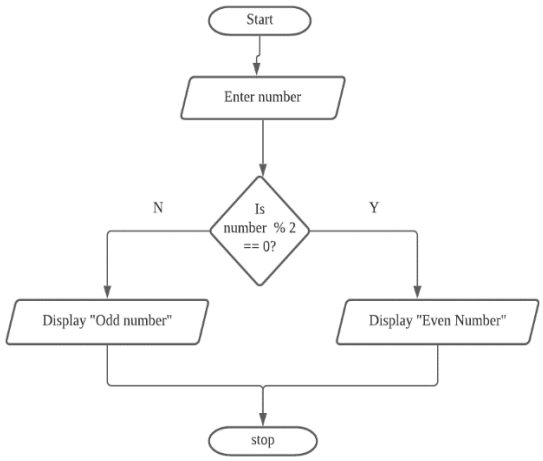
```
1. Prompt for num1, num2
2. Get num1, num2
3. IF number <> 2 THEN
       answer = num1/num2
   ENDIF
4. Display answer
   END
```



2. Simple selection (simple IF statement)

Simple selection occurs when a choice is made between two alternate paths depending on the result of a condition being true or false.
Keywords: IF, THEN, ELSE, ENDIF

```
1. Prompt for number
2. Get number
3. IF(num %2 == 0) THEN
        Display "Even number"
   ELSE
        Display "ODD number"
   ENDIF
END
```



A decision structure testing the condition number modulus 2 equals to zero. If this is true, the action displaying even number will be performed. If the condition is false, the action displaying odd number will be performed.

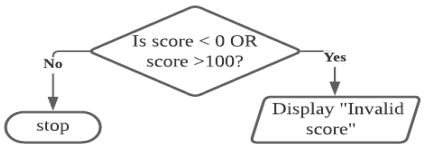
3. Combined selection (combined IF statement)

- When you have multiple conditions to be checked we use AND/OR operator.
- Truth table for AND & OR:

Condition 1	Condition 2	AND	OR
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

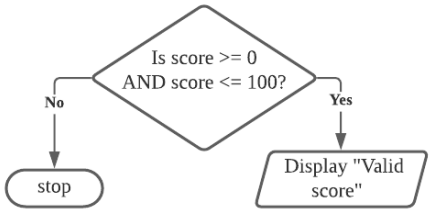
The OR operator takes two Boolean expressions as operands and creates a compound Boolean expression that is true when either of the sub expressions is true. The following is an example of an If-Then statement that uses the OR operator:

```
IF score < 0 OR score> 100 THEN
        Display "Invalid score"
End If
```



The display statement will execute only if score is less than 0 OR score is greater than 100. If either expression is true, the compound expression is true.

The AND operator takes two Boolean expressions as operands and creates a compound Boolean expression that is true when both of the sub expressions are true. The following is an example of an If-Then statement that uses the AND.

IF score >= 0 AND score <= 100 THEN
  Display “Valid Score”
ENDIF
	

The display statement will execute only if score is 0 to 100. If both expression are true, the compound expression is true.

4. Nested selection (nested IF statement)

Nested selection can be classified as Linear nested IF and Non-linear nested IF statements.

Linear nested IF statement is used when a field is being tested for various values and a different action is to be taken for each value.

```
IF record_code = ‘A’ THEN
        increment counter_A
    ELSE
        IF record_code = ‘B’ THEN
            increment counter_B
        ELSE
            increment error_counter
        ENDIF
    ENDIF
```

The above example is called linear due to each ELSE immediately following the IF condition to which it corresponds.

Non-linear nested IF occurs when a number of different conditions need to be satisfied before a particular action can occur. It is termed non-linear because the ELSE statement may be separated from the IF statement with which it is paired.

```
IF student_attendance = part_time THEN
        IF student_gender = female THEN
                IF student_age >21 THEN
                        add 1 to mature_female_pt_students
                ELSE
                        add 1 to young_female_pt_students
                ENDIF
        ELSE
                add 1 to male_pt_students
        ENDIF
ELSE
        add 1 to full_time_students
ENDIF
```

3.3. Algorithms using selection

Example 1:

Design an algorithm that prompts users to the weight of a parcel and calculate the shipping cost for that parcel and display the shipping charge. Charges are calculated as follows:

Parcel Weight (kg)	**Shipping charge per kg ($)**
<3.5 kg	$4.50 per kg
3.5 – 8 kg	$3.30 per kg
>8 kg	$2.75 per kg

A Defining diagram

Input	**Processing**	**Output**
parcel_weight	Prompt for parcel_weight Get parcel_weight Calculate shipping_charge Display shipping_charge	shipping_charge

B Solution algorithm

Note that the shipping charge for each parcel is determined as a certain cost, *per kilogram* of parcel weight.

```
    Calculate_shipping_charge
1       Prompt for parcel_weight
2       Get parcel_weight
3       IF parcel_weight < 3.5 THEN
          shipping_charge = 4.50 * parcel_weight
        ELSE
          IF parcel_weight >= 3.5 AND parcel_weight <= 8 THEN
            shipping_charge = 3.30 * parcel_weight
          ELSE
            shipping_charge = 2.75 * parcel_weight
          ENDIF
        ENDIF
4       Display shipping_charge
    END
```

C Desk checking

i Input data:

	First data set	**Second data set**
parcel_weight	2	4

ii Expected results:

	First data set	**Second data set**
shipping_charge	9.00	13.20

iii Desk check table:

Statement number	**parcel_weight**	**shipping_charge**
First pass		
1, 2	2	
3		9.00
4		display
Second pass		
1, 2	4	
3		13.20
4		display

Example 2:

Design an algorithm that will prompt users to input two numbers and an integer code. The value of the integer code should be 1, 2, 3 or 4. If the value of the code is 1, compute the sum of the two numbers. If the code is 2, compute the difference (first minus second). If the code is 3, compute the product of the two numbers. If the code is 4, compute the quotient (first divided by second). If the code is not equal to 1, 2, 3 or 4, display an error message. The program is then to display the two numbers, the integer code and computed result to the screen.

A. Defining diagram

Input	**Processing**	**Output**
Num1 Num2 Code	- Prompt for Num1, num2, Code - Get Num1, Num2, Code - Calculate Result - Display Num1, Num2, Code, Result	Num1 Num2 Code Result msg

B. Solution Algorithm (pseudocode)

Display_Result

1. Prompt for Num1, Num2, Code
2. Get Num1, Num2, Code
3. SET Result = 0, msg = blank
4. CASEOF(Code)

    1 : Result = Num1 + Num2

    2 : Result = Num1 - Num2

    3 : Result = Num1 * Num2

    4 : Result = Num1/ Num2

    Other : msg = 'Invalid Code'

    ENDCASE

5. IF (msg == blank) THEN

        Display Num1, Num2, Code, Result

    ELSE

        Display msg

    ENDIF

    END

C. Desk checking

(i) Input data

	First Data Set	Second Data Set	Third Data Set
Num1	45	-10	-5
Num2	20	24	-4
Code	1	2	7

(ii) Expected Result

	First Data Set	Second Data Set	Third Data Set
Num1	45	-10	
Num2	20	24	
Code	1	2	
Result	65	-34	
msg			Invalid code

(iii) Desk check table

Statement No	Num1	Num2	Code	Result	msg
First Pass					
1 , 2	45	20	1		
3				0	blank
4				65	
5	display	display	display	display	
Second Pass					
1, 2	-10	24	2		
3				0	blank
4				-34	
5	display	display	display	display	
Third Pass					
1, 2	-5	-4	7		
3				0	blank

4					Invalid Code
5					display

REFERENCES

Robertson, L. (2007). Simple program design (5th ed.). Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving (7th ed.). Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology, ISBN: 0-7600-

Topic 4: Repetition Control Structures

Chapter Objectives

At the completion of this chapter, you would have learnt:

- To develop algorithms that use the DOWHILE and REPEAT…UNTIL control structures
- To introduce a pseudo code structure for counted repetition loops
- To develop algorithms using variations of the repetition construct

Introduction

The iteration or repetition control structure allows certain part of the program to repeat until certain condition(s) is/are met. This is also known as looping construct. In this chapter, we will look at two broad categories of loops: condition-controlled and count-controlled. A *condition-controlled loop* uses a true/false condition to control the number of times that it repeats. A *count-controlled loop* repeats a specific number of times.

4.1 Repetition using the DOWHILE structure

Three different ways that a set of instructions can be repeated:

- Condition tested at the beginning of the loop (leading decision loop)
- Condition tested the end of the loop (trailing decision loop)
- A counted number of times (counted loop)

1. Leading decision loop

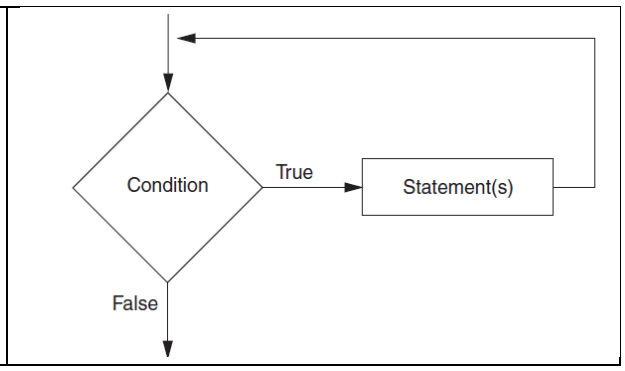
- DOWHILE construct is a leading decision loop – In this loop structure, the condition is tested before any statements are executed.

Format:

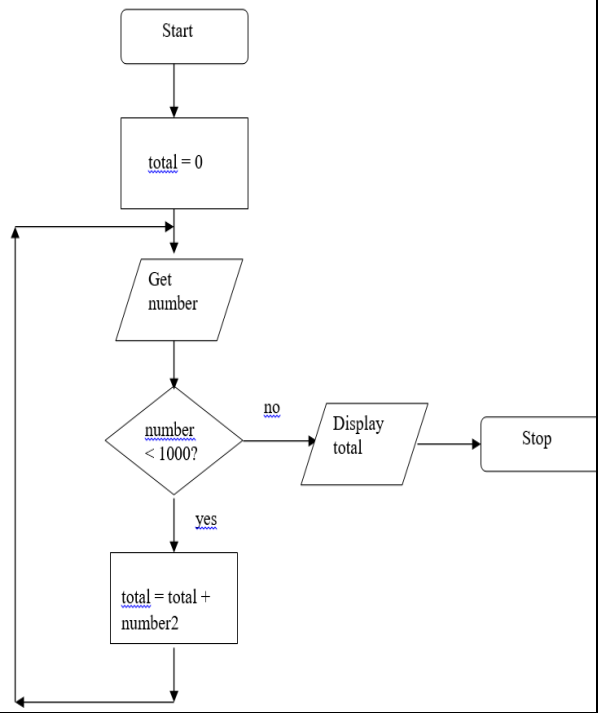
```
DOWHILE condition  is true

    statement block

ENDDO
```



Example 1:

Pseudo code	Flowchart
1. SET total = 0 2. Prompt for number 3. GET number 4. DOWHILE number < 1000 5. total = total + number 6. Prompt for number 7. GET number ENDDO 8. DISPLAY total END	

The condition stated above is whether number less than 1000. If condition is true, i.e. number is less than 1000, then number is accumulated as the total. After which, you get another number. The next execution is going back to the while condition to check if the condition is true, if yes, you accumulate the current number with the previous numbers and get another number.

The loop or set of instructions will end when number is >= 1000, whereby the accumulated total will be displayed.

- **Using DOWHILE to repeat a set of instructions a known number of times**.
  When a set of instruction is repeated a specific number of times, a counter can be used in pseudocode, which is initialised before the DOWHILE statement and incremented just before the ENDDO statement

- **Using DOWHILE to repeat a set of instruction an unknown number of times**.
  (i) When a trailer record or sentinel exists

  Special record or value placed at the end of valid data to signify the end of that data

  (ii) When a trailer record or sentinel does not exist

  End-of-file (EOF) marker is added when the file is created as the last character in the file

4.2 Counted repetition

Counted loop iterates a specific number of times. The keyword DO is used for counter loop. The way that a count-controlled loop works is simple: the loop keeps a count of the number of times that it iterates, and when the count reaches a specified amount, the loop stops.

```
DO loop_index = initial value to final_value

        Statement block

ENDDO
```

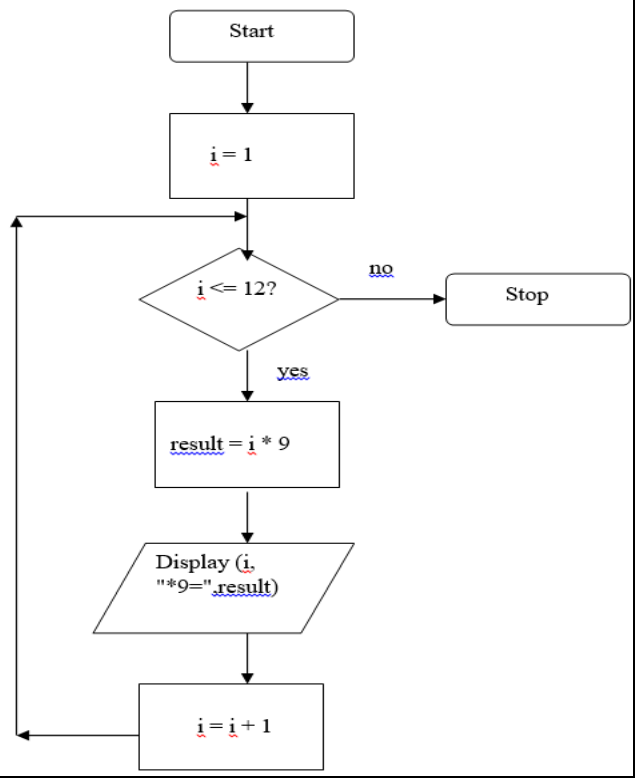
The DO loop typically performs the following three actions: *initialization, test,* and *increment:*

1. **Initialization:** Before the loop begins, the counter variable is initialized to a starting value. The starting value that is used will depend on the situation.

2. **Test:** The loop tests the counter variable by comparing it to a maximum value. If the counter variable is less than or equal to the maximum value, the loop iterates. If the counter is greater than the maximum value, the program exits the loop.

3. **Increment:** To *increment* a variable means to increase its value. During each iteration,
In other words, a counted repetition construct will perform the initialising, incrementing and testing of the loop counter automatically. It will also terminate the loop once the require number of repetition has been executed.

Example

Pseudo code	Flowchart
1. DO i = 1 TO 12 2. result = i * 9 3. Display (i, "*9=",result) ENDDO	

4.3 Repetition using the REPEAT…UNTIL structure

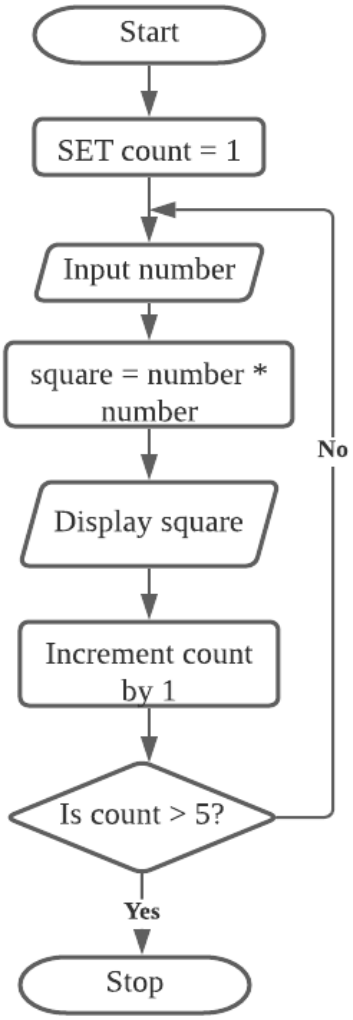
Trailing decision loop

REPEAT…UNTIL structure is similar to the DOWHILE structure (group of statements are repeated in accordance with specific condition). REPEAT…UNTIL structure is called trailing decision loop because it tests condition at the end of the loop.

In this loop the statement within the loop will be executed once before the condition is tested. In this loop, the condition is false, the statement will be repeated. If the condition becomes true, the loop will be terminated. This structure will always be executed at least once.

```
1.  SET count =1
2.  REPEAT
3.          Prompt for number
4.          GET number
5.          Square = number * number
6.          Display square
7.          count = count + 1
8.  UNTIL (count >5)
     ENDDO

    END
```



In this example, count value is set to 1. When condition (count >5) becomes true, the loop will terminate. When condition (count is less than or equal to 5) is false, the loop will be repeated.

4.4. Algorithm using repetition

Example : using DO…WHILE loop

Design an algorithm that will output the four times table, as follows:

4 x 1 = 4
4 x 2 = 4
4 x 3 = 12 . . .

4x 10 =40

Example 1: using DO…WHILE loop

A. Defining Diagram

Input	Processing	Output
x	- Calculate product - Display four times table	Four times table

B. Solution Algorithm (Pseudocode)

Display_FourTimesTable (DOWHILE loop)

```
1. SET x =1
2. DOWHILE (x <=10)
3.       Product = 4 * x
4.       Display  '4 * x = product '
5.       x = x + 1
   ENDDO
   END
```

C. Desk Checking

(i) Input data

	First Data Set
x	1, 2, 3,…..

I. Expected Result

First Data Set
4 * 1 =4 4 * 2 =8 7 * 3 =12….

