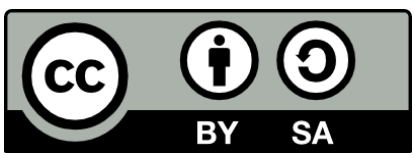
CSC 112: INTRODUCTION TO COMPUTER SCIENCE II

Published by the Centre for Open and Distance Learning,
University of Ilorin, Nigeria

E-mail: codl@unilorin.edu.ng
Website: https://codl.unilorin.edu.ng



Course Development Team

Subject Matter Expert

Dr Olawoyin
Department of Computer Science
University of Ilorin, Nigeria

Instructional Designers

Olawale Koledafe
Center for Open and Distance (CODL)
University of Ilorin, Nigeria

Miss. Damilola Adesodun
Department of Educational Technology,
University of Ilorin, Nigeria

Mr Jibril Mohammed

Content Editor:

Bankole Ogechi Ijeoma

From the Vice Chancellor

Courseware development for instructional use by the Centre for Open and Distance Learning (CODL) has been achieved through the dedication of authors and the team involved in quality assurance based on the core values of the University of Ilorin. The availability, relevance and use of the courseware cannot be timelier than now that the whole world has to bring online education to the front burner. A necessary equipping for addressing some of the weaknesses of regular classroom teaching and learning has thus been achieved in this effort.

This basic course material is available in different electronic modes to ease access and use for the students. They are available on the University's website for download to students and others who have interest in learning from the contents. This is UNILORIN CODL's way of extending knowledge and promoting skills acquisition as open source to those who are interested. As expected, graduates of the University of Ilorin are equipped with requisite skills and competencies for excellence in life. That same expectation applies to all users of these learning materials.

Needless to say, that availability and delivery of the courseware to achieve expected CODL goals are of essence. Ultimate attention is paid to quality and excellence in these complementary processes of teaching and learning. Students are confident that they have the best available to them in every sense.

It is hoped that students will make the best use of these valuable course materials.

Professor S. A. Abdulkareem
Vice Chancellor

Foreword

Courseware remains the nerve centre of Open and Distance Learning. Whereas some institutions and tutors depend entirely on Open Educational Resources (OER), CODL at the University of Ilorin considers it necessary to develop its own materials. Rich as OERs are and widely as they are deployed for supporting online education, adding to them in content and quality by individuals and institutions guarantees progress. Doing it in-house as we have done at the University of Ilorin has brought the best out of the Course Development Team across Faculties in the University. Credit must be given to the team for prompt completion and delivery of assigned tasks in spite of their very busy schedules.

The development of the courseware is similar in many ways to the experience of a pregnant woman eagerly looking forward to the D-day when she will put to bed. It is customary that families waiting for the arrival of a new baby usually do so with high hopes. This is the apt description of the eagerness of the University of Ilorin in seeing that the centre for open and distance learning [CODL] takes off.

The Vice-Chancellor, Prof. Sulyman Age Abdulkareem, deserves every accolade for committing huge financial and material resources to the centre. This commitment, no doubt, boosted the efforts of the team. Careful attention to quality standards, ODL compliance and UNILORIN CODL House Style brought the best out from the course development team. Responses to quality assurance with respect to writing, subject matter content, language and instructional design by authors, reviewers, editors and designers, though painstaking, have yielded the course materials now made available primarily to CODL students as open resources.

Aiming at a parity of standards and esteem with regular university programmes is usually an expectation from students on open and distance education programmes. The reason being that stakeholders hold the view that graduates of face-to-face teaching and learning are superior to those exposed to online education. CODL has the dual-mode mandate. This implies a combination of face-to-face with open and distance education. It is in the light of this that our centre has developed its courseware to combine the strength of both modes to bring out the best from the students. CODL students, other categories of students of the University of Ilorin and similar institutions will find the courseware to be their most dependable companion for the acquisition of knowledge, skills and competences in their respective courses and programmes.

Activities, assessments, assignments, exercises, reports, discussions and projects amongst others at various points in the courseware are targeted at achieving the objectives of teaching and learning. The courseware is interactive and directly points the attention of students and users to key issues helpful to their particular learning. Students' understanding has been viewed as a necessary ingredient at every point. Each course has also been broken into modules and their component units in sequential order.