II. Desk check table

Statement No	x	product	DOWHILE	Output
First Pass				
1	1			
2			True	
3		4		
4				4* 1 = 4
5	2			
2			True	
3		8		
4				4 * 2 = 8
5	3			
2			True	
3		12		
4				4 * 3 =12
5	4			

Example 2: using REPEAT… loop

Note: Repeat….until loop repeats when condition is false and stops when condition is true

B. Solution Algorithm (Pseudocode)

Display_FourTimesTable (REPEAT … UNTIL loop)

```
1.  SET x =1
    REPEAT
2.      Product = 4 * x
3.      Display  '4 * x = product '
4.      x = x + 1
5.  UNTIL (x > 10)
    END
```

Statement No	x	product	condition	Output
First Pass				
1	1			
2		4		
3				4* 1 = 4
4	2			
5			False	
2		8		
3				4 * 2 = 8
4	3			
5			False	
2		12		
3				4 * 3 =12
4	4			
…	…	…	…	….
4	10			
5			False	
2		40		
3				4 * 10 = 40
4	11			
5			True	

Example 3: using DO … ENDDO loop

B. Solution Algorithm (Pseudocode)

Display_FourTimesTable (DO … ENDDO loop)

```
1. DO x = 1 to 10
2.        Product = 4 * x
3.        Display  '4 * x = product '
   ENDDO
   END
```

Statement No	x	product	Output
First Pass			
1	1		
2		4	
3			4 * 1 = 4
1	2		
2		8	
3			4 * 2 = 8
1	3		
2		12	
3			4 * 3 =12
1	4		
…	…	…	….
1	10		
2		40	
3			4 * 10 = 40

REFERENCES

Robertson, L. (2007). Simple program design. Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving. Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology.

Topic 5: Array Processing

Chapter Objectives

At the completion of this chapter, you would have learnt:

- To introduce arrays and the uses of arrays
- To develop pseudocode algorithms for common operations on arrays
- To illustrate the manipulation of single- and two-dimensional arrays

Introduction

An array allows you to store a group of items of the same data type together in memory. Processing many items in an array is usually easier. Arrays are one of the most powerful programming tools available. This chapter covers an array data structure that is made up of several variables all of which have the same data type. Processing a large number of items in an array is usually easier than processing a large number of items stored in separate variables.

5.1 Array processing

Variables are not well-suited for storing and processing lists of data. Each variable is a separate item that must be declared and individually processed. Fortunately, most programming languages allow you to create arrays, which are specifically designed for storing and processing lists of data. Like a variable, an array is a named storage location in memory. Unlike a variable, an array can hold a group of values. All of the values in an array must be the same data type. You can have an array of Integers, an array of characters, or an array of Strings, but you cannot store a mixture of data types in an array.

The following example shows how we will declare an array in pseudocode:

```
Student_scores [10]
```

The individual data items that make up the array are referred to as the element of the array. Elements in the array are distinguished from one another by the use of an index or subscript, enclosed in parentheses, following the array name – for example: ‘scores (3)’

The most typical operations performed on arrays are:

- Loading a set of initial values into the element of an array
- Processing the elements of an array
- Searching an array, using a linear or binary search, for a particular element
- Writing out the contents of an array to a report

The elements of an array are processed in sequence, starting with the first element. This can be accomplished easily in pseudocode with either a DO loop or a DOWHILE loop.

The following example shows the simple manipulation of an array using DO and DOWHHILE loop.

In this example, each of the sum of all the elements stored in the array. When all elements have been added, the variable sum is printed.

DO …. ENDO loop	DOWHILE…..END loop
Find_sum_of_elements 1. Set sum to zero 2. DO index = 1 to number_of_elements 3. sum = sum + array (index) ENDDO 4. Print sum END	1. Set sum to zero 2. Set index to 1 3. DOWHILE (index < = 5) 4. sum = sum + array (index) 5. index = index + 1 ENDDO END

5.2 Initialising the elements of an array

Because an array is an internal data structure, initial values must be placed into the array before any information can be retrieved from it. These initial values can be assigned to the elements of the array as constants, or they can be read into the array from a file.

```
Read_values_into_variable_array

1. Set  max_num_elements to required value
2. Set index to zero
3. Read first input value
4. DOWHILE (input value NOT = 9999) AND (index <  max_num_elements)
5.     index = index + 1
6.     array (index) = input value
7.     Read next input value
8. ENDDO
9. IF index < max_num_elements THEN
        index = index + 1

        array (index) = 9999

   ELSE

        Print 'Array size too small'

   ENDIF
   END
```

Paired arrays

- Two arrays that have the same number of elements. The arrays are paired because the elements in the first array correspond to the elements in the same position in the second array.

5.3 Searching an array

An array can be searched using either a linear search or a binary search. The sequential search algorithm is a simple technique for finding an item in an array. It steps through the array, beginning at the first element, and compares each element to the item being searched for. The search stops when the item is found or the end of the array is reached.

Programs commonly need to search for data that is stored in an array. Various techniques known as search algorithms have been developed to locate a specific item in a larger collection of data, such as an array. The sequential search algorithm uses a loop to sequentially step through an array, starting with the first element. It compares each element with the value being searched for and stops when the value is found or the end of the array is encountered. If the value being searched for is not in the array, the algorithm unsuccessfully searches to the end of the array.

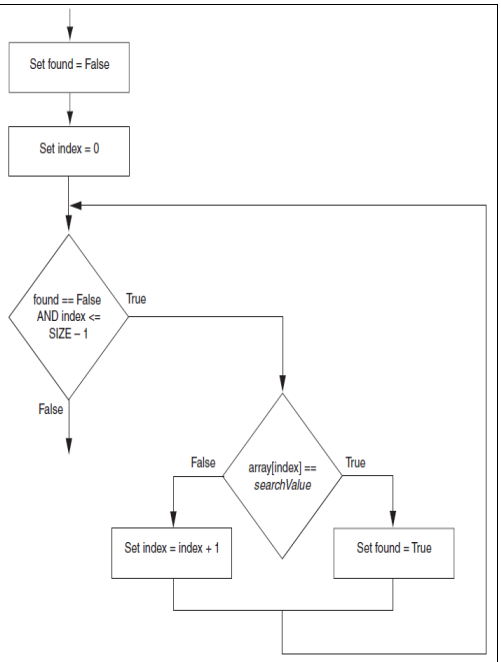
The flowchart below shows the general logic of the sequential search algorithm. Here is a summary of the data items in the figure:

- Array is the array being searched.
- *searchValue* is the value that the algorithm is searching for.
- Found is a Boolean variable that is used as a flag. Setting found to False indicates that *searchValue* has not been found. Setting found to True indicates that *searchValue* has been found.
- index is an Integer variable used as a loop counter.

When the algorithm finishes, the found variable will be set to true if the *searchValue* was found in the array. When this is the case, the index variable will be set to the subscript of the element containing the *searchValue.* If the *searchValue* was not found in the array, found will be set to False.

The following shows how you can express this logic in pseudo code:

```
SET found = False
SET index = 0
DOWHILE found == False AND index <= SIZE – 1
   IF array[index] == searchValue THEN
         SET found = True
   ELSE
         SET index = index + 1
  ENDIF
ENDDO
```

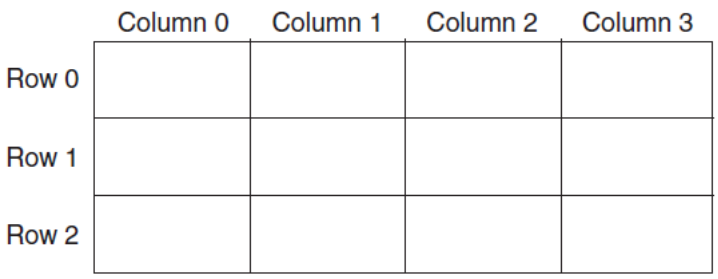


Example of linear search array:

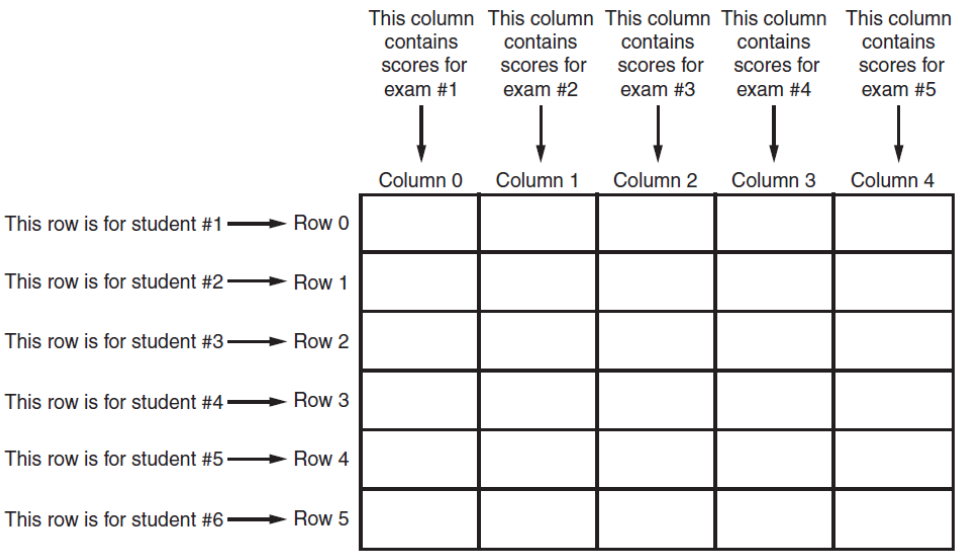
```
Linear_search_of_an_array
1.  Set max_num_elements to required value
2.  Set element_found to false
3.  DOWHILE (NOT element_found) AND (index <= max_num_elements)
4.  IF array (index) = input_value THEN
            Set element_found to true
    ELSE
            index = index +1
    ENDIF
    ENDDO
5.  IF element_found THEN
            Print array (index)
    ELSE
            Print 'value not found', input_value
    ENDIF
END
```

5.4 Two-dimensional arrays

An element of a two-dimensional array is specified using the name of the array, followed by two subscripts, enclosed in parentheses, separated by a comma. The row subscript is specified first, followed by the column subscript.



Two-dimensional arrays are useful for working with multiple sets of data. For example, suppose you are designing a grade-averaging program for a teacher. The teacher has six students, and each student takes five exams during the semester. One approach would be to create six one-dimensional arrays, one for each student. Each of these arrays would have five elements, one for each exam score. This approach would be cumbersome, however, because you would have to separately process each of the arrays. A better approach would be to use a two-dimensional array with six rows (one for each student) and five columns (one for each exam score), as shown in the figure below.



Declaring a Two-Dimensional Array

To declare a two-dimensional array, two size declarators are required:

The first one is for the number of rows and the second one is for the number of columns.

The following pseudocode shows an example of how to declare a two-dimensional array:

```
Integer_values[3][4]
```

This statement declares a two-dimensional Integer array with three rows and four columns. The name of the array is values, and there are a total of twelve elements in the array. As with one-dimensional arrays, it is best to use named constants as the size declarators.

Here is an example:

```
ROWS = 3
COLS = 4
Integer_values[ROWS][COLS]
```

When processing the data in a two-dimensional array, each element has two subscripts: one for its row and another for its column. In the values array, the elements in row 0 are referenced as follows:

```
Values [0][0]
Values [0][1]
Values [0][2]
Values [0][3]
```

The elements in row 1 are as follows:

```
Values [1][0]
Values [1][1]
Values [1][2]
Values [1][3]
```

And the elements in row 2 are as follows:

```
Values [2][0]
Values [2][1]
Values [2][2]
Values [2][3]
```

The figure below illustrates the array with the subscripts shown for each element.

	Column 0	Column 1	Column 2	Column 3
Row 0	values[0][0]	values[0][1]	values[0][2]	values[0][3]
Row 1	values[1][0]	values[1][1]	values[1][2]	values[1][3]
Row 2	values[2][0]	values[2][1]	values[2][2]	values[2][3]

Programs that process two-dimensional arrays commonly do so with nested loops.

REFERENCES

Robertson, L. (2007). Simple program design. Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving. Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology.

Topic 6: Introduction to Modularisation

Chapter Objectives

At the completion of this chapter, you would have learnt:

- To introduce modularisation as a means of dividing a problem into subtasks
- To present hierarchy charts as a pictorial representation of modular program structure
- To develop programming examples that use a simple modularised structure

Introduction to Modularisation

Modularisation is defined as dividing a problem into separate units called modules. Each unit perform a single task. Top-down design methodology allows the programmer to focus on the overall design of the problem without getting too involved with the details of the lower-level modules. The concept of modules makes the algorithm simpler, more systematic and more likely to be error free.

Module

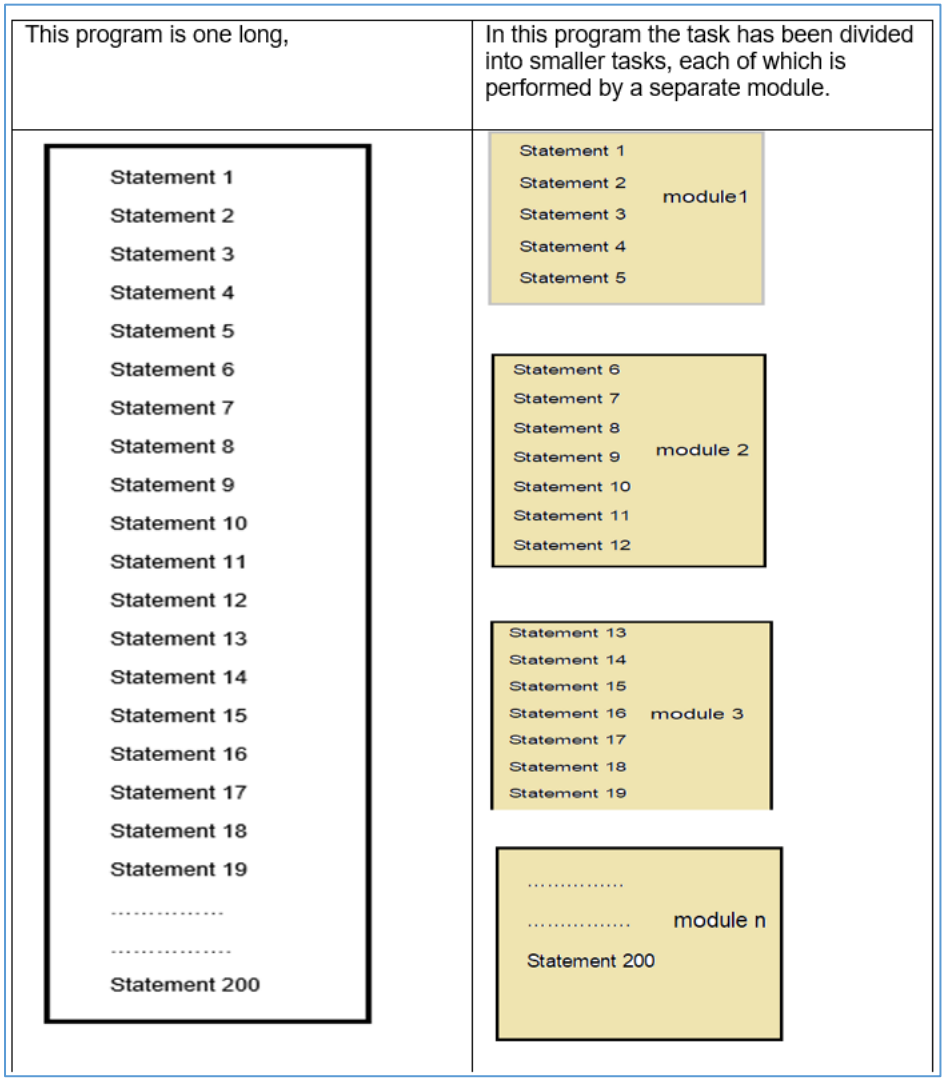
A module is a single unit consisting of a group of statements that exists within a program for performing a specific task.

Let us consider a payroll application. This application's overall task is calculating an employee's pay that would consist of several subtasks as shown below:

- Getting the employee's hourly rate
- Getting the number of hours worked
- Calculating the gross pay
- Calculating overtime pay
- Calculating taxes and benefits
- Calculating the net pay
- Printing the pay check

Most applications consist of several tasks to perform. Programmers usually break down their programs into modules so that it is easy to understand and build the application. A *module* consists of a group of statements that exists within a program for the purpose of processing a specific task. Instead of writing a large program as one long sequence of statements, it can be divided as several small modules, each module performing a specific part of the task. These small modules can then be executed in the desired order to perform the overall task.

This approach is sometimes called *divide and conquer* because a large task is divided into many smaller tasks that are easily performed. The figure below shows this by comparing two programs: one that uses a long large program as one long sequence of statements, to perform a task, and another one that divides the long sequence of statements into smaller tasks, each of which are performed by a separate module.



When using modules in a program, we generally separate each task within the program in its own module. In payroll application for example, a pay calculating program might have the modules such as pay rate, worked hours, calculate the gross pay, overtime pay, calculating taxes, calculating net pay and printing the pay check.

Although many modern programming language allow us to create modules, they are not often referred to as modules. Modules are generally known as *procedures, subroutines, subprograms, methods,* and *functions*.

Benefits of Using Modules

Simpler Code

Modularisation makes the programming codes simpler and easier to understand. Breaking the larger program into many small modules makes it easier to read than one long sequence of codes.

Code Reuse

Modularisation reduce the duplication of a code within a program. If a certain operation is performed in many places in a program, a module can be written once to perform that operation, and then be called any time it is needed. This advantage of using modules is known as *code reuse* because we are writing the code once and then reusing the module again and again whenever we need to perform the task.

Better Testing

Modularisation makes the testing and debugging of the code easier. Modularisation allows programmers to test each module in a program individually when it is developed. Module testing can be done at an earlier stage of the development in order to determine whether the module correctly performs its operation. Modularisation makes it easier to isolate each module and fix errors.

Faster Development

Suppose a programmer or a team of programmers is developing multiple programs. They discover that each of the programs performs several common tasks, such as asking for a username and a password, displaying the current time, and so on. It doesn't make sense to write the code for these tasks multiple times. Instead, modules can be written for the commonly needed tasks, and those modules can be incorporated into each program that needs them.

Easier Facilitation of Teamwork

Modularisation allows programmers to work in teams. When a program is developed as a set of modules that each performs an individual task, then different programmers can be assigned the job of writing different modules.