Courseware for the Bachelor of Science in Computer Science housed primarily in the Faculty of Communication and Information Science provide the foundational model for Open and Distance Learning in the Centre for Open and Distance Learning at the University of Ilorin.

At this juncture, I must commend past directors of this great centre for their painstaking efforts at ensuring that it sees the light of the day. Prof. M. O. Yusuf, Prof. A. A. Fajonyomi and Prof. H. O. Owolabi shall always be remembered for doing their best during their respective tenures. May God continually be pleased with them, Aameen.

Bashiru, A. Omipidan
Director, CODL

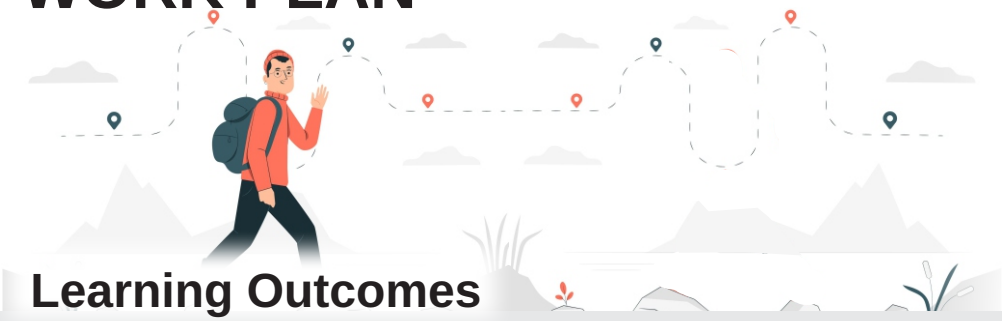
COURSE GUIDE

You are welcome to CSC 112 (Introduction to Computer Science II), a two (2) credit unit course of 12 study units. The main objective of this course is to further introduce you to computer science. This is because Introduction to Computer Science I have exposed you to the basic information on computer science. Therefore, CSC 112 discusses the basic concepts of computer and fundamental concepts of Basic programming language with a specific focus on QBasic programming language. The course is organized into five (5) distinct modules with each module addressing more than one study unit. The first module introduces you to the computer operating system, highlighting the characteristics, functions, and objectives of the operating system. Types of operating systems based on nature of computers and the operation(s) performed are also covered. The next module discusses problem-solving techniques, briefly explaining computer science and the computer system. This leads us to another module that discusses algorithms, flowcharts and pseudo codes. This module covers steps involved in a developed algorithm, flowchart, and pseudo code, with examples in solving problems, in detail. Also, the module discusses the computer programming language concepts generally. Concepts like compiler, interpreter, procedural, and object-oriented programming language are also covered. The next module discusses how to solve a problem using basic code in BASIC programming by explaining how to install QBASIC programming language, basic QBASIC character set, language contents, variables, and constant QBASIC language statement. Lastly, the module concludes with examples of a program written in QBASIC programming language concepts. The control structure is equally covered with a detailed explanation of selection, iterative, and sequence control structure. The built-in system function and string manipulation are not left out.

Course Goal

The goal of this course is to expose you to the knowledge required in programming using QBASIC Programming language. The course covered QBasic programming with a specific focus on how to use the QBASIC tools and code in Quick Basic to perform different arithmetic operations and string manipulations.

WORK PLAN



Learning Outcomes

At the end of this course, you should be able to:

I. Explain the fundamentals of operating system;

II. Describe the various types of operating system;

III. Differentiate between different types of operating system;

IV. Explain the fundamentals of problem solving techniques;

V. Describe algorithm, flowchart, and pseudocode and all their underlying concepts;

VI. Explain programming language;

Course Content

Module 1

Operating System

Unit 1: Operating System
Unit 2: Types of Operating System

Module 2

Introduction To Problem-Solving

Unit 1: Problem Solving Techniques

Related Courses

CSC 111 – Introduction to Computer Science I

CSC 216 – Assembly language

CSC 217 – Computer Programming I

VII. Describe the concept of computer programming languages;

VIII. Discuss the concept of compiler and interpreter;

IX. Highlight the operations of Switches, Routers, and Bridges;

X. Explain the basic principle of Procedural and Object-Oriented Programming Language; and

XI. Explain the concept of BASIC Programming Language.

Module 3

Algorithm And Flowchart

Unit 1: Algorithm
Unit 2: Flowchart
Unit 3: Pseudocode

Module 4

Programming Language

Unit 1: Computer Programming Language
Unit 2: Compiler and Interpreter
Unit 3: Procedural and Object-Oriented Programming Language

Module 5

Introduction To Basic Language (Qbasic)

Unit 1: BASIC Programming Language
Unit 2: Control Structures
Unit 3: System Built-in Functions and String Manipulation

Course Requirements

Requirements for success

The CODL Programme is designed for learners who are absent from the lecturer in time and space. Therefore, you should refer to your Student Handbook, available on the website and in hard copy form, to get information on the procedure of distance/e-learning. You can contact the CODL helpdesk which is available 24/7 for every of your enquiry.