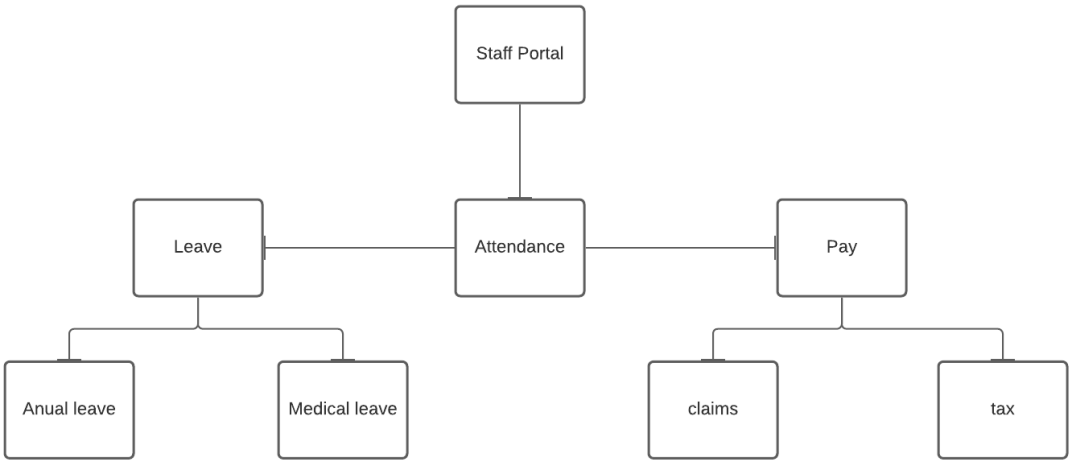
Top-Down Design

Developers commonly use *top-down design* to break down a problem into several modules. The process of top-down design is performed in the following manner:

- The main task of the program is to perform is broken down into a series of subtasks.
- Each of the subtasks is checked to determine whether it can be further broken down into more subtasks. This step is repeated until no more break down of subtasks.
- Once all of the subtasks have been identified, a code must be written for each subtask.
- This process is called top-down design because the programmer begins by looking at the top level of tasks that must be performed, and then breaks down those tasks into lower levels of subtasks.

Hierarchy Charts

Flowcharts are good tools for graphically representing the flow of logic inside a module, but flowcharts do not give a visual representation of the relationships between modules. For this reason, programmers commonly use *hierarchy charts*. A hierarchy chart, which is also known as a *structure chart*, uses rectangular boxes to represent each module in a program. The rectangles are connected in a way that illustrates their relationship to one another.
The figure below shows an example of a hierarchy chart of staff portal.



Staff portal is a cloud portal where the employees have access to their own attendance, leave details, pay slip, tax and their claims. Employees can clock in daily attendance, apply leave, apply payroll claims, and view their payslip and tax.

Steps in modularisation

1. Define the problem by dividing into its three components (input, output and processing)
2. Group the activities in subtasks or functions to determine the modules that will make up the program
3. Construct a hierarchy chart to illustrate the modules and their relationship to each other
4. Establish the logic of the mainline of the algorithm in pseudocode
5. Develop the pseudocode for each successive module in the hierarchy chart
6. Desk check the solution algorithm

REFERENCES

Robertson, L. (2007). Simple program design. Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving. Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology.

Herbert Schildt, Java: The Complete Reference,11th Edition, McGraw-Hill Education.

Topic 7: Introduction to Object-Oriented Programming

Chapter Objectives

- To introduce object-oriented design
- To define objects, classes, attributes, operations and information hiding
- To list the steps required to create an object-oriented design solution to a problem
- Discuss the evolution of programming languages
- Discuss the fundamental concepts of object-oriented programming
- Tell the difference between syntax errors, run-time errors, and logic errors

Introduction

There are two methods of programming used namely procedural and object oriented. Procedural programming is the earliest programming language, meaning a program was made of one or more procedures. A procedure is simply a module or function that performs a specific task using input from the user, performing calculations, reading and writing files, displaying output, and so on.

Object-oriented programming (OOP) focuses on an application's data and the methods needed to manipulate that data. Methods in object-oriented programs continue to use sequence, selection, and looping structures and make use of arrays. However, object-oriented programming adds several new concepts to programming and involves a different way of thinking. In this chapter we are going to learn *object-oriented programming* (*OOP*) which is focusing on creating objects. An *object* is a software entity that contains both data and procedures.

7.1 Generations of Programming Languages

- **Generation 1—machine languages:** Program data entered directly into RAM in the form of 1s and 0s
  - Using switches and, later, punch cards
  - Error prone, tedious, and slow

- **Generation 2—assembly languages:** Mnemonic symbols represent instructions and data.
  - One-to-one correspondence with machine-language instructions
    - **Assembler:** Translates to machine language
    - L**oader:** Loads machine language into memory

- **Generation 3—high-level languages:** Designed to be easy to write, read, and manipulate.
  - C, C++, Java
  - High-level instructions may represent many machine-language instructions.
  - **Compiler** translates high-level language into machine language.
- Two major groups of high-level languages:
  - **Procedural languages:** COBOL, FORTRAN, BASIC, C, Pascal
  - **Object-oriented languages:** Smalltalk, C++, Java

7.2 Object-oriented Programming (OOP)

Five important features of object-oriented languages are:

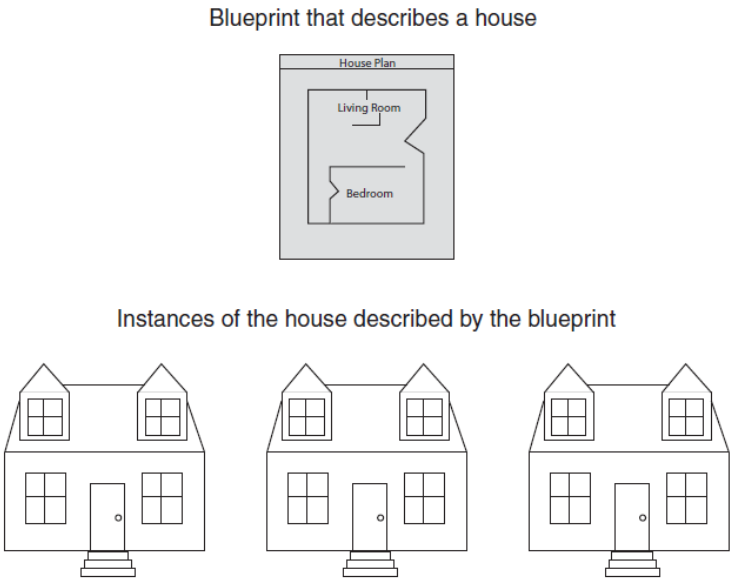
- Classes
- Objects
- Polymorphism
- Inheritance
- Encapsulation

7.2.1 Classes and Objects

In Objected Oriented Programming, a **class** is a term that describes a group or collection of objects with common attributes. An **object** is one **instance** of a class.

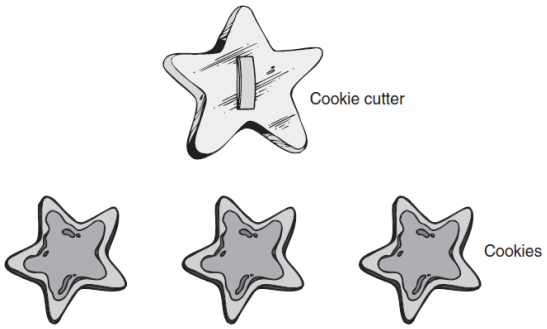
A class is like a blueprint from which many houses might be constructed, or like a recipe from which many meals can be prepared. One house and one meal are each an instance of their class.

The blueprint itself is not a house, but is a detailed description of a house. When we use the blueprint to build an actual house, we could say we are building an instance of the house described by the blueprint. If we so desire, we can build several identical houses from the same blueprint. Each house is a separate instance of the house described by the blueprint.



Gaddis _Starting Out with Programming Logic and Design

Another way of thinking about the difference between a class and an object is to think of the difference between a cookie cutter and a cookie. While a cookie cutter itself is not a cookie, it describes a cookie. The cookie cutter can be used to make several cookies, as shown in the figure below. Think of a class as a cookie cutter and the objects created from the class as cookies.



The data contained in an object is known as the *object's fields.* An object's fields are simply variables, arrays, or other data structures that are stored in the object. The procedures that an object performs are known as *methods*.

An object's data is stored in attributes. Attributes are the properties or characteristics that describe that particular object.

Example of an object: Alarm clock

Alarm clock has the following fields called attributes:

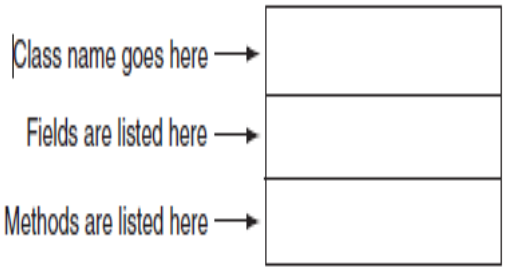
- second (a value in the range of 0–59)
- minute (a value in the range of 0–59)
- hour (a value in the range of 1–12)
- alarm set time
- alarm state : on or off (“on” or “off”)

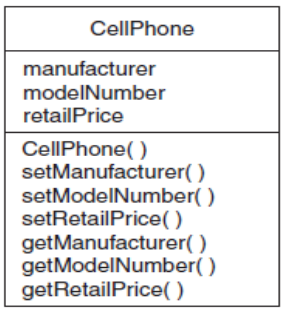
To change an object’s value, we must use one of the methods. The following are some of the alarm clock’s methods:

- Set time
- Set alarm time
- Turn alarm on
- Turn alarm off

Class diagram

- Object-oriented programming requires classes, attributes and methods to be represented in a class diagram.
- Consists of a rectangular box divided into three sections with the name of the class at the top, the attributes of the class in the middle and the methods at the bottom.





7.2.2 Inheritance

Inheritance involves a superclass and a subclass. The *superclass* is the general class and the *subclass* is the specialised class. The subclass inherits the fields and methods from the superclass without any of them having to be rewritten.

7.2.3 Polymorphism

Polymorphism allows you to create methods with the same name in different classes (that are related through inheritance) and gives you the ability to call the correct method depending on the type of object that is used to call it. The term *polymorphism* refers to an object's ability to take different forms. It is a powerful feature of object-oriented programming.

7.2.4 Encapsulation

OOP addresses the problem of code/data separation through encapsulation and data hiding. *Encapsulation* refers to the combining of data and code into a single object. *Data hiding* refers to an object's ability to hide its data from code that is outside the object. Only the object's methods may then directly access and make changes to the object's data. An object typically hides its data but allows outside code to access its methods.

7.3 Access modifiers

An access modifier restricts the access of a class, constructor, data member and method in another class. This concept is called 'visibility'. In Java we have four access modifiers:

Java uses four types of access modifiers:

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package. You can also say that the protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes. This access modifier is generally used in a parent child relationship.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Steps in creating an object-oriented solution

Three steps in creating an object-oriented solution for a problem with just one class:

1. Identify the objects and their attributes, responsibilities and operations
2. Design the algorithms for the operations and methods using structured design
3. Develop a test or driver algorithm to test the solution

7.3 Introduction to Programming

Computer memory stores patterns of electronic signals. The CPU reads, manipulates, and transforms patterns. Patterns can be stored/viewed as strings of bits. Sequences of 1s and 0sTo determine what a sequence of bits represents, you must know the context.

- **Bit (binary digit):** Smallest unit of information processed by a computer.
  - A single 0 or 1
- **Byte:** 8 adjacent bits
  - Capacity of computer memory and storage devices usually expressed in bytes.
- **Characters and Strings**
  - **ASCII (American Standard Code for Information Interchange) encoding scheme:** Each character represented as a pattern of 8 bits (1 byte); 256 characters may be represented.
  - Java uses **Unicode** encoding scheme: 2 bytes used to represent a character.

Language Elements

Every language, including Java has the following elements:

- Vocabulary: Set of all of the words and symbols in the language
- Syntax: Rules for combining words into sentences (*statements*)
- Semantics: Rules for interpreting the meaning of statements

Programming languages have small vocabularies and simple syntax and semantics.

- Programming language syntax must be absolutely correct.
- Programming language statements are interpreted literally.
- Every detail must be present.

Java uses two categories of data types:

1. **Primitive data types**: integer, float, double, character, boolean
2. **Complex data types:** object, string, arrays

Variable declaration statement:

Declares the identifier and data type for a variable

- int age; (declares one int variable)
- int a, b, c; (declares three int variables)
- double d = 2.45; (declares and initialises a variable)

- Constants are variables whose value cannot change.
  - final double PI = 3.14;

Comments

- Explanatory sentences inserted in a program. Compiler ignores them. Purpose is to make program more readable.
- Two varieties:
  - **End of line comments:** All text following a double slash (//) on a single line
  - **Multiline comments:** All text occurring between a /* and a */

Programming Errors

Three types of programming errors we find in Java:

- **Syntax errors**: These errors occurs when a syntax rule is violated. These errors are detected during compilation. Compiler helps identify the syntax error.
- **Run-time errors:** These error **o**ccurs during program execution. Example is dividing by 0 error. This will be detected only when the program runs. JVM indicates type of error and location.
- **Logic errors (design errors** or **bugs):** These error occurs when incorrect logic is implemented in the program. Code may be correct in every other way, but does not do what it is supposed to do. Need to thoroughly test and **debug** the program when an error is found.
  - **Desk checking:** Examine code immediately after it is written.

REFERENCES

Robertson, L. (2007). Simple program design. Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving. Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology.

Topic 8: Introduction to JAVA Programming

Chapter Objectives

At the completion of this chapter, you would have learnt:

- Discuss why Java is an important programming language
- Explain the Java virtual machine and byte code
- Choose a user interface style
- Describe the structure of a simple Java program
- Edit, compile, and run a program using a Java development environment

Introduction

James Gosling developed Java at Sun Microsystems in 1995.It was created with the idea to write one code one time and run it on any operating system, hence the term "**Write Once, Run Anywhere**."

Java can be described as a "simple, object-oriented, multi-threaded, garbage-collected, secure, robust, architecture-neutral, portable, high performance, dynamic language." Soon after its introduction, WWW browsers were updated to include applets, known as small Java programs, in HTML documents. First through applets, and later through complete Internet applications, Java has become widespread on the Internet. Its built-in support for URLs and the HTTP protocol allows Java applications to access standard WWW servers. These Java features have improved the feasibility of the development of Internet hypermedia that does not rely on a WWW browser for a hypermedia client.

Why Java?

Java is the world's fastest growing programming language.

- **Secure:** Enables construction of virus-free, tamper-free systems.
- **Robust:** Supports development of programs that do not overwrite memory.
- **Portable:** Programs can be run on different types of computers without change.
  - Java is well-suited for distributed, network-based applications.
  - Java supports **threads**.
    - A process that can run concurrently with other processes.
- Java resembles C++. Easy for C++ programmers to learn Java. Java runs more slowly than other languages.
- The Java compiler translates Java source code into Java **byte code** - A pseudo-machine language. Byte code is executed by the **Java Virtual Machine (JVM)**.
- **Interpreter:** A program that behaves like a computer. Any computer with the JVM installed can execute Java byte code (portability).
- Some JVMs support **just-in-time compilation (JIT)**. Translate byte code instructions into machine language when they are first encountered.
- **Java applets:** Java programs that run in a Web browser. A JVM is incorporated into the browser.

How to write, compile, and execute our first Java program.

Step1. Go to NetBeans, and write the following code.

```
1   public class Program1
2   {
3       public static void main(String[] args)
4       {
5           System.out.println("Welcome to Java Program");
6       }
7   }
```

Save it as Program1.java, remember the class name and the file name should be the same.

Step2. Run the file.

It will show the following output:

```
run:
Welcome to Java Program
BUILD SUCCESSFUL (total time: 0 seconds)
```

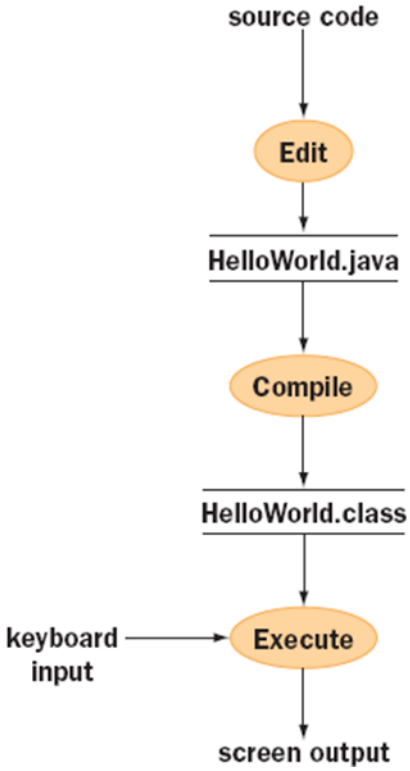
Line 1: public class Program1
Here public is an access specifier which means this class can be visible to all, class is a keyword and Program1 is the name of the class.

Line 3: public static void main(String[] args)
main() is a method which specifies the start of a program, void is return type of main method which means it will not return any value. If we declare any method as static it means that it does not require any object to execute that method. String[] args is an array of strings which is used for command line arguments.

Line 5: System.out.println("Welcome to Java Program");
It can be considered as a print statement, anything written inside double quotes will be displayed on the console. System is a predefined class, out is the object of PrintStream class, println() is the method of PrintStream class.

The following figure shows how the source code is compiled and executed. The file extension for a Java source code file is .java. The file extension for a Java class file is .class.

Data types and operations



A data type is a set of data with values having predefined characteristics.
Example: Integer, floating point, character, etc.
Data types are broadly divided into two types.

1. Primitive data types
2. Non-primitive data types

Primitive data types

There are eight primitive data types - **byte, short, int, long, float, double, char and boolean**,
byte, short, int, long involve whole numbers without any decimals whereas float, double contains decimal values.*boolean* contains TRUE or FALSE value. Char contains only one single character.

Type	Size	Range
byte	8 bits	-128 .. 127
short	16 bits	-32,768 .. 32,767
int	32 bits	-2,147,483,648 .. 2,147,483,647
long	64 bits	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807
float	32 bits	$3.40282347 \times 10^{38}$, $1.40239846 \times 10^{-45}$
double	64 bits	$1.7976931348623157 \times 10^{308}$, $4.9406564584124654 \times 10^{-324}$

Example of using different data in Java program:

```java
public class Program2
{
    public static void main(String[] args)
    {
        int a = 10;
        float b = 10.50f;
        char c = 'a';
        boolean d = false;
        System.out.println("the int value is : "+a);
        System.out.println("the float value is : "+b);
        System.out.println("the char value is : "+c);
        System.out.println("the boolean value is : "+d);
    }
}
```

Output:

```
Output - LessonICS (run)
run:
the int value is : 10
the float value is : 10.5
the char value is : a
the boolean value is : false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Operators used in Java Program

An operator is a symbol that helps the compiler to do a particular logical or mathematical operation.

- Arithmetic Operators
- Relational or Comparison Operators
- Logical Operators

Arithmetic operators

Given: A=10, B=4

Operator	Description	Example
+	Addition, adds two values	A + B=14
-	Subtract A and B	A - B=6
*	Multiplication, multiplies two values.	A * B = 40
/	Division, divides left side value by right side value	A / B = 2
%	Modulo returns remainder after division.	A % B = 2
++	Increases the value by 1.	A++ = 11
--	Decreases the value by1.	B-- = 3

Relational Operators

Given: A=2, B=1

Operator	Description	Example
==	Returns true if two values are equal else false	(A == B) is false
!=	Returns true if two values are not equal else false	(A != B) is true
>	Returns true if left side value is strictly greater than right side value else false.	(A > B) is true
<	Returns true if left side value is strictly lesser than right side value else false.	(A < B) is false
>=	Returns true if left side value is greater than or equal to right side value else false.	(A >= B) is true
<=	Returns true if left side value is lesser than or equal to right side value else false.	(A <= B) is false

Logical Operators

Given A=0 and B=1

Operator	Description	Example
&&	Return true if both values are non-zero else false	A && B= false
||	Return true if any of the two values is non zero else false	A || B=true
!	Return True if the value is zero else false	!A = true

Example: Convert the following pseudocode into Java program:

1. Set test1 = 88.0
2. Set test2 = 92.5
3. Set test3 = 97.0
4. Average = (test1 + test2 + test3) / 3
5. Display Average

END

Declaring Variables before using them

The purpose of a variable declaration statement is to tell the compiler or interpreter that you plan to use a particular variable in the program. A variable declaration statement typically causes the variable to be created in memory. For this reason, you have to write a variable's declaration statement *before* any other statements in the program that use the variable. This makes perfect sense because you cannot store a value in a variable if the variable has not been created in memory.

For example, look at the above pseudocode. If this code were converted to actual code in a language like Java, it would cause an error because the SET statement uses the test1, test2, test3 variables before the variable has been declared. In pseudocode we do not declare the variable as its design of the program. In actual development of the program, all the variables used in the program must be declared before it is used. The below Java program shows how to declare the variable before it is used.

Java Program

The Java program below calculates the average of three test scores. When we convert the pseudocode to Java program,

```
public class Program3
{
    public static void main(String[] args)
    {
        double test1, test2, test3, Average;
        test1 = 88.0;
        test2 =92.5;
        test3 = 97.0;
        Average = (test1 + test2 + test3)/3;
        System.out.println("Average : "+Average);
    }
}
```

Output:

```
Output - LessonICS (run)
run:
Average : 92.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

REFERENCES

Robertson, L. (2007). Simple program design. Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving. Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology.

Herbert Schildt, Java: The Complete Reference,11th Edition, McGraw-Hill Education.

Topic 9: Introduction to Selection Control

Chapter Objectives

At the completion of this chapter, you would have learnt:

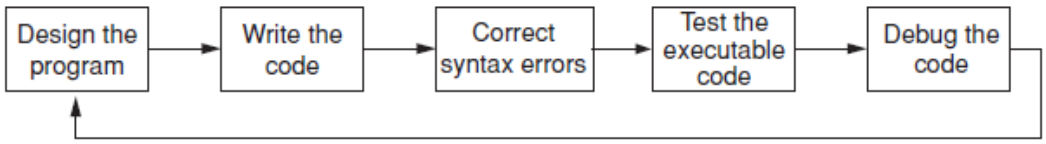
- To convert the algorithm to appropriate Java program
- Use if and if-else statements to make choices.
- Construct appropriate conditions for control statements using relational operators.

Introduction

Statements in Java can be either simple statements or compound statements. Simple statements, such as assignment statements and subroutine call statements, are the basic building blocks of a program. Compound statements, such as while loops and if statements, are used to organise simple statements into complex structures, which are called control structures because they control the order in which the statements are executed.

Control statements are the statements that define the flow of programs. There are 3 types of control statements in Java: Selection, iteration, and jump statements. In this topic, we are going to convert all the selection pseudocode exercises we have done in the topic 4 to Java programs.

The program development cycle



- Design the Program: Programs must be carefully designed before they are written. Use design tools such as pseudocode and flowcharts to create models of programs.
- Write the code: After designing the program, the programmer begins writing the code in a high-level language. Recall from Chapter 7 that each language has its own rules, known as syntax that must be followed when writing a program. A language's syntax rules dictate things such as how key words, operators, and punctuation characters can be used.
- Correct syntax errors: The programmer has to identify the syntax errors when it is first written and spend adequate time correcting these.
- Test the executable code and debug the code: After debugging the syntax errors and simple typing mistakes, the program can be compiled and translated into a machine language program (or executed by an interpreter, depending on the language being used).

Let us convert all the pseudocode example to Java program;

Example 1: Simple selection with null false branch (null Else statement)

If the condition is true, we follow one path, which leads to an action being performed. If the condition is false, we follow another path, which skips the action.

```
1. Prompt for num1, num2
2. Get num1, num2
3. IF number <> 2 THEN
       answer = num1/num2
   ENDIF
4. Display answer
   END
```

Java Program for the above pseudocode:

```java
import java.util.Scanner;
public class Example1
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number1, number2, answer; // declaration
        System.out.print("Enter number 1 : ");
        number1 = keyboard.nextInt();
        System.out.print("Enter number 2 : ");
        number2 = keyboard.nextInt();
        if(number2 != 0)
        {
            answer = number1/number2;
            System.out.println("answer = " +answer);
        }
    }
}
```

The above program only displays the answer when the condition is true. When number 2 is not equal to zero, then it will display the answer as shown below:

```
Output - LessonICS (run)
run:
Enter number 1 : 25
Enter number 2 : 3
answer = 8
BUILD SUCCESSFUL (total time: 7 seconds)
```

The above program will terminate the condition if it is not true. When number 2 is equal to zero, then it will terminate the program as shown below:

```
Output - LessonICS (run)
run:
Enter number 1 : 12
Enter number 2 : 0
BUILD SUCCESSFUL (total time: 12 seconds)
```

Example 2: Simple selection (simple IF statement)

The if-else statement has two blocks of codes, depending on the value of the condition. If the condition is true, then if block executes otherwise else block executes.

Pseudo code

```
1.  Prompt for number1
2.  GET number1
3.  Prompt for number2
4.  GET number2
5.  IF number2 = 0 THEN
    DISPLAY "Division by zero   error"
    ELSE
    answer = number1/number2
    DISPLAY answer
   ENDIF
END
```

Java program:

```
import java.util.Scanner;
public class Example1
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number1, number2, answer; // declaration
        System.out.print("Enter number 1 : ");
        number1 = keyboard.nextInt();
        System.out.print("Enter number 2 : ");
        number2 = keyboard.nextInt();
        if(number2 == 0)
        {
            System.out.println("Division by zero error");
        }
        else
        {
            answer = number1/number2;
            System.out.println("answer = " +answer);
        }
    }
}
```

After writing the Java program, we need to test the program by inputting valid and invalid data types. The program gives the expected result, then the design and development of the program is completed.

Output for the condition true:

```
Output - LessonICS (run)
run:
Enter number 1 : 12
Enter number 2 : 0
Division by zero error
BUILD SUCCESSFUL (total time: 7 seconds)
```

Output for the condition false:

```
Output - LessonICS (run)
run:
Enter number 1 : 12
Enter number 2 : 3
answer = 4
BUILD SUCCESSFUL (total time: 5 seconds)
```

Example 3: Nested if selection structure

B Solution algorithm

Note that the shipping charge for each parcel is determined as a certain cost, *per kilogram* of parcel weight.

```
    Calculate_shipping_charge
1       Prompt for parcel_weight
2       Get parcel_weight
3       IF parcel_weight < 3.5 THEN
          shipping_charge = 4.50 * parcel_weight
        ELSE
          IF parcel_weight >= 3.5 AND parcel_weight <= 8 THEN
            shipping_charge = 3.30 * parcel_weight
          ELSE
            shipping_charge = 2.75 * parcel_weight
          ENDIF
        ENDIF
4       Display shipping_charge
    END
```

Java program:

```
import java.util.Scanner;
public class Example3
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        double parcel_weight, shipping_charge; // declaration
        System.out.print("Enter parcel weight : ");
        parcel_weight = keyboard.nextDouble();
        if(parcel_weight < 3.5)
        {
            shipping_charge = parcel_weight * 4.50;
        }
        else if(parcel_weight >= 3.5 && parcel_weight <=8)
        {
            shipping_charge = parcel_weight * 3.30;
        }
        else
        {
            shipping_charge = parcel_weight * 2.75;
        }
       System.out.printf("Shipping charge : $%.2f\n", shipping_charge);
    }
}
```

If we follow the flow of execution, we will see that the condition parcel weight < 4.5 is tested. If the condition is true, then it will calculate the shipping charge and display the display charge. However, if the condition is false, it will test the next condition and this condition is false then it will calculate the shipping charge of the last option and display the shipping charge.

After completing the program we need to test the program for the test data which we had tested in desk checking algorithm (Topic 3 – page 13) shown below:

C Desk checking

i Input data:

	First data set	Second data set
parcel_weight	2	4

ii Expected results:

	First data set	Second data set
shipping_charge	9.00	13.20

First data set	Second data set
Output - LessonICS (run) run: Enter parcel weight : 2 Shipping charge : $9.00 BUILD SUCCESSFUL (total time: 11 seconds)	Output - LessonICS (run) run: Enter parcel weight : 4 Shipping charge : $13.20 BUILD SUCCESSFUL (total time: 2 seconds)

The program produced the above output, which is the same as desk checking result.

Example 4: Switch…case structure

Let us try to write a Java program for the following pseudocode that uses case structure.

B. Solution Algorithm (Pseudocode)

```
Display_Result
    1. Prompt for Num1, Num2, Code
    2. Get Num1, Num2, Code
    3. SET Result = 0, msg = blank
    4. CASEOF(Code)
          1 :  Result = Num1 + Num2
          2 :  Result = Num1 - Num2
          3 :  Result = Num1 * Num2
          4 :  Result = Num1/  Num2
          Other : msg = 'Invalid Code'
       ENDCASE
    5. IF (msg == blank) THEN
             Display Num1, Num2, Code, Result
       ELSE
             Display msg
       ENDIF
       END
```

Java Program

```
import java.util.Scanner;
public class Example4
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int Num1, Num2, Code, Result = 0; // declaration
        String msg=null;
        System.out.print("Enter number 1 : ");
        Num1 = keyboard.nextInt();
        System.out.print("Enter number 2 : ");
        Num2 = keyboard.nextInt();
        System.out.print("Enter code : ");
        Code = keyboard.nextInt();
        switch(Code)
        {
            case 1:
                Result = Num1 + Num2;
            break;
            case 2:
                Result = Num1 - Num2;
            break;
            case 3:
                Result = Num1 * Num2;
            break;
           case 4:
                Result = Num1 / Num2;
            break;
            default:
                msg = "Invalid code";
            break;
        }
        if(msg == null)
        {
             System.out.printf("Number 1 = %d \n", Num1);
             System.out.printf("Number 2 = %d \n", Num2);
             System.out.printf("Code     = %d \n", Code);
             System.out.printf("Result   = %d \n", Result);
        }
        else
        {
            System.out.printf(msg);
        }


    }
}
```

The *case structure* is a *multiple alternative decision structure*. It allows you to test the value of a variable or an expression and then use that value to determine which statement or set of statements to execute. The above java program shows an example of how a case structure is written in Java.

If the code is 1, addition is executed. If code is 2, subtraction is executed. If the code is 3, multiplication is executed. If the code is 4, division is executed. If the code contains none of these values, the statement labelled default is executed. In this case, the statement sets message to Invalid code.

After writing the program, we test the program using the data we tested in desk checking:

C. Desk checking

(i) Input data

	First Data set	Second data set	Third Data Set
Num1	45	-10	-5
Num2	20	24	-4
Code	1	2	7

(ii) Expected Result

	First Data set	Second data set	Third Data Set
Num1	45	-10	
Num2	20	24	
Code	1	2	
Result	65	-34	
msg			Invalid code

Java program output:

First data set	Second data set	Third data set
run: Enter number 1 : 45 Enter number 2 : 20 Enter code : 1 Number 1 = 45 Number 2 = 20 Code = 1 Result = 65 BUILD SUCCESSFUL (total	run: Enter number 1 : -10 Enter number 2 : 24 Enter code : 2 Number 1 = -10 Number 2 = 24 Code = 2 Result = -34 BUILD SUCCESSFUL (tota	run: Enter number 1 : -5 Enter number 2 : -4 Enter code : 7 Invalid codeBUILD SUCC

REFERENCES

Robertson, L. (2007). Simple program design. Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving. Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology.

Herbert Schildt, Java: The Complete Reference,11th Edition, McGraw-Hill Education.

Topic 10: Introduction to REPETITION Control

Chapter Objectives

At the completion of this chapter, you would have learnt:

- To convert the algorithm to appropriate Java program
- Use while and for and do while loops to repeat a process
- Construct appropriate conditions for control statements using relational operators.
- Detect and correct common loop errors

Introduction

When we use a loop, we can write one set of instructions that operates on multiple, separate sets of data. Recall the loop structure that you learned about in Chapter 4; As long as a Boolean expression remains true, a while loop's body executes. We have already designed many programs using a loop to control structure. In this chapter, we will learn how to convert the algorithm into Java program.

Loop Design Strategies

Loop design comprises of several steps. To properly understand a loop, we must consider:

- pre-condition
- post-condition
- loop variant
- loop invariant

We must properly define all four criteria to ensure a loop that is correct and terminates.
Pre-condition: It must be properly defined and true before the loop executes.

Post-condition: It must be true upon loop termination.

Loop variant (exit condition): It controls the execution of a loop which must be properly defined to ensure loop termination.

The most important aspect of a loop is the loop invariant. The loop invariant is a statement that is true before and after each iteration of the loop. While the values of variables may change, it is the truth of the loop invariant that does not vary.

Many statements can be made that are true both before and after each iteration of the loop, but the loop invariant must relate the work that the loop does such that when the loop is finished the truth of the loop invariant implies the desired result.

Let us convert all the pseudocode examples to Java program:

Example 1: Pseudocode - DOWHILE structure

```
1. SET total = 0
2. Prompt for number
3. GET number
4. DOWHILE number < 1000
5.         total = total + number
6.         Prompt for number

7.         GET number
    ENDDO
8. DISPLAY total
    END
```

Java Program:
Pseudocode DOWHILE structure is a while loop in a Java program. The program will keep prompting user to input a number. If the number is below 1000, it will calculate the total and prompt for the user to input a number. Once the user input the number above 1000, the loop terminates and displays the total.

```java
import java.util.Scanner;
public class Example_1
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number, total=0; // declaration
        System.out.print("Enter number  : ");
        number = keyboard.nextInt();
        while(number < 1000)
        {
            total = total + number;
            System.out.print("Enter number  : ");
            number = keyboard.nextInt();
        }
        System.out.println("Total = " +total);
    }
}
```

Sometimes, loop control may need to be based on the value of what we are processing. In this case, we would use a sentinel-controlled loop. Sentinel-controlled loop is sometimes called indefinite repetition because it is not known in advance how many times the loop will be executed.

Output:

```
run:
Enter number  : 10
Enter number  : 234
Enter number  : 589
Enter number  : 789
Enter number  : 321
Enter number  : 1020
Total = 1943
BUILD SUCCESSFUL (total time: 20 seconds)
```

Example 2: Pseudocode - Counted loop structure

Counted repetition construct will perform the initialising, incrementing and testing of the loop counter automatically. It will also terminate the loop once the required number of repetitions has been executed.

```
1.  DO i = 1 TO 12
2.          result = i * 9
3.          Display (i, "*9=",result)

    ENDDO
```

Java Program:

Pseudocode DO... ENDO structure is called for loop in Java program.

```java
import java.util.Scanner;
public class Example_2
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int i, result; // declaration
        for(i=1; i<=12; i ++)
        {
            result = i * 9;
            System.out.println(i + " * 9  = " +result);
        }

    }
}
```

Output:

```
run:
1 * 9   = 9
2 * 9   = 18
3 * 9   = 27
4 * 9   = 36
5 * 9   = 45
6 * 9   = 54
7 * 9   = 63
8 * 9   = 72
9 * 9   = 81
10 * 9   = 90
11 * 9   = 99
12 * 9   = 108
BUILD SUCCESSFUL
```

Example 3: Pseudocode- REPEAT…UNTIL structure

REPEAT…UNTIL structure is similar to the DOWHILE structure (group of statements are repeated in accordance with a specific condition). REPEAT…UNTIL structure is called trailing decision loop because it tests the condition at the end of the loop.

```
SET count =1
REPEAT
        Prompt for number
        GET number
        Square = number * number
        Display square
        count = count + 1
UNTIL (count >5)
 ENDDO

END
```

Java Program:

Pseudocode REPEAT … UNTIL structure is called do … while loop in Java program.

```java
import java.util.Scanner;
public class Example_3
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number, count=1, square; // declaration
        do
        {
            System.out.print("Enter number  : ");
            number = keyboard.nextInt();
            square = number * number;
            System.out.printf("%d square is %d \n", number, square);
            count++;
        }while(count <=5);
    }
}
```

Output:

```
run:
Enter number  : 12
12 square is 144
Enter number  : 5
5 square is 25
Enter number  : -6
-6 square is 36
Enter number  : 10
10 square is 100
Enter number  : -14
-14 square is 196
BUILD SUCCESSFUL (total time: 1
```

Example 4: Algorithm using DO…WHILE loop to print 4 times table

B. Solution Algorithm (Pseudocode)

Display_FourTimesTable (DOWHILE loop)

```
1. SET x =1
2. DOWHILE (x <=10)
3.      Product = 4 * x
4.      Display  '4 * x = product '
5.      x = x + 1
   ENDDO
   END
```

Java program for the above Pseudocode:

```java
import java.util.Scanner;
public class Example_1
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int x, product; // declaration
        x=1;
        while(x <= 10)
        {
            product = 4 * x;
             System.out.printf("4 * %2d = %2d\n", x, product);
            x = x+1;
        }
    }
}
```

After completing the program we need to test the program for the test data which we had tested in desk checking algorithm (Topic 4 – page 8) shown below:

C. Desk Checking

(i) Input data

	First Data set
x	1, 2, 3,.....