Visit CODL virtual classroom on http://codllms.unilorin.edu.ng. Then, log in with your credentials and click on CSC 112. Download and read through the unit of instruction for each week before the scheduled time of interaction with the course tutor/facilitator. You should also download and watch the relevant video and listen to the podcast so that you will understand and follow the course facilitator.

At the scheduled time, you are expected to log in to the classroom for interaction. Self-assessment component of the courseware is available as exercises to help you learn and master the content you have gone through.

You are to answer the Tutor Marked Assignment (TMA) for each unit and submit for assessment

Embedded Support Devices

Support menus for guide and references

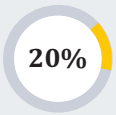
Throughout your interaction with this course material, you will notice some set of icons used for easier navigation of this course materials. We advise that you familiarize yourself with each of these icons as they will help you in no small ways in achieving success and easy completion of this course. Find in the table below, the complete icon set and their meaning.

 **Introduction**	 **Learning Outcomes**	 **Main Content**

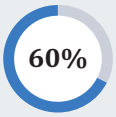
 Summary	 Tutor Marked Assignment	 Self Assessment
 Web Resources	 Downloadable Resources	 Discuss with Colleagues
 References	 Futher Reading	 Self Exploration

Grading and Assessment

TMA



CA

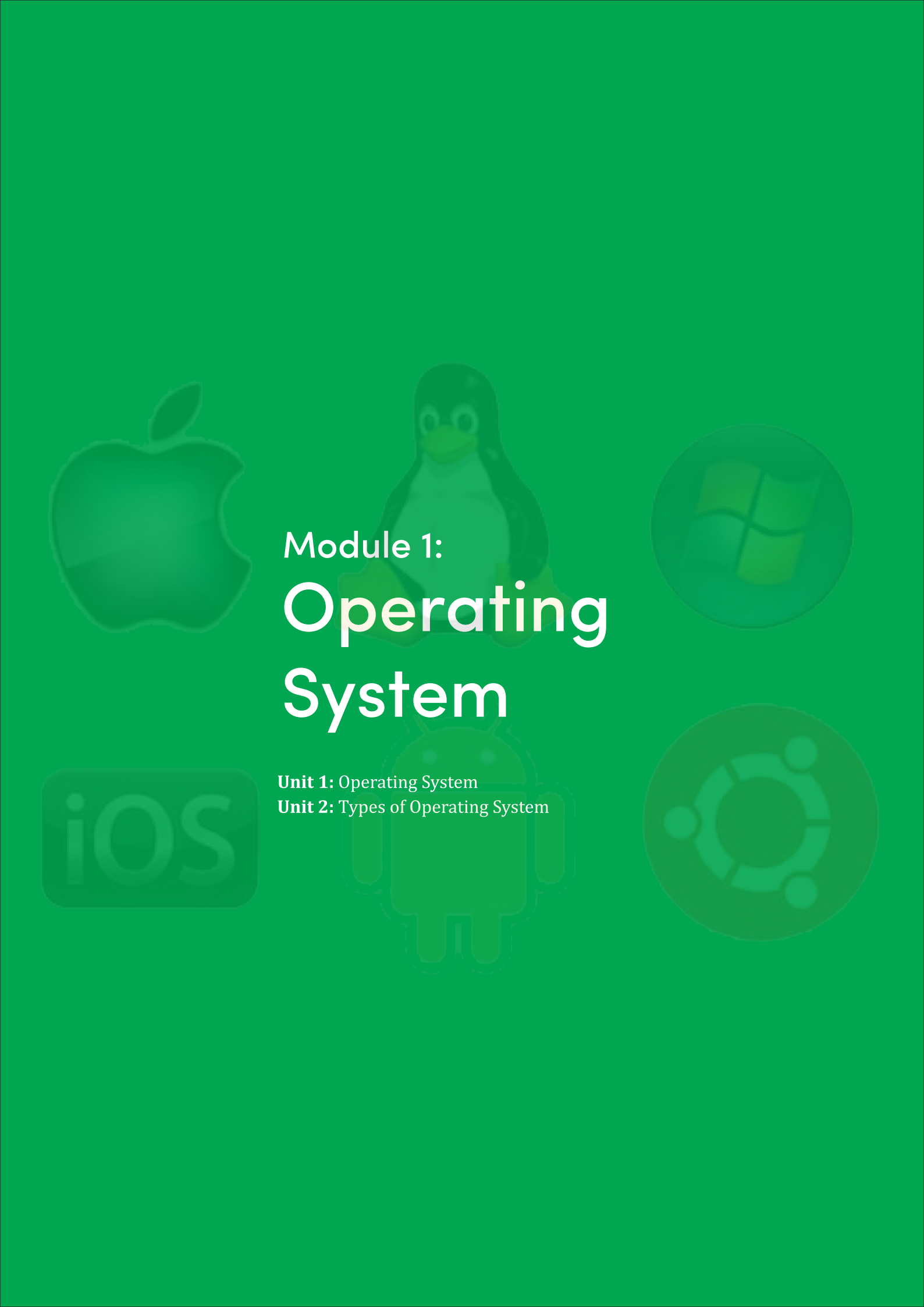


Exam

Total

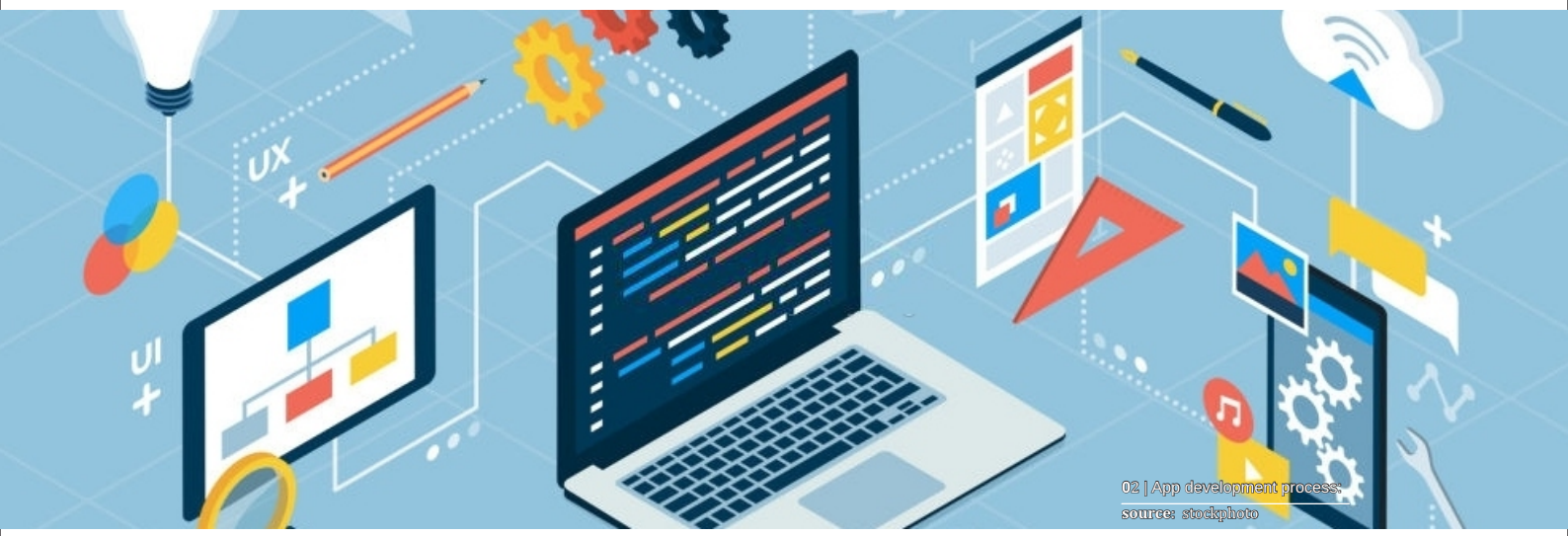
Picture: Operating system

Photo Source: Adobe Stock