(ii) Expected Result

First Data set
4 * 1 =4 4 * 2 =8 7 * 3 =12....

The program produced the output below, which is the same as desk checking result.

```
run:
4 *  1 =  4
4 *  2 =  8
4 *  3 = 12
4 *  4 = 16
4 *  5 = 20
4 *  6 = 24
4 *  7 = 28
4 *  8 = 32
4 *  9 = 36
4 * 10 = 40
BUILD SUCCESSFUL
```

REFERENCES

Robertson, L. (2007). Simple program design. Australia: Course Technology.

Savitch, W., (2014). Java: Introduction to Problem Solving. Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology.

Herbert Schildt, Java: The Complete Reference,11th Edition, McGraw-Hill Education.

Kaplan City Campus @ Wilkie Edge | 8 Wilkie Road, Level 2, Singapore 228095
Kaplan City Campus @ PoMo | 1 Selegie Rd, Level 7, Singapore 188306

6733 1877 info.sg@kaplan.com

KaplanSingapore

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 01 – System Life Cycle and Program Design

1

Topic 01 – Learning Outcomes

- The concept of a system;
- The various stages of a system life cycle;
- To identify the activities and people involved in each stage.
- To describe the steps in the program development process
- To introduce current design methodology

2 KAPLAN

2

Introduction

- An Information system is collection of interrelated components that collect, process, store, and provide as output the information needed to complete business tasks.
- For example, Human resource management (HRM) system – a system that supports such employee-related tasks as payroll, benefits, hiring, and training.
- System Life Cycle is a method for developing Information System which consists of either a program or a collection of programs.
- Therefore, a system life cycle describes how programs are developed.
- System Life Cycle describe the stages, starting from when an idea is conceived, to the stage when the completed system is reviewed.
- After that we will discuss about the Program Design.

3

KAPLAN

3

What Is A System?

- We have been using the word “system” quite freely, but before we discuss all the stages of a system life cycle and the part that a programmer plays, we need to be clear what a system is.
- Many definitions exist, and we can also find satisfactory definitions in the publications of various standard institutions.

4

KAPLAN

4

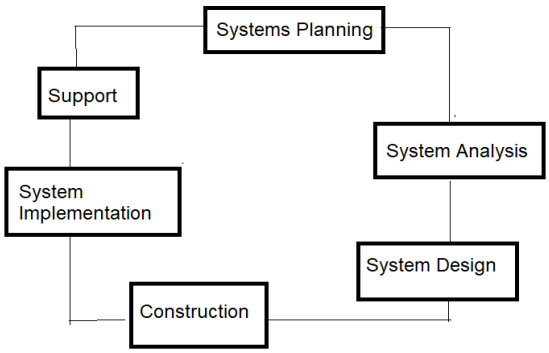
What Is A System?

- The environment in which any business system or enterprise function does not remain static for very long.
- Because of this, the systems has to change. When there are changes, the system will eventually need to be replaced, or significantly altered to become a new system or a new version.
- The process of replacing the old system with the new, happens in a series of stages and the whole process is called the “system life cycle”.
- The system life cycle can be divided into five phases, as shown in Figure 1-1. Although each phase is presented discretely, it is never accomplished as a separate step.

5

5

Stages Of A System Life Cycle



6

6

Introduction To Program Design

- In this module we are going learn Program design in detail.
- Programming is really all about solving problems...but we use a computer to do that for us

7

7

Steps In Program Development

There are seven steps used in program development. The steps are shown below:

1. Define the problem.
2. Outline the solution.
3. Develop the outline into an algorithm.
4. Test the algorithm for correctness.
5. Code the algorithm into a specific programming language.
6. Run the program on the computer.
7. Document and maintain the program.

8

8

Errors In Program

- **Syntax errors:** Error in the rules or keywords of a specific programming language. E.g. forget to put a semicolon at the end of the statement.
- **Logic errors:** Error in the calculations or using incorrect operators or symbols. E.g. using a greater than symbol (>) instead of less than (<). Logic errors will execute the program but the outputs will not be the same as expected.
- **Runtime errors:** Errors during the execution of program. E.g. Accessing 6th element from an array where the last index is 5.

9

KAPLAN

9

Algorithm, Pseudo Code And Flowchart

- **Algorithm:** set of detailed instructions that describe the processes necessary to follow in order to produce a desired output. An algorithm is like a recipe. Follow the set of instructions step-by-step.
- Properties of an algorithm:
  - -An algorithm must be very clear, precise and self-explanatory.
  - -An algorithm must return correct results for all different types of inputs.
  - -It doesn't matter how many times the algorithm executes but eventually it must end.

10

KAPLAN

10

Algorithm, Pseudo Code And Flowchart

- **Pseudo code:** Structured English language like instructions in solving a problem. They resemble high level computer language instructions. E.g. IF, ELSE etc.,
- A **flow chart** is a pictorial representation of the logical steps it takes to solve a problem. It uses the standard symbols in designing input, process, and output of a program. The program flow should be read from top to bottom, left to right.

11

KAPLAN

11

Algorithm, Pseudo Code And Flowchart

Example: Program to convert the given length in feet to centimeter:

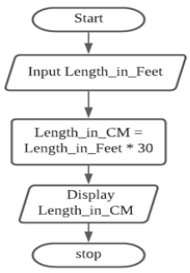
Algorithm:

- Step 1 : Input Length in feet
- Step 2 : Length in centimetre = Length in meter * 30
- Step 3: Display Length in centimeter.

Pseudocode:

1. Prompt for Length_in_Feet
2. Get Length_in_Feet
3. Length_in_CM = Length_in_Feet*30
4. Display Length_in_CM
5. END

Flowchart

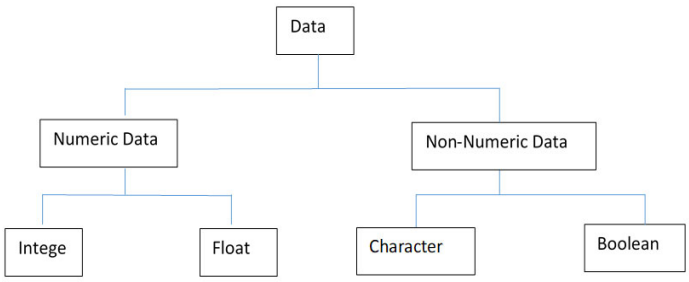


12

KAPLAN

12

Data Types Used In Program



13

13

Identifiers

- All data items used in a program need to be declared before usage.
- This means to state down the name given to each data item storage cell known as “identifier”.

  - An identifier may be numeric or non-numeric.
  - The data type that can be contained in the memory location will depend on the declaration of the data.
  - These memory location can be a *constant* value or *variable* value.

14

14

Variable And Constant

Constants

- It is a “Read-only” variable which contains fixed data values that cannot be changed during the program execution.
- Constant: As the name suggests, a constant is a value that remains the same throughout the execution of program.
- e.g. pi = 3.14

Variables

- The value that is stored in memory which changes during the execution of a program. During programs/algorithms we always use variables to store values.
- A variable can store only one value at a time.
- e.g. num, sum, income etc.,

15

KAPLAN

15

Meaningful Names

- It is essential to have unique names for variables or objects in programs.
- Names should be meaningful and must be easy to understand.
- E.g. age is better than a
- Names should be transparent to adequately describe variables (Integer1, int2, etc.).
- Programming language does not tolerate a space between variable names if multiple words are used.
- In such instances, an underscore is used. (income_tax or number_of_employees).
- Another method is to use capital letters as a word separator (incomeTax or numberOfEmployees).

16

KAPLAN

16

Summary Of Rules:

- ✓ Variable names should always start with an alphabet
- ✓ Variable names are case sensitive
- ✓ No spaces between variable names if using multiple words
- ✓ No special characters in variable names
- ✓ Keywords are not supposed to be used as variable name.

17

17

Data Structures Used In Program

- Data items grouped logically together to give information about an individual.
- For example, an address book is a data structure containing data items (names, addresses, phone numbers etc.).

  -Array
  -String
  -File
  -Record

18

18

The End

19

19

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 02 – Program Design

1

Topic 02 – Learning Outcomes

- Introduce keywords when writing pseudo code.
- Introduce flowchart Symbols used for different functions.
- To define the three basic control structures as set out in the Structure Theorem
- To illustrate the three basic control structures using pseudo code and flowchart

2 KAPLAN

2

Introduction

- Program is set instruction to computer to perform certain task.
- That means, system accept input from the user as data, process the data and display the output
- Computer require data to perform task.
- In this chapter, we will discuss, basic computer operations and how write pseudo code and Flow chart for these operations.

3

3

Six Basic Computer Operations

- A computer can receive information
- A computer can put out information
- A computer can perform arithmetic
- A computer can assign a value to a variable or memory location
- A computer can compare two variables and select one of two alternate actions
- A computer can repeat a group of actions

4

4

Pseudo Code Keywords For Computer Basic Operations

- **Computer Basic operation 1: A computer can receive information**
- To receive information, the following keywords are used.
  - -GET
  - -READ
- Get is used to receive input from the user or keyboard. Get will receive a value from user and store them in a variable.
- Read is used to read input data values from a file.
- There are many other keywords used, but the most popularly used keywords are shown below:

5

5

Computer Basic Operation 2: A Computer Can Put Out Information

- To output information, the following keywords are used.
  - -DISPLAY
  - -PRINT
- The verbs Print, Write, Put, Output or Display are used in pseudocode when a computer is required to supply information or output to a device.
- Print is used when the output is to be sent to a printer.

6

6

Computer Basic Operation 3: A Computer Can Perform Arithmetic:

- Computer performs arithmetic operations such as addition, subtraction, multiplication, division and modulus calculations. In pseudo code, the following keywords or operators are used to represent arithmetic operations.
  - -Add (+)
  - -Subtract (-)
  - -Product (*)
  - -Divide (/)
  - -Remainder (%)
  - -Orders of operation
    - -BIDMAS/BODMAS
    - -Brackets Index Division Multiplication Addition Subtraction
- Examples of order of operation:
  - -5 + 2 * 8 = 21
  - -(5 + 2) * 8 = 80

7

KAPLAN

7

Computer Basic Operation 4: Computer Can Assign A Value To A Variable Or Memory Location.

- To assign a value to a variable we use SET as the keyword.

  Example:

```
SET Sum to zero
SET COUNT = 0
SET message to Blank
```

8

KAPLAN

8

Computer Basic Operation 5: A Computer Can Compare Two Variables And Select One Of Two Alternate Actions

- For comparison of two variables we use IF THEN ELSE ENDIF as the keywords

Comparisons operators:

>
>=
<
<=
==
!= or <>

9

KAPLAN

9

Computer Basic Operation 6: A Computer Can Repeat A Group Of Actions

- For repetitions we use the following keywords:
  - -DOWHILE…ENDDO
  - -DO …. ENDO
  - -REPEAT….UNTIL

10

KAPLAN

10

Flow Chart Symbols

- Input / Output (parallelogram)
- Processing (rectangle)
- Processing (rectangle)
- Decision (diamond)
- Terminal (racetrack)
- Connector (circle)
- Program Flow

11

The Structure Theorem

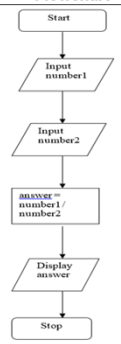
Programs produced using only the three kinds of control structures called sequential, selection and repetition, are called **structured programs**.

Structured programs are easier to test, modify, and are also easier for other programmers to understand.

12

Sequence Structure

- Step by step execution of the program
- First four steps of the computer operations represents sequence

Pseudo code	Flowchart
1. Prompt for number1 2. GET number1 3. Prompt for number2 4. GET number2 5. answer = number1 / number2 6. DISPLAY answer END	

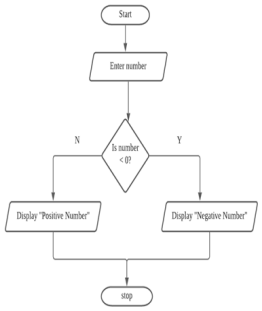
13

13

Selection Structure

- Asks a question, then takes one of two possible courses of action based on the answer. It is also called a **decision structure** or an **if-then-else**
- Fifth step of the computer operations represents selection

```
1. Prompt for number
2. Get number
3. IF (number < 0) THEN
        Display 'Negative Number'
   ELSE
        Display 'Positive Number'
   ENDIF
   END
```



14

14

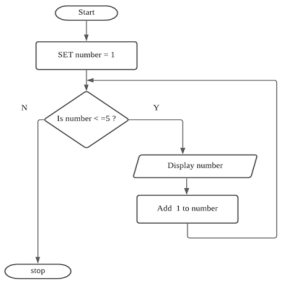
Repetition Structure

- Repeats a set of actions based on the condition. Also called **loop** or **iteration**
- The statement block will repeat if the condition is true.
- Sixth step of the computer operations represents repetition

1. Set number to 1
2. DOWHILE (number < =5)
3. Display number
4. Add 1 to number

ENDDO

END



15

15

References

Robertson, L. (2007). Simple program design (5th ed.). Australia: Course Technology.

Savitch, W., (2014) Java: Introduction to Problem Solving (7th ed.). Prentice Hall.

Joyce Farrell, A guide to Programming Logic and Design, Course Technology

16

16

The End

17

KAPLAN

17

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 03 – introduction to programming using sequence and selection control structure

1

Topic 03 – Learning Outcomes

At the completion of this chapter, you would have learnt:

- To introduce methods of analysing a problem and developing a solution using Pseudo code and Flowchart
- To introduce methods of manually checking the developed solution
- To develop simple algorithms using the sequence control structure
- To develop simple algorithms using the selection control structure

2 KAPLAN

2

Introduction

- Programs must be carefully designed before they are written.
- During the design process, programmers use tools such as pseudo code and flowcharts to create models of programs.
- Program Design consists of the steps a programmer should do before they start coding the program in a specific language.
- In this chapter, we shall look into those steps.

3

3

Program Design

- Program design must be extremely structured, having the ultimate intentions of performing a specific calculation efficiently with attractive, understandable, efficient programs.
- Three broad areas of activities in program design:
  (1) Understanding the Program,
  (2) Using Design Tools to Create a Model and
  (3) Develop Test Data

4

4

Program Design Is Divided Into Three Sections:

A defining the program,
B designing a solution algorithm,
C desk checking the solution algorithm

5

5

A . Defining The Program

- It is the first step in the program development. Need to identify the input required for the program, expecting output of the program and processing steps.

- Any given program should be divided into three separate components:

  -Input: list of data required for the program to run
  -Processing: a list of actions needed to produce the required outputs
  -Output: list of expected outputs from the program.

Input	Processing	Output
List all the inputs	Explain what needs to be done NOT how to be done.	List all the outputs

6

6

B. Designing A Solution Algorithm

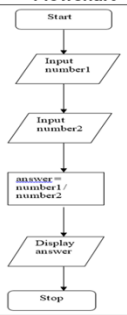
- Designing solution algorithm is the most challenging task in the program development.
- First attempt at designing a solution algorithm usually does not result in a finished product.
- Pseudo code is useful in the trial-and-error process where it is relatively easy to add, alter or delete instructions.

7

KAPLAN

7

Designing A Solution Algorithm

Pseudo code	Flowchart
1. Prompt for number1 2. GET number1 3. Prompt for number2 4. GET number2 5. answer = number1 / number2 6. DISPLAY answer END	

If you translated this pseudocode into an actual program and ran it, the first statement would execute, followed by the second statement, and followed by the third statement, fourth, fifth and sixth statement. If the sequence of 1, 2, and 3, 4 is reversed, there will be an error. This is called sequence control structure.

8

KAPLAN

8

C. Desk Checking The Solution Algorithm(testing)

- Desk checking is going through each of the program's statements one by one.
- As we carefully look at each statement, we record the contents of each variable on a sheet of paper after the statement executes.
- This process is often helpful in finding mathematical mistakes and other logic errors.
- Desk checking helps detect errors early and allows the user to become familiar with the way the program runs.

9

9

Steps In Desk Checking

1) Choose simple input test cases that are valid
2) Establish the expected result for each test case
3) Make a table on a piece of paper of the relevant variable names within the algorithm
4) Walk the first test case through the algorithm
5) Repeat the walk-through process using other test data
6) Check the expected result established in Step 2 matches the actual in Step 5

10

10

Sequence Control Structure:

Let us learn to design the program using the following example:

Example 1:

Write a Program that will prompt a user to input two integer values and display their sum, difference, product and quotient.

11

11

A Defining The Problem (Understanding Program):

- Any given program should be divided into three separate components as shown below:

Input	Processing	Output
- value1 - value 2	- prompt for value1 and value2 - Get value1 and value2 - Calculate sum, difference, product and quotient - Display sum, difference, product and quotient	- sum - difference - product - quotient

12

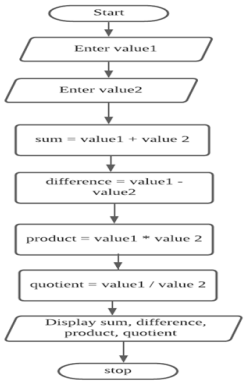
12

B Using Design Tools To Create Model

Solution Algorithm (Pseudo code)

Display_Results
1 prompt for value1, value 2
2 Get value1, value 2
3 sum = value1 + value2
4 difference = value1 - value2
5 product = value1 * value2
6 quotient = value1 / value2
7 Display sum, difference, product, quotient
END

Flowchart



The solution algorithm shows the steps of the statements in this program execute in the order that they appear, from the top of the program to the bottom.

13

13

C . Desk Checking (Testing)

- We need to choose simple test data based on the requirement of the program specification. To desk check the algorithm, only a few simple test cases are required to follow the major parts of the algorithm logic.
- **Input data:** You can choose any two sets of input data for the checking of the algorithm.

	First data set	Second data set
value1	10	-20
value2	5	7

(i) Expected Result : You can write your expected results from the program as shown below:

	First data set	Second data set
sum	15	-13
difference	5	-27
product	50	-140
quotient	2	-2

14

14

Desk Checking (Testing)

(iii) Desk check table

Statement No.	value1	value2	sum	difference	product	quotient
First Pass						
1, 2	10	5				
3			15			
4				5		
5					50	
6						2
7			display	display	display	display
Second pass						
1,2	-20	7				
3			-13			
4				-27		
5					-140	
6						-2
7			display	display	display	display

15

15

Desk Check Table

- Crate a table on a piece of paper with the relevant variable names used in the algorithm.
- Walk through the algorithm steps from 1 to 7 using the first test data set. This called First pass.
- Repeat the walk-through process using second test data set. This is called second Pass.
- Check the expected result matches with the actual result in the desk check table.
- If expected result and actual result are same, then the program is correct. Then the design part is completed.

16

16

The Selection Control Structure

- The condition in the IF statement is based on a comparison of two items and it is expressed with the following relational operations:
- There are a number of variation of the selection structure as follows:

1. Simple selection with null false branch (null ELSE statement)
2. Simple selection (simple IF statement)
3. Combined selection (combined IF statement)
4. Nested selection (nested IF statement)

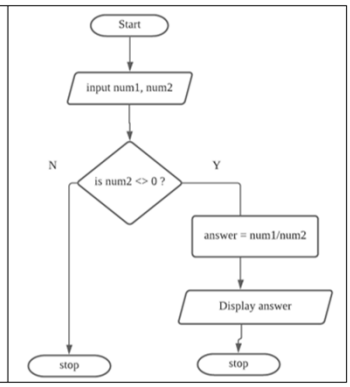
17

KAPLAN

17

Simple Selection With Null False Branch(null Else Statement)

1. Prompt for num1, num2
2. Get num1, num2
3. IF number <> 2 THEN
   answer = num1/num2
   ENDIF
4. Display answer
   END



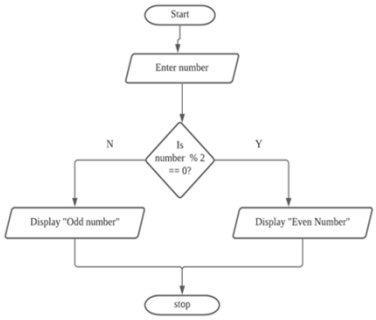
18

KAPLAN

18

Simple Selection (Simple IF Statement)

```
1. Prompt for number
2. Get number
3. IF(num %2 == 0) THEN
        Display "Even number"
   ELSE
        Display "ODD number"
   ENDIF
END
```



19

19

Combined Selection (Combined IF Statement)

- When you have multiple conditions to be checked we use AND/OR operator.

```
IF score < 0 OR score> 100 THEN
        Display "Invalid score"
End If
```

- Truth Table for AND & OR:

Condition 1	Condition 2	AND	OR
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

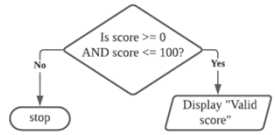
20

20

Combined Selection (Combined IF Statement)

- The AND operator takes two Boolean expressions as operands and creates a compound Boolean expression that is true when both of the sub expressions is true. The following is an example of an If-Then statement that uses the AND.

```
IF score >= 0 AND score <= 100 THEN
   Display "Valid Score"
ENDIF
```



21

21

Nested Selection (Nested If Statement)

- Nested selection can be classified as Linear nested IF and Non-linear nested IF statements.
- **Linear nested IF** statement is used when a field is being tested for various values and a different action is to be taken for each value.
- **Non-linear nested IF** occurs when a number of different conditions needs to be satisfied before a particular action can occur. It is termed non-linear because the ELSE statement may be separated from the IF statement with which it is paired.

```
IF record_code = 'A' THEN
        increment counter_A
    ELSE
        IF record_code = 'B' THEN
            increment counter_B
        ELSE
            increment error_counter
        ENDIF
    ENDIF
```

```
IF student_attendance = part_time THEN
    IF student_gender = female THEN
        IF student_age >21 THEN
            add 1 to mature_female_pt_students
        ELSE
            add 1 to young_female_pt_students
        ENDIF
    ELSE
        add 1 to male_pt_students
    ENDIF
ELSE
    add 1 to full_time_students
ENDIF
```

22

22

Algorithms Using Selection

Design an algorithm that prompt user to the weight of a parcel and calculate the shipping cost for that parcel and display the shipping charge. Charges are calculated as follows:

Parcel Weight (kg)	Shipping charge per kg ($)
<3.5 kg	$4.50 per kg
3.5 – 8 kg	$3.30 per kg
>8 kg	$2.75 per kg

23

23

A Defining Diagram

Input	Processing	Output
parcel_weight	Prompt for parcel_weight Get parcel_weight Calculate shipping_charge Display shipping_charge	shipping_charge

24

24

Solution Algorithm

- Note that the shipping charge for each parcel is determined as a certain cost, *per kilogram* of parcel weight.

```
    Calculate_shipping_charge
1     Prompt for parcel_weight
2     Get parcel_weight
3     IF parcel_weight < 3.5 THEN
        shipping_charge = 4.50 * parcel_weight
      ELSE
        IF parcel_weight >= 3.5 AND parcel_weight <= 8 THEN
          shipping_charge = 3.30 * parcel_weight
        ELSE
          shipping_charge = 2.75 * parcel_weight
        ENDIF
      ENDIF
4     Display shipping_charge
    END
```

25

25

C. *Desk checking*

Input data:

	First data set	Second data set
parcel_weight	2	4

Expected results:

	First data set	Second data set
shipping_charge	9.00	13.20

iii Desk check table:

Statement number	parcel_weight	shipping_charge
First pass		
1, 2	2	
3		9.00
4		display
Second pass		
1, 2	4	
3		13.20
4		display

26

26

Example 2:

Design an algorithm that will prompt user to input two numbers and an integer code. The value of the integer code should be 1, 2, 3 or 4. If the value of the code is 1, compute the sum of the two numbers. If the code is 2, compute the difference (first minus second). If the code is 3, compute the product of the two numbers. If the code is 4, compute the quotient (first divided by second). If the code is not equal to 1, 2, 3 or 4, display an error message. The program is then to display the two numbers, the integer code and computed result to the screen.

27

27

A. Defining Diagram

Processing	Input	Output
- Prompt for Num1, num2, Code - Get Num1, Num2, Code - Calculate Result - Display Num1, Num2, Code, Result	Num1 Num2 Code	Num1 Num2 Code Result msg

28

28

Solution Algorithm (Pseudocode)

```
Display_Result
    1.  Prompt for Num1, Num2, Code
    2.  Get Num1, Num2, Code
    3.  SET Result = 0, msg = blank
    4.  CASEOF(Code)
            1 :  Result = Num1 + Num2
            2 :  Result = Num1 - Num2
            3 :  Result = Num1 * Num2
            4 :  Result = Num1/  Num2
            Other : msg = 'Invalid Code'
        ENDCASE
    5.  IF (msg == blank) THEN
                Display Num1, Num2, Code, Result
        ELSE
                Display msg
        ENDIF
        END
```

29

29

C. Desk Checking

(i) Input data

	First Data set	Second data set	Third Data Set
Num1	45	-10	-5
Num2	20	24	-4
Code	1	2	7

(ii) Expected Result

	First Data set	Second data set	Third Data Set
Num1	45	-10	
Num2	20	24	
Code	1	2	
Result	65	-34	
msg			Invalid code

30

30

Desk Checking

(iii) **Desk check table**

Statement No	Num1	Num2	Code	Result	msg
First Pass					
1 , 2	45	20	1		
3				0	blank
4				65	
5	display	display	display	display	
Second Pass					
1, 2	-10	24	2		
3				0	blank
4				-34	
5	display	display	display	display	
Third Pass					
1, 2	-5	-4	7		
3				0	blank
4					Invalid Code
5					display

31

31

The End

32

32

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 4: repetition control structures

1

Topic 04 – Learning Outcomes

At the completion of this chapter, you would have learnt:

- To develop algorithms that use the DOWHILE and REPEAT…UNTIL control structures
- To introduce a pseudo code structure for counted repetition loops
- To develop algorithms using variations of the repetition construct

2 KAPLAN

2

Introduction

The iteration or repetition control structure allows certain part of the program to repeat until certain condition(s) is/are met.

This is also known as looping construct. In this chapter, we will look at two broad categories of loops: condition-controlled and count-controlled.

A *condition-controlled loop* uses a true/false condition to control the number of times that it repeats.

A *count-controlled loop* repeats a specific number of times

3

3

Repetition Using The DOWHILE Structure

- Three different ways that a set of instructions can be repeated:
  - -Condition tested at the beginning of the loop (leading decision loop)
  - -Condition tested the end of the loop (trailing decision loop)
  - -A counted number of times (counted loop)

4

4

Leading Decision Loop

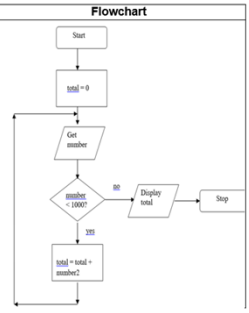
- DOWHILE construct is a leading decision loop – In this loop structure, the condition is tested before any statements are executed.

Pseudo code

```
1. SET total = 0
2. Prompt for number
3. GET number
4. DOWHILE number < 1000
5.         total = total + number
6.         Prompt for number

7.         GET number
     ENDDO
8. DISPLAY total
   END
```

Flowchart



The condition stated above is whether nu is mber less than 1000. If condition is true, i.e. number is less than 1000, then number accumulated as the total. After which, you get another number. The next execution is going back to the while condition to check if the condition is true, if yes, you accumulate the current number with the previous numbers and get another number.

Using DOWHILE To Repeat A Set Of Instruction An Unknown Number Of Times.

- When a trailer record or sentinel exists
  - Special record or value placed at the end of valid data to signify the end of that data
- When a trailer record or sentinel does not exist
  - End-of-file (EOF) marker is added when the file is created as the last character in the file

Counted Repetition

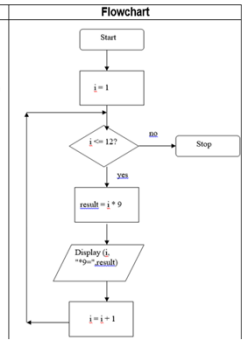
Counted loop iterates a specific number of times. The keyword DO is used for counter loop. The way that a count-controlled loop works is simple: the loop keeps a count of the number of times that it iterates, and when the count reaches a specified amount, the loop stops.

Pseudo code

```
1.  DO i = 1 TO 12
2.          result = i * 9
3.          Display (i, "*9=",result)

    ENDDO
```

Flowchart



7

KAPLAN

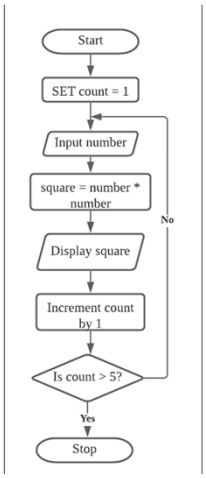
7

Repetition Using The REPEAT…UNTIL Structure

- Trailing decision loop
- REPEAT…UNTIL structure is similar to the DOWHILE structure (group of statements are repeated in accordance with specific condition). REPEAT…UNTIL structure is called trailing decision loop because it tests condition at the end of the loop.

```
1.  SET count =1
2.  REPEAT
3.          Prompt for number
4.          GET number
5.          Square = number * number
6.          Display square
7.          count = count + 1
8.  UNTIL (count >5)
     ENDDO

    END
```



8

KAPLAN

8

Algorithm Using Repetition

- Example 1: using DO…WHILE loop

Design an algorithm that will output the four times table, as follows:

4 x 1 = 4
4 x 2 = 4
4 x 3 = 12 . . .

4x 10 =40

9

9

A. Defining Diagram

Input	Processing	Output
x	- Calculate product - Display four times table	Four times table

B. Solution Algorithm (Pseudocode)

Display_FourTimesTable (DOWHILE loop)

```
1. SET x =1
2. DOWHILE (x <=10)
3.       Product = 4 * x
4.       Display  '4 * x = product '
5.       x = x + 1
   ENDDO
   END
```

10

10

C. Desk Checking

(i) Input data

	First Data set
x	1, 2, 3,……

(ii) Expected Result

First Data set
4 * 1 =4 4 * 2 =8 7 * 3 =12….

(iii) Desk check table

Statement No	x	product	DOWHILE	Output
First Pass				
1	1			
2			True	
3		4		
4				4* 1 = 4
5	2			
2			True	
3		8		
4				4 * 2 = 8
5	3			
2			True	
3		12		
4				4 * 3 =12
5	4			

11

KAPLAN

11

Example 2: Using REPEAT… Loop

Note: Repeat….until loop repeats when condition is false and stops when condition is true

B. Solution Algorithm (Pseudocode)

Display_FourTimesTable (REPEAT … UNTIL loop)

```
1. SET x =1
   REPEAT
2.     Product = 4 * x
3.     Display '4 * x = product '
4.     x = x + 1
5. UNTIL (x > 10)
   END
```

Statement No	x	product	condition	Output
First Pass				
1	1			
2		4		
3				4* 1 = 4
4	2			
5			False	
2		8		
3				4 * 2 = 8
4	3			
5			False	
2		12		
3				4 * 3 =12
4	4			
…	…	…	…	….
4	10			
5			False	
2		40		
3				4 * 10 = 40
4	11			
5			True	

12

KAPLAN

12

Example 3: Using DO … ENDDO Loop

B. Solution Algorithm (Pseudocode)

Display_FourTimesTable (DO … ENDDO loop)

```
1.  DO x = 1 to 10
2.      Product = 4 * x
3.      Display  '4 * x = product '
    ENDDO
    END
```

Statement No	x	product	Output
First Pass			
1	1		
2		4	
3			4 * 1 = 4
1	2		
2		8	
3			4 * 2 = 8
1	3		
2		12	
3			4 * 3 =12
1	4		
…	…	…	….
1	10		
2		40	
3			4 * 10 = 40

13

13

The End

14

14

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 5: array processing

1

Topic 05 – Learning Outcomes

- To introduce arrays and the uses of arrays
- To develop pseudocode algorithms for common operations on arrays
- To illustrate the manipulation of single- and two-dimensional arrays

2 KAPLAN

2

Introduction

- An array allows you to store a group of items of the same data type together in memory.
- Processing a large number of items in an array is usually easier than. Arrays are one of the most powerful programming tools available.
- This chapter covers an array data structure that is made up of a number of variables all of which have the same data type.
- Processing a large number of items in an array is usually easier than processing a large number of items stored in separate variables.

3

3

Array Processing

- An array is a named storage location in memory.
- Unlike a variable, an array can hold a group of values.
- All of the values in an array must be the same data type.
- You can have an array of Integers, an array of characters, or an array of Strings, but you cannot store a mixture of data types in an array.
- The following example shows how we will declare an array in pseudocode:

```
Student_scores [10]
```

4

4

Array Processing

- The most typical operations performed on arrays are:
  - -Loading a set of initial values into the element of an array
  - -Processing the elements of an array
  - -Searching an array, using a linear or binary search, for a particular element
  - -Writing out the contents of an array to a report

5

KAPLAN

5

Array Processing

- The following example shows the simple manipulation of an array using DO and DOWHHILE loop
- In this example, each the sum of all the elements stored in the array. When all elements have been added, the variable sum is printed.

DO …. ENDO loop	DOWHILE…..END loop
Find_sum_of_elements 1. Set sum to zero 2. DO index = 1 to number_of_elements 3. sum = sum + array (index) ENDDO 4. Print sum END	1. Set sum to zero 2. Set index to 1 3. DOWHILE (index < = 5) 4. sum = sum + array (index) 5. index = index + 1 ENDDO END

6

KAPLAN

6

Initialising The Elements Of An Array

- Because an array is an internal data structure, initial values must be placed into the array before any information can be retrieved from it.
- These initial values can be assigned to the elements of the array as constants, or they can be read into the array from a file.

```
Read_values_into_variable_array
    1. Set  max_num_elements to required value
    2. Set index to zero
    3. Read first input value
    4. DOWHILE (input value NOT = 9999) AND (index <  max_num_elements)
    5.     index = index + 1
    6.     array (index) = input value
    7.     Read next input value
    8. ENDDO
    9. IF index < max_num_elements THEN
            index = index + 1

            array (index) = 9999

       ELSE

            Print 'Array size too small'

       ENDIF
       END
```

7

7

Paired Arrays

- Two arrays that have the same number of elements.
- The arrays are paired because the elements in the first array correspond to the elements in the same position in the second array.

8

8

Searching An Array

- The below flow chart shows the general logic of the sequential search algorithm. Here is a summary of the data items in the figure:
  - Array is the array being searched.
  - *searchValue* is the value that the algorithm is searching for.
  - Found is a Boolean variable that is used as a flag. Setting found to False indicates that *searchValue* has not been found. Setting found to True indicates that *searchValue* has been found.
  - index is an Integer variable used as a loop counter.

9

9

The Following Shows How You Can Express This Logic In Pseudo Code:

```
SET found = False
SET index = 0
DOWHILE found == False AND index <= SIZE – 1
   IF array[index] == searchValue THEN
         SET found = True
   ELSE
         SET index = index + 1
  ENDIF
ENDDO
```

10

10

Two-dimensional Arrays

- An element of a two-dimensional array is specified using the name of the array, followed by two subscript, enclosed in parentheses, separated by a comma.
- The row subscript is specified first, followed by the column subscript.