Module 1:
Operating System

Unit 1: Operating System
Unit 2: Types of Operating System



02 | App development process.
source: stockphoto

UNIT 1

Introduction to Operating System

Introduction

I welcome you to the first unit of this lesson. Here, I will define an **operating system** (**OS**) to you as a set of computer programs that manage the hardware and software resources of a computer. An operating system maintains a proper balance between the software and hardware present in a computer system. It may interest you to know that operating system could also be called the infrastructure software component of a computer system because it is responsible for the management and coordination of activities and the sharing of the limited resources of the computer. The operating system acts as a host for applications that are run on the machine. As a host, one of the purposes of an operating system is to handle the details of the operation of the hardware. This relieves application programs from having to manage these details and makes it easier to write applications. Operating Systems can be viewed from two points of view**: resource manager** and **extended machines**. From the **resource manager** point of view, Operating Systems manage the different parts of the system efficiently, and from an **extended machine** point of view, Operating Systems provide a virtual machine to a user that is more convenient to use.

Learning Outcomes

At the end of this unit, you should be able to:

1. Define an operating system,
2. State at least five (5) characteristics of an operating system,
3. List at least six (6) objectives of an operating system
4. Explain how an operating system works

Main Content

Definition of Operating system

1 min

SAQ 1

It is important you know that an operating system, or OS, controls the overall activities of the computer system, including the hardware and the software. We can say it is the software that allows other programs to run for smooth communication with the hardware. It is a group of computer programs that coordinates the activities of all computer system. I want you to carefully note the common examples of desktop operating systems which are listed as follows: Windows, Mac OS X, and Linux. While each OS is different, they all provide a graphical user, or GUI, that includes a desktop and the ability to manage files and folders. They also allow you to install and run programs written for the operating system.

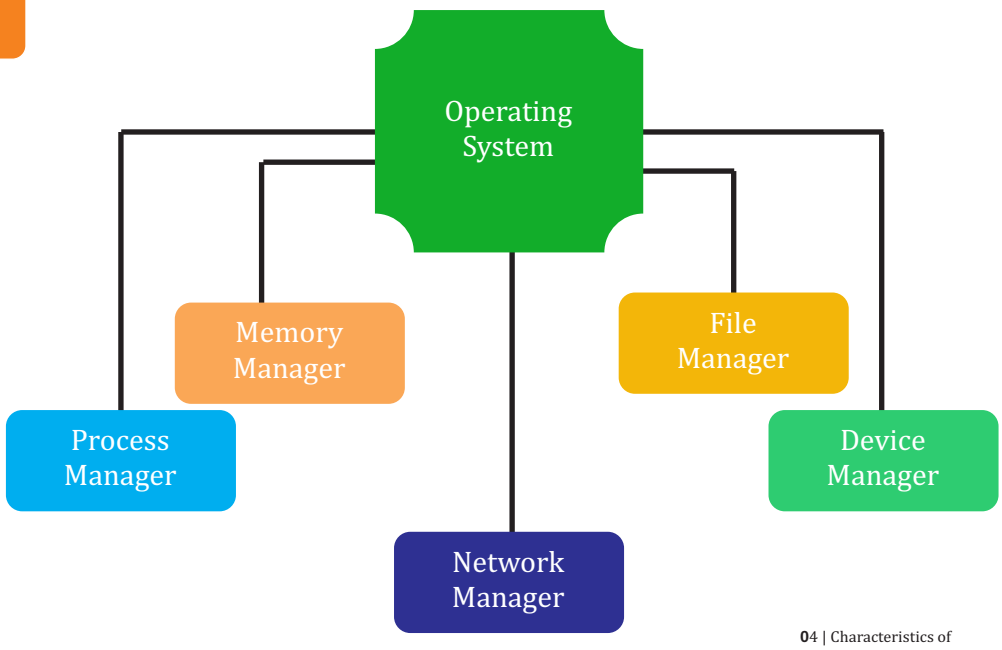
At the foundation of all system software, an operating system performs basic tasks such as controlling and allocating memory, prioritizing system requests, controlling input and output devices, facilitating networking, and managing file systems. It is important you know that the operating system forms a platform for other system software and application software. You should understand Windows, Linux, and Mac OS are some of the most popular OS's.

03 | Operating System process
source: Lloogg

Characteristics of Operating System

1 min

SAQ 2



1. **Memory Management:** It keeps track of primary memory; that is what parts of it are in use by whom, what parts are not in use etc. It allocates the memory when the process or program requests it.