	Column 0	Column 1	Column 2	Column 3
Row 0				
Row 1				
Row 2				

11

11

Declaring A Two-dimensional Array

- To declare a two-dimensional array, two size declarators are required:
- The first one is for the number of rows and the second one is for the number of columns.
- The following pseudocode shows an example of how to declare a two-dimensional array:

  Integer_values[3][4]
- This statement declares a two-dimensional Integer array with three rows and four columns.
- The name of the array is values, and there are a total of twelve elements in the array.
- As with one-dimensional arrays, it is best to use named constants as the size declarators.

12

12

Here Is An Example

- ROWS = 3
- COLS = 4
- Integer_values[ROWS][COLS]
- Figure below illustrates the array with the subscripts shown for each element.

	Column 0	Column 1	Column 2	Column 3
Row 0	values[0][0]	values[0][1]	values[0][2]	values[0][3]
Row 1	values[1][0]	values[1][1]	values[1][2]	values[1][3]
Row 2	values[2][0]	values[2][1]	values[2][2]	values[2][3]

13

13

The End

14

14

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 06 – Introduction To Modularisation

1

Objectives

- To introduce modularisation as a means of dividing a problem into subtasks
- To present hierarchy charts as a pictorial representation of modular program structure
- To develop programming examples that use a simple modularised structure

2 KAPLAN

2

Introduction to Modularization

- Modularisation is defined as dividing a problem into separate units called modules.
- Each unit perform single task. Top-down design methodology allows the programmer to focus on the overall design of the problem without getting too involved with the details of the lower-level modules.
- The concept of modules make the algorithm simpler, more systematic and more likely to be error free.

3

3

Module

- A module is a single unit consist of group of statements that exist within a program for performing specific task.

4

4

Modules in Payroll application

Let us consider a payroll application. This application's overall task is calculating an employee's pay would consist of several subtasks as shown below:

- Getting the employee's hourly rate
- Getting the hours worked details of employee
- Calculating the gross pay
- Calculating overtime pay
- Calculating taxes and benefits
- Calculating the net pay
- Printing the pay check

5

5

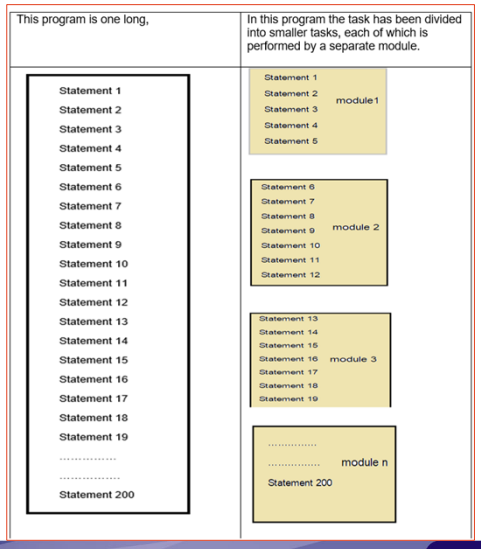
Modularization

- This approach is sometimes called *divide and conquer* because a large task is divided into many smaller tasks that are easily performed.
- Figure below shows this by comparing two programs:
- one that uses a long a large program as one long sequence of statements, to perform a task, and
- another one that divides the long sequence of statements into smaller tasks, each of which are performed by a separate module.

6

6

Modularization



7

7

Modularization approach

- When using modules in a program, we generally separate each task within the program in its own module.
- In payroll application example, a pay calculating program might have the modules such as pay rate, worked hours, calculate the gross pay, overtime pay, calculating taxes, calculating net pay and printing the pay check.

8

8

Benefits of Using Modules

Simpler Code

- Modularization makes the programming codes to be simpler and easier to understand.
- Breaking the larger program into many small modules makes it easier to read than one long sequence of codes.

9

9

Benefits of Using Modules

Code Reuse

- Modularization reduce the duplication of code within a program.
- If a certain operation is performed in many places in a program, a module can be written once to perform that operation, and then be called any time it is needed.
- This advantage of using modules is known as *code reuse* because we are writing the code once and then reusing the module again and again whenever we need to perform the task.

10

10

Benefits of Using Modules

Better Testing

- Modularization makes the testing and debugging of the code easier.
- Modularization allows programmers to test each module in a program individually, when it is developed.
- Module testing can be done earlier stage of the development in order to determine whether the module correctly performs its operation.
- Modularization makes it easier to isolate each module and fix errors.

11

11

Benefits of Using Modules

Easier Facilitation of Teamwork

- Modularization allows programmers to work in teams.
- When a program is developed as a set of modules that each performs an individual task, then different programmers can be assigned the job of writing different modules.

12

12

Top-Down Design

The process of top-down design is performed in the following manner:

- The main task of the program is to perform is broken down into a series of subtasks.
- Each of the subtasks is checked to determine whether it can be further broken down into more subtasks. This step is repeated until no more break down of subtasks.
- Once all of the subtasks have been identified, code must be written for each subtask.
- This process is called top-down design because the programmer begins by looking at
- the top level of tasks that must be performed, and then breaks down those tasks into lower levels of subtasks.

13

13

Hierarchy Charts

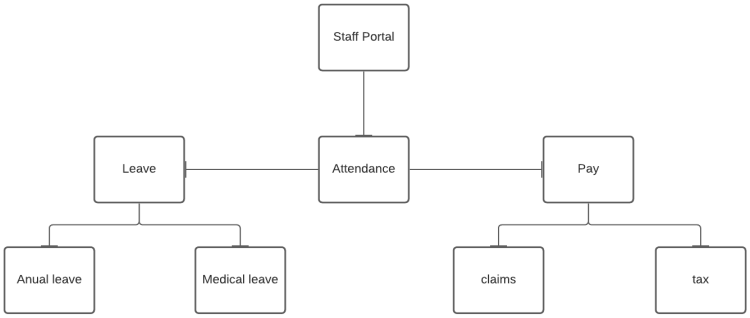
- Flowcharts are good tools for graphically representing the flow of logic inside a module, but flowchart do not give a visual representation of the relationships between modules.
- A hierarchy chart, which is also known as a *structure chart*, uses rectangular boxes to represent each module in a program.
- The rectangles are connected in a way that illustrates their relationship to one another.
- Figure below shows an example of a hierarchy chart of staff portal.

14

14

14

Hierarchy Charts



Staff portal is a cloud portal where the employees have access to their own attendance, leave details, pay slip, tax & their claims.

Employees can clock in daily Attendance, apply leave, apply payroll claims, and view their payslip and tax.

15

15

Modularisation

- The use of modules make the algorithm simpler, more systematic and more likely to be free of errors
- Modularisation is the process of dividing a problem into separate tasks, each with a single purpose
- Top-down design methodology allows the programmer to concentrate on the overall design of the algorithm without getting too involved with the details of the lower-level modules

16

16

Modularisation

- Benefits of modular design

  -Ease of understanding: each module should perform just one function

  -Reusable code: modules used in one algorithm can be used in other algorithms

  -Elimination of redundancy: using modules can help to avoid the repetition of writing out the same segment of code more than once

  -Efficiency of maintenance: each module should be self-contained and have little or no effect on other modules within the algorithm

17

17

Steps in modularisation

1. Define the problem by dividing into its three components (input, output and processing)
2. Group the activities in subtask or functions to determine the modules that will make up the program
3. Construct a hierarchy chart to illustrate the modules and their relationship to each other

18

18

Steps in modularisation

4. Establish the logic of the mainline of the algorithm in pseudocode
5. Develop the pseudocode for each successive module in the hierarchy chart
6. Desk check the solution algorithm

19

19

References

- Robertson, L. (2007). *Simple program design* (5th ed.). Australia: Course Technology.

20

20

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 7: Introduction To Object-oriented Programming

1

Topic 07 – Learning Outcomes

- To introduce object-oriented design
- To define objects, classes, attributes, operations and information hiding
- To list the steps required to create an object-oriented design solution to a problem
- Discuss the evolution of programming languages.
- Discuss the fundamental concepts of object-oriented programming
- Tell the difference between syntax errors, run-time errors, and logic errors.

2 KAPLAN

2

Introduction

- There are two methods of programming used namely procedural and object oriented.
- Procedural programming is the earliest programming language, meaning a program was made of one or more procedures.
- Object-oriented programming (OOP) focuses on an application's data and the methods need to manipulate that data.
- Methods in object-oriented programs continue to use sequence, selection, and looping structures and make use of arrays.
- An *object-oriented programming* (*OOP*) which is focusing on creating objects.
- An *object* is a software entity that contains both data and procedures.

3

KAPLAN

3

Generations Of Programming Languages

- **Generation 1—machine languages:** Program data entered directly into RAM in form of 1s and 0s
- **Generation 2—assembly languages:** Mnemonic symbols represent instructions and data.
  - One-to-one correspondence with machine-language instructions
    - **Assembler:** Translates to machine language
    - **Loader:** Loads machine language into memory
- **Generation 3—high-level languages:** Designed to be easy to write, read, and manipulate.
    - C, C++, Java
    - High-level instructions may represent many machine-language instructions.
    - **Compiler** translates high-level language into machine language.
  - Two major groups of high-level languages:
    - **Procedural languages:** COBOL, FORTRAN, BASIC, C, Pascal
    - **Object-oriented languages:** Smalltalk, C++, Java

4

KAPLAN

4

Object-oriented Programming (OOP)

- **Classes and Objects**
- In objected oriented Programming, a **class** is a term that describes a group or collection of objects with common attributes.
- An **object** is one **instance** of a class.
- A class is like a blueprint from which many houses might be constructed, or like a recipe from which many meals can be prepared.
- One house and one meal are each an instance of their class.

5

KAPLAN

5

Object

- The data contained in an object is known as the *object's fields.*
- An object's fields are simply variables, arrays, or other data structures that are stored in the object.
- The procedures that an object performs are known as *methods*.
- An object's data is stored in attributes.
- Attributes are the properties or characteristics that describe that particular object.

Example of an object: Alarm clock

Alarm clock has the following fields called attributes:

- second (a value in the range of 0–59)
- minute (a value in the range of 0–59)
- hour (a value in the range of 1–12)
- alarm set time
- alarm state : on or off ("on" or "off")

To change object's value, we must use one of the methods.

The following are some of the alarm clock's methods:

- Set time
- Set alarm time
- Turn alarm on
- Turn alarm off

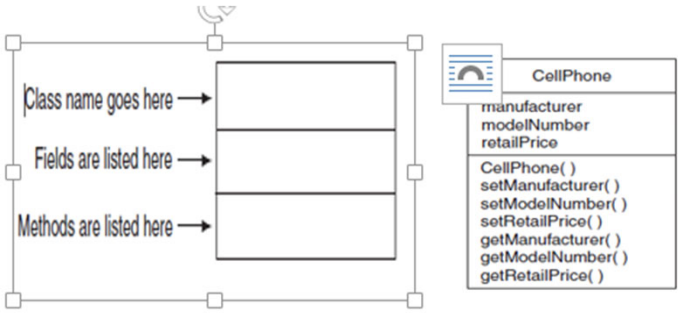
6

KAPLAN

6

Class Diagram

- Object-oriented programming requires classes, attributes and methods to be represented in a class diagram.
- Consists of a rectangular box divided into three section with the name of the class at the top, the attributes of the class in the middle and the methods at the bottom.



7

7

Inheritance

- Inheritance inolves a superclass and a subclass.
- The *superclass* is the general class and the *subclass* is the specialized class.
- The subclass inherits the fields and methods from the superclass without any of them having to be rewritten.

8

8

Polymorphism

- Polymorphism allows you to create methods with the same name in different classes (that are related through inheritance) and gives you the ability to call the correct method depending on the type of object that is used to call it.
- The term *polymorphism* refers to an object's ability to take different forms. It is a powerful feature of object-oriented programming.

9

9

Encapsulation

- OOP addresses the problem of code/data separation through encapsulation and data hiding.
- Encapsulation refers to the combining of data and code into a single object.
- Data hiding refers to an object's ability to hide its data from code that is outside the object.
- Only the object's methods may then directly access and make changes to the object's data.
- An object typically hides its data, but allows outside code to access its methods.

10

10

Access Modifiers

- An access modifier restricts the access of a class, constructor, data member and method in another class.
- This concept is called 'visibility'. In java we have four access modifiers:

11

11

Java Uses Four Types Of Access Modifiers:

- **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected:** Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package. You can also say that the protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes. This access modifier is generally used in a parent child relationship.
- **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

12

12

Steps In Creating An Object-oriented Solution

- Three steps in creating an object-oriented solution for a problem with just one class:
- Identify the objects and their attributes, responsibilities and operations
- Design the algorithms for the operations and methods using structured design
- Develop a test or driver algorithm to test the solution

13

KAPLAN

13

Introduction To Programming

- **Language Elements**
  - -Every language, including Java has the following elements:
  - -Vocabulary: Set of all of the words and symbols in the language
  - -Syntax: Rules for combining words into sentences (*statements*)
  - -Semantics: Rules for interpreting the meaning of statements

- Programming languages have small vocabularies and simple syntax and semantics.
  - -Programming language syntax must be absolutely correct.
  - -Programming language statements are interpreted literally.
  - -Every detail must be present.

14

KAPLAN

14

Introduction To Programming

- Java uses two categories of data types:
- 1. **Primitive data types**: integer, float, double, character, boolean
- 2. **Complex data types:** object, string, arrays

Variable declaration statement:

- Declares the identifier and data type for a variable
  - -int age; (declares one int variable)
  - -int a, b, c; (declares three int variables)
  - -double d = 2.45; (declares and initializes a variable)
- Constants are variables whose value cannot change.
  - -final double PI = 3.14;

15

15

Comments

- Explanatory sentences inserted in a program. Compiler ignores them. Purpose is to make program more readable.
- Two varieties:
  - -**End of line comments:** All text following a double slash (//) on a single line
  - -**Multiline comments:** All text occurring between a /* and a */

16

16

Programming Errors

- **Syntax errors**: These error occurs when a syntax rule is violated. These errors are detected during compilation. Compiler helps identify the syntax error.
- **Run-time errors:** These error **o**ccurs during program execution. Example is Dividing by 0 error. This will be detected only when program runs. JVM indicates type of error and location.
- **Logic errors (design errors** or **bugs):** These error occurs when incorrect logic implemented in the program. Code may be correct in every other way, but does not do what it is supposed to do. Need to thoroughly test and **debug** the program when an error is found.

17 KAPLAN

17

THE END

18 KAPLAN

18

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 8: Introduction To JAVA Programming

1

Topic 08 – Learning Outcomes

- Discuss why Java is an important programming language.
- Explain the Java virtual machine and byte code.
- Choose a user interface style.
- Describe the structure of a simple Java program.
- Edit, compile, and run a program using a Java development environment.

2 KAPLAN

2

Introduction

- James Gosling developed Java at Sun Microsystems in 1995.
- It was created with the idea to write code one time and run it on any operating system, hence the term "**Write Once, Run Anywhere**."
- Java can be described as a "simple, object-oriented, multi-threaded, garbage-collected, secure, robust, architecture-neutral, portable, high performance, dynamic language."
- Soon after its introduction, WWW browsers updated to include applets, known as small Java programs, in HTML documents.
- First through applets, and later through complete Internet applications, Java has become widespread on the Internet.
- Its built-in support for URLs and the HTTP protocol allows Java applications to access standard WWW servers.
- These Java features have improved the feasibility of the development of Internet hypermedia that does not rely on a WWW browser for a hypermedia client.

3

KAPLAN

3

Why Java?

- **Secure:** Enables construction of virus-free, tamper-free systems.
- **Robust:** Supports development of programs that do not overwrite memory.
- **Portable:** Programs can be run on different types of computers without change.
  - -Java is well-suited for distributed, network-based applications.
  - -Java supports **threads**.
    - -A process that can run concurrently with other processes
- Java resembles C++. Easy for C++ programmers to learn Java. Java runs more slowly than other languages.

4

KAPLAN

4

Why Java?

- The Java compiler translates Java source code into Java **byte code** - A pseudo-machine language. Byte code is executed by the **Java Virtual Machine (JVM)**.
- **Interpreter:** A program that behaves like a computer. Any computer with the JVM installed can execute Java byte code (portability).
- Some JVMs support **just-in-time compilation (JIT)**. Translate byte code instructions into machine language when they are first encountered
- **Java applets:** Java programs that run in a Web browser. A JVM is incorporated into the browser.

5

5

How To Write, Compile, And Execute Our First Java Program.

Step1. Go to NetBeans, and write the following code.

Program1.java

```
public class Program1
{
    public static void main(String[] args)
    {
        System.out.println("Welcome to Java Program");
    }
}
```

Save it as Program1.java, remember the class name and the file name should be the same.

Step2. Run the file.

It will show the following output:

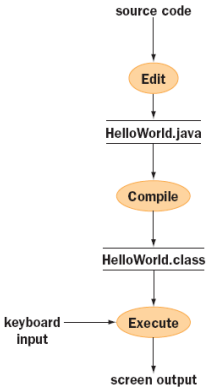
```
run:
Welcome to Java Program
BUILD SUCCESSFUL (total time: 0 seconds)
```

6

6

How To Write, Compile, And Execute Our First Java Program

- The following figure shows that how the source code is compiled and executed.
- The file extension for a Java source code file is .java. The file extension for a Java class file is .class.



7

7

Primitive Data Types

- There are eight primitive data types - byte, short, int, long, float, double, char and boolean,
- byte, short, int, long involve whole numbers without any decimals
- whereas float, double contains decimal values.
- boolean contains TRUE or FALSE value. Char contains only one single character.

Type	Size	Range
byte	8 bits	-128 .. 127
short	16 bits	-32,768 .. 32,767
int	32 bits	-2,147,483,648 .. 2,147,483,647
long	64 bits	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807
float	32 bits	$3.40282347 \times 10^{38}$, $1.40239846 \times 10^{-45}$
double	64 bits	$1.7976931348623157 \times 10^{308}$, $4.9406564584124654 \times 10^{-324}$

8

8

Example Of Using Different Data In Java Program:

```java
public class Program2
{
    public static void main(String[] args)
    {
        int a = 10;
        float b = 10.50f;
        char c = 'a';
        boolean d = false;
        System.out.println("the int value is : "+a);
        System.out.println("the float value is : "+b);
        System.out.println("the char value is : "+c);
        System.out.println("the boolean value is : "+d);
    }
}
```

Output:

```
run:
the int value is : 10
the float value is : 10.5
the char value is : a
the boolean value is : false
BUILD SUCCESSFUL (total time: 0 seconds)
```

9

9

Arithmetic Operators

- Given: A=10, B=4

Operator	Description	Example
+	Addition, adds two values	A + B=14
-	Subtract A and B	A - B=6
*	Multiplication, multiplies two values.	A * B = 40
/	Division, divides left side value by right side value	A / B = 2
%	Modulo returns remainder after division.	A % B = 2
++	Increases the value by 1.	A++ = 11
--	Decreases the value by1.	B-- = 3

10

10

Relational Operators

- Given: A=2, B=1

Operator	Description	Example
==	Returns true if two values are equal else false	(A == B) is false
!=	Returns true if two values are not equal else false	(A != B) is true
>	Returns true if left side value is strictly greater than right side value else false.	(A > B) is true
<	Returns true if left side value is strictly lesser than right side value else false.	(A < B) is false
>=	Returns true if left side value is greater than or equal to right side value else false.	(A >= B) is true
<=	Returns true if left side value is lesser than or equal to right side value else false.	(A <= B) is false

11

11

Logical Operators

- Given A=0 and B=1

Operator	Description	Example
&&	Return true if both values are non-zero else false	A && B= false
||	Return true if any of the two values is non zero else false	A || B=true
!	Return True if the value is zero else false	!A = true

12

12

Example

Example : convert the following pseudocode into java program:

1. Set test1 = 88.0
2. Set test2 = 92.5
3. Set test3 = 97.0
4. Average = (test1 + test2 + test3) / 3
5. Display Average

   END

- In pseudocode we don't declare the variable as its design of the program.
- In actual development of the program, all the variables used in the program must be declared before it is used.
- The below java program shows how to declare the variable before it is used.

13

13

Java Program

- The below java program calculates the average of three test score. When we convert the pseudocode to java program,

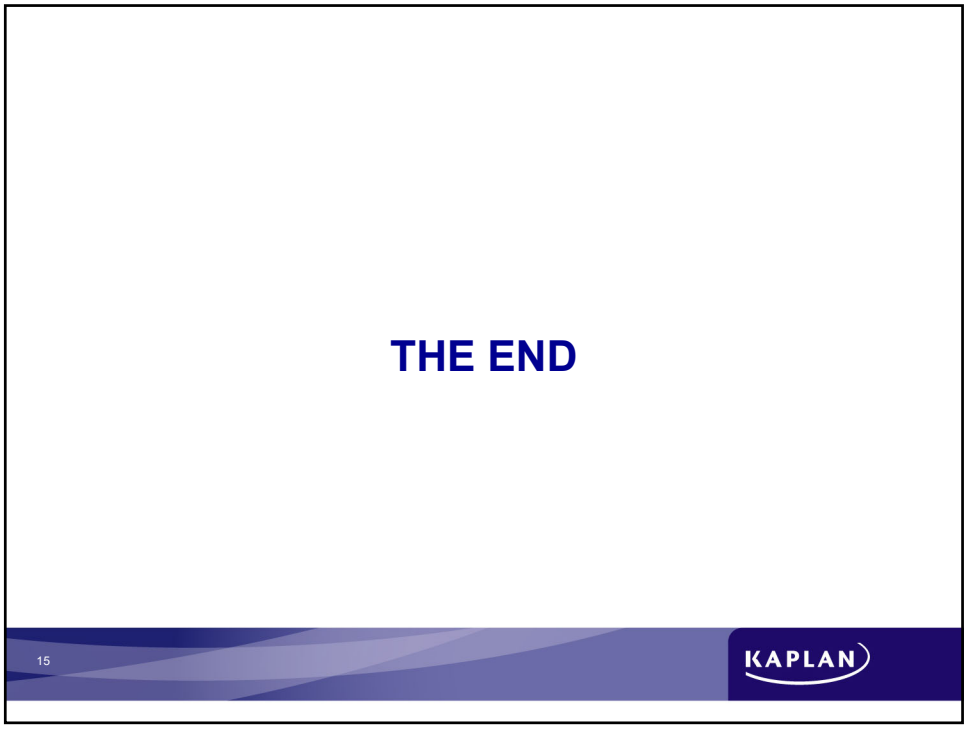
```
public class Program3
{
    public static void main(String[] args)
    {
        double test1, test2, test3, Average;
        test1 = 88.0;
        test2 =92.5;
        test3 = 97.0;
        Average = (test1 + test2 + test3)/3;
        System.out.println("Average : "+Average);
    }
}
```

Output:

```
Output - LessonICS (run)
run:
Average : 92.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

14

14



15

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 9: introduction to selection control

1

Topic 09 – Learning Outcomes

- To convert the algorithm to appropriate java program
- Use if and if-else statements to make choices.
- Construct appropriate conditions for control statements sing relational operators.

2 KAPLAN

2

Introduction

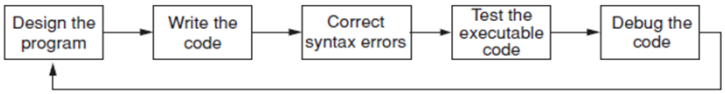
- Statements in Java can be either simple statements or compound statements.
- Simple statements, such as assignment statements and subroutine call statements, are the basic building blocks of a program.
- Compound statements, such as while loops and if statements, are used to organize simple statements into complex structures, which are called control structures because they control the order in which the statements are executed.
- Control statements are the statements that define the flow of programs.
- There are 3 types of control statements in Java: Selection, iteration, and jump statements.
- In this topic, we are going to convert all the selection pseudocode exercises we have done in the topic 4 to java programs.

3

KAPLAN

3

The Program Development Cycle



- Programs must be carefully designed before they are written.
- During the design process, programmers use tools such as pseudocode and flowcharts to create models of programs.
- After designing the program, the programmer begins writing code in a high-level language.
- Virtually all code contains syntax errors when it is first written, so the programmer will typically spend some time correcting these.
- Once all of the syntax errors and simple typing mistakes have been corrected, the program can be compiled and translated into a machine language program (or executed by an interpreter, depending on the language being used).

4

KAPLAN

4

Let Us Convert All The Pseudocode Example To Java Program;

- **Example1: Simple Selection with null false branch (null Else statement)**

1. Prompt for num1, num2
2. Get num1, num2
3. IF number <> 2 THEN
   answer = num1/num2
   ENDIF
4. Display answer
   END

```
run:
Enter number 1 : 25
Enter number 2 : 3
answer = 8
BUILD SUCCESSFUL (total time: 7 seconds)
```

Java Program for the above pseudocode:

```java
import java.util.Scanner;
public class Example1
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number1, number2, answer; // declaration
        System.out.print("Enter number 1 : ");
        number1 = keyboard.nextInt();
        System.out.print("Enter number 2 : ");
        number2 = keyboard.nextInt();
        if(number2 != 0)
        {
            answer = number1/number2;
            System.out.println("answer = " +answer);
        }
    }
}
```

5

KAPLAN

5

Example2: Simple Selection (Simple IF Statement)

Pseudo code

1. Prompt for number1
2. GET number1
3. Prompt for number2
4. GET number2
5. IF number2 = 0 THEN
   DISPLAY "Division by zero error"
   ELSE
   answer = number1/number2
   DISPLAY answer
   ENDIF

END

Output for the condition true:

```
run:
Enter number 1 : 12
Enter number 2 : 3
answer = 4
BUILD SUCCESSFUL (total time: 5 seconds)
```

Java program:

```java
import java.util.Scanner;
public class Example1
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number1, number2, answer; // declaration
        System.out.print("Enter number 1 : ");
        number1 = keyboard.nextInt();
        System.out.print("Enter number 2 : ");
        number2 = keyboard.nextInt();
        if(number2 == 0)
        {
            System.out.println("Division by zero error");
        }
        else
        {
            answer = number1/number2;
            System.out.println("answer = " +answer);
        }
    }
}
```

6

KAPLAN

6

Example 3: Nested If Selection Structure

B Solution algorithm

Note that the shipping charge for each parcel is determined as a certain cost, *per kilogram* of parcel weight.

```
    Calculate_shipping_charge
1     Prompt for parcel_weight
2     Get parcel_weight
3     IF parcel_weight < 3.5 THEN
        shipping_charge = 4.50 * parcel_weight
      ELSE
        IF parcel_weight >= 3.5 AND parcel_weight <= 8 THEN
          shipping_charge = 3.30 * parcel_weight
        ELSE
          shipping_charge = 2.75 * parcel_weight
        ENDIF
      ENDIF
4     Display shipping_charge
    END
```

7

7

Example 3: Nested If Selection Structure

Java program:

```java
import java.util.Scanner;
public class Example3
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        double parcel_weight, shipping_charge; // declaration
        System.out.print("Enter parcel weight : ");
        parcel_weight = keyboard.nextDouble();
        if(parcel_weight < 3.5)
        {
            shipping_charge = parcel_weight * 4.50;
        }
        else if(parcel_weight >= 3.5 && parcel_weight <=8)
        {
            shipping_charge = parcel_weight * 3.30;
        }
        else
        {
            shipping_charge = parcel_weight * 2.75;
        }
       System.out.printf("Shipping charge : $%.2f\n", shipping_charge);
    }
}
```

8

8

- If we follow the flow of execution, we see that the condition parcel weight < 4.5 is tested. If the condition is true, then it will calculate the shipping charge and display the display charge. However, if the condition is false, it will test the next condition and this condition is false then it will calculate the shipping charge of last option and display shipping charge.
- After completing the program we need to test the program for the test data which we had tested in desk checking algorithm (topic 3 – page 13) shown below:

C Desk checking

i Input data:

	First data set	Second data set
parcel_weight	2	4

ii Expected results:

	First data set	Second data set
shipping_charge	9.00	13.20

First data set	Second data set
Output - LessonC3 (run) run: Enter parcel weight : 2 Shipping charge : $9.00 BUILD SUCCESSFUL (total time: 11 seconds)	Output - LessonC3 (run) run: Enter parcel weight : 4 Shipping charge : $13.20 BUILD SUCCESSFUL (total time: 2 seconds)

9

9

Example 4: Switch…case Structure

- Let us try to write java program for the following pseudocode that uses case of structure.

B. Solution Algorithm (Pseudocode)

```
Display_Result
    1.  Prompt for Num1, Num2, Code
    2.  Get Num1, Num2, Code
    3.  SET Result = 0, msg = blank
    4.  CASEOF(Code)
            1 :  Result = Num1 + Num2
            2 :  Result = Num1 - Num2
            3 :  Result = Num1 * Num2
            4 :  Result = Num1/  Num2
            Other : msg = 'Invalid Code'
        ENDCASE
    5.  IF (msg == blank) THEN
                Display Num1, Num2, Code, Result
        ELSE
                Display msg
        ENDIF
        END
```

10

10

Java Program

```
import java.util.Scanner;
public class Example4
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int Num1, Num2, Code, Result = 0; // declaration
        String msg=null;
        System.out.print("Enter number 1 : ");
        Num1 = keyboard.nextInt();
        System.out.print("Enter number 2 : ");
        Num2 = keyboard.nextInt();
        System.out.print("Enter code : ");
        Code = keyboard.nextInt();
        switch(Code)
        {
            case 1:
                Result = Num1 + Num2;
            break;
            case 2:
                Result = Num1 - Num2;
            break;
            case 3:
                Result = Num1 * Num2;
            break;
           case 4:
                Result = Num1 / Num2;
            break;
            default:
                msg = "Invalid code";
            break;
        }
        if(msg == null)
        {
            System.out.printf("Number 1 = %d \n", Num1);
            System.out.printf("Number 2 = %d \n", Num2);
            System.out.printf("Code     = %d \n", Code);
            System.out.printf("Result   = %d \n", Result);
        }
        else
        {
            System.out.printf(msg);
        }

    }
}
```

11

11

Java Program

- The *case structure* is a *multiple alternative decision structure*.
- It allows you to test the value of a variable or an expression and then use that value to determine which statement or set of statements to execute.
- The above java program shows an example of how a case structure written in java.
- If the code is 1, addition is executed. If code is 2, subtraction is executed is executed. If the code is 3, multiplication is executed.
- If the code is 4, division is executed. If the code contains none of these values, the statement labelled default is executed.
- In this case, the statement sets message to Invalid code.

12

12

Java Program : Testing

- After writing the program, we test the program using the data we tested in desk checking:

C. Desk checking

(i) Input data

	First Data set	Second data set	Third Data Set
Num1	45	-10	-5
Num2	20	24	-4
Code	1	2	7

(ii) Expected Result

	First Data set	Second data set	Third Data Set
Num1	45	-10	
Num2	20	24	
Code	1	2	
Result	65	-34	
msg			Invalid code

13

KAPLAN

13

Java Program: Testing

- **Java program output:**

First data set	Second data set	Third data set
run: Enter number 1 : 45 Enter number 2 : 20 Enter code : 1 Number 1 = 45 Number 2 = 20 Code = 1 Result = 65 BUILD SUCCESSFUL (total	run: Enter number 1 : -10 Enter number 2 : 24 Enter code : 2 Number 1 = -10 Number 2 = 24 Code = 2 Result = -34 BUILD SUCCESSFUL (tota	run: Enter number 1 : -5 Enter number 2 : -4 Enter code : 7 Invalid codeBUILD SUCC

14

KAPLAN

14

THE END

15

KAPLAN

15

KAPLAN

INTRODUCTION TO COMPUTER SCIENCE

Topic 10: Introduction To Repetition Control

1

Topic 10 – Learning Outcomes

- To convert the algorithm to appropriate java program
- Use while and for and do.. while loops to repeat a process.
- Construct appropriate conditions for control statements using relational operators.
- Detect and correct common loop errors.

2 KAPLAN

2

Introduction

- When we use a loop, we can write one set of instructions that operates on multiple, separate sets of data.
- Recall the loop structure that you learned about in Chapter 4; As long as a Boolean expression remains true, a while loop's body executes.
- We have already designed many programs use a loop to control structure. In this chapter, we will learn how to convert the algorithm into java program.

3

KAPLAN

3

Loop Design Strategies

- There are several steps that we can follow when it comes to loop designs. To properly understand a loop, we must consider
  - -pre-condition
  - -post-condition
  - -loop variant
  - -loop invariant

4

KAPLAN

4

Example1: Pseudocode - DOWHILE structure

- Let us convert all the pseudocode example to Java program;

```
1.  SET total = 0
2.  Prompt for number
3.  GET number
4.  DOWHILE number < 1000
5.          total = total + number
6.          Prompt for number

7.          GET number
      ENDDO
8.  DISPLAY total
     END
```

5

5

Java Program:

- Pseudocode DOWHILE structure is while loop in java program.
- The program keep prompting user to input number.
- If the number is below 1000, it will calculate the total and prompt for user to input the number. Once the user input the number above 1000, the loop terminates and displays the total.
- Sometimes, loop control may need to be based on the value of what we are processing.
- In this case, we would use a sentinel-controlled loop.
- Sentinel-controlled loop is sometimes called indefinite repetition because it is not known in advance how many times the loop will be executed

6

6

Java Program:

```java
import java.util.Scanner;
public class Example_1
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number, total=0; // declaration
        System.out.print("Enter number  : ");
        number = keyboard.nextInt();
        while(number < 1000)
        {
            total = total + number;
            System.out.print("Enter number  : ");
            number = keyboard.nextInt();
        }
        System.out.println("Total = " +total);
    }
}
```

Output:

```
run:
Enter number  : 10
Enter number  : 234
Enter number  : 589
Enter number  : 789
Enter number  : 321
Enter number  : 1020
Total = 1943
BUILD SUCCESSFUL (total time: 20 seconds)
```

7

7

Example2: Pseudocode - Counted loop structure

- Counted repetition construct will perform the initialising, incrementing and testing of the loop counter automatically. It will also terminate the loop once the require number of repetition has been executed.

```
1.  DO i = 1 TO 12
2.          result = i * 9
3.          Display (i, "*9=",result)

    ENDDO
```

8

8

Java Program:

- Pseudocode DO... ENDO structure is called for loop in

```
import java.util.Scanner;
public class Example_2
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int i, result; // declaration
        for(i=1; i<=12; i ++)
        {
            result = i * 9;
            System.out.println(i + " * 9  = " +result);
        }

    }
}
```

Output:

```
run:
1 * 9  = 9
2 * 9  = 18
3 * 9  = 27
4 * 9  = 36
5 * 9  = 45
6 * 9  = 54
7 * 9  = 63
8 * 9  = 72
9 * 9  = 81
10 * 9  = 90
11 * 9  = 99
12 * 9  = 108
BUILD SUCCESSFUL
```

9

9

Example3: Pseudocode- REPEAT…UNTIL structure

- REPEAT…UNTIL structure is similar to the DOWHILE structure (group of statements are repeated in accordance with specific condition).
- REPEAT…UNTIL structure is called trailing decision loop because it tests condition at the end of the loop.

```
1. SET count =1
2. REPEAT
3.        Prompt for number
4.        GET number
5.        Square = number * number
6.        Display square
7.        count = count + 1
8. UNTIL (count >5)
     ENDDO

     END
```

10

10

Java Program:

- Pseudocode REPEAT … UNTIL structure is called do … while loop in java program.

```
import java.util.Scanner;
public class Example_3
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number, count=1, square; // declaration
        do
        {
            System.out.print("Enter number  : ");
            number = keyboard.nextInt();
            square = number * number;
            System.out.printf("%d square is %d \n", number, square);
            count++;
        }while(count <=5);
    }
}
```

11

KAPLAN

Output

```
run:
Enter number  : 12
12 square is 144
Enter number  : 5
5 square is 25
Enter number  : -6
-6 square is 36
Enter number  : 10
10 square is 100
Enter number  : -14
-14 square is 196
BUILD SUCCESSFUL (total time: 1
```

12

KAPLAN