2. **Processor Management:** It allocates the processor (CPU) to a processor. It allocates processor when the processor is no longer required.

3. **Device Management:** Keep tracks of all devices. This is also called the I/O controller. Decides which process gets the device when and for how much time.

4. **File Management:** Allocates resources. De-allocates the resource. Decides who gets the resources.

5. **Security:** The provision of the use of passwords and other similar techniques, operating system preventing unauthorized access to programs and data.

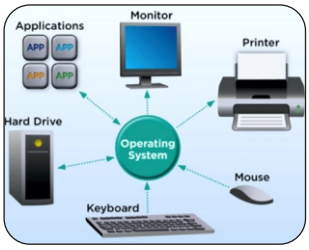
6. **Error-detecting aids** Production of dumps, traces, error messages, and other debugging and error-detecting methods.

Objectives of Operating System

2 mins

It's also very important you know Modern Operating systems generally have the following three major goals.

03 | Operating System process
source: Lloogg



To hide details of hardware by creating an abstraction

Abstraction occurs when we use software to hides lower-level details and provides a set of higher-level functions. We use an operating system to transforms the physical world of devices, instructions, memory, and time into a virtual world that is the result of abstractions built by the operating system. It may interest you to know there are several reasons for abstraction.

First, the code needed to control peripheral devices is not standardized. Operating systems provide subroutines called device drivers that perform operations on behalf of programs, for example, input/output operations.

Second, the operating system introduces new functions as it abstracts the hardware. For instance, an operating system introduces the file abstraction so that programs do not have to deal with disks.

Third, the operating system transforms the computer hardware into multiple virtual computers, each belonging to a different program. Each program that is running is called a process. Each process views the hardware through the lens of abstraction. Fourth, the operating system can enforce security through abstraction.

To allocate resources to processes (Resource management)

An operating system controls how processes (the active agents) may access resources (passive entities).

Provide a pleasant and effective user interface

You or the user interacts with the operating systems through the user interface and usually interested in the "look and feel" of the operating system. You should know the most important components of the user interface are the command interpreter, the file system, on-line help, and application integration. The recent trend has been toward increasingly integrated graphical user interfaces that encompass the activities of multiple processes on networks of computers.

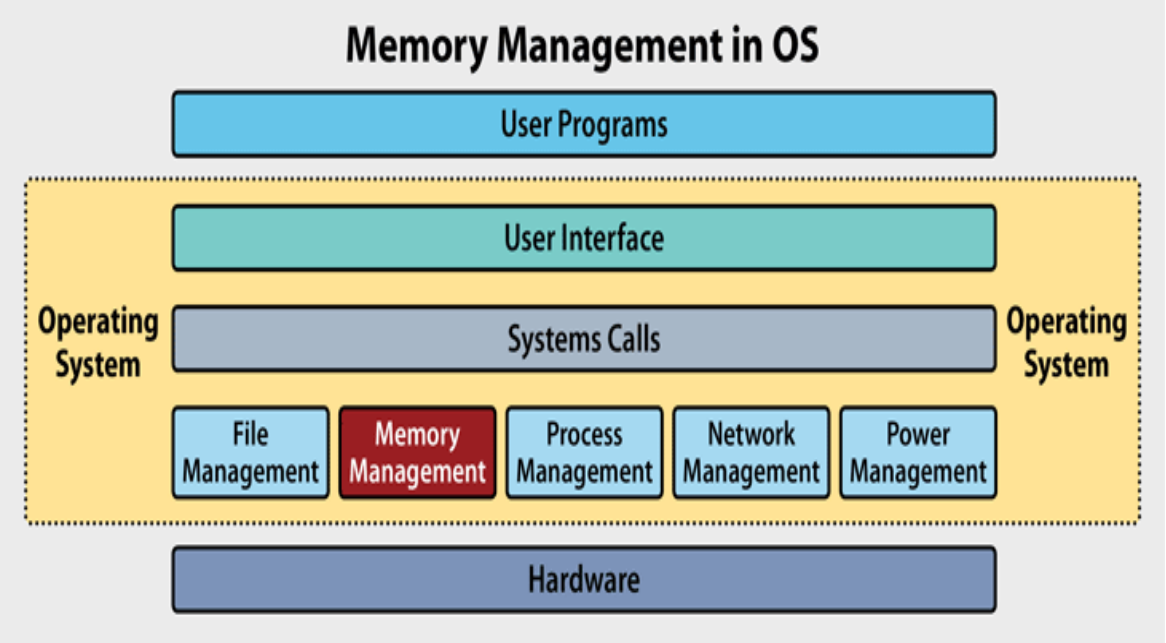
I will like to tell you the other operating system objectives which is as follows:

SAQ 3

Objectives of an Operating System

1 min

1. Making a computer system convenient for use in an efficient manner
2. To hide the details of the hardware resources from the users
3. To provide users a convenient interface to use the computer system.
4. To act as an intermediary between the hardware and its users and thus make it easier for the users to access and use other resources.
5. Manage the resources of a computer system.
6. Keep track of who is using which resource by granting resource requests, according to a resource use and mediating conflicting requests from different programs and users.
7. The efficient and fair sharing of resources among users and programs.



How an Operating System Works (1 mins) [SAQ 4]

1 min

Do you know when the power of a computer is turned on; the first program that runs is usually a set of instructions kept in the computer's read-only memory (ROM)? It may interest you to know that this code examines the system hardware to make sure everything is functioning properly. **This power-on self-test (POST)** checks the CPU, memory, and **basic input-output system (BIOS)** for errors and stores the result in a special memory location. Once the POST has completed, the software loaded in ROM (sometimes called the BIOS or firmware) will begin to activate the computer's disk drives. In most modern computers, when the computer activates the hard disk drive, it finds the first piece of the operating system: **the bootstrap loader**. I also want you to know the bootstrap loader is a small program that has a single function: It loads the operating system into memory and allows it to begin operation. In the most basic form, you should know the bootstrap loader sets up the small driver programs that interface with and control the various hardware subsystems of the computer. It sets up the divisions of memory that hold the operating system, user information, and applications. It establishes the data structures that will hold the myriad signals, flags, and semaphores that are used to communicate within and between the subsystems and applications of the computer. Then it turns control of the computer over to the operating system.

SAQ 4

Summary

In this unit, you have learned that:

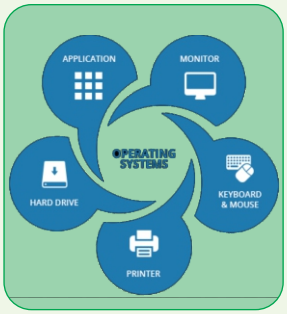
- The operating system supervises the overall activities of the computer system.
- Is the software that allows other programs to run smoothly with the hardware.
- Allocate processor when a processor is no longer required
- Making a computer system convenient to use in an efficient manner

Activity 1

Introduction to Operating System

Questions

- (i) In your own words, what is an Operating System?
- (ii) What are the functions/importance of the bootstrap loader



Self Assessment Questions

1 Define an operating system?
2 State six (6) characteristics of an operating system
3 Mention five (5) objectives of an operating system?
4 Briefly explain how an operating system works

Tutor Marked Assessment

- Explain six (6) characteristics of an operating system
- Discuss the three (3) major objectives of an operating system
- Explain the procedures of function of an operating system

Further Reading

- Adele Goldberg and Kenneth S. Rubin, Succeeding with Objects: Decision Frameworks for Project Management, Addison-Wesley, 1995, ISBN 0-201-62878-3.
- [Brinch-Hansen (1973)] P. Brinch-Hansen, Operating System Principles, Prentice-Hall (1973).
- Operating System Concepts, by Abraham Silberschatz, ISBN13: 978-1118063330 9th Edition, 2013.

References

- Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dolling, Helena Gilchrist, Fiona Hayes, and Paul Jeremaes, Object-Oriented Development: The Fusion Method, Prentice-Hall, 1994, ISBN 0-13-338823-9.
- Frank Buschmann, Regine Meunier, Hans Johnert, Peter Sommerlad, and Michael Stal, Pattern-Oriented Software Architecture: A System of Patterns, Wiley, 1996, ISBN 0-471-95869-7.

UNIT 2

Types of Operating System

Introduction

In this unit, I will explain in details, the types of operating systems. This will give you an understanding of the types of operating systems and how they operate. The functions of an operating system were also discussed and the flaws of the operating system.

Learning Outcomes

At the end of this unit, you should be able to:

1. Explain types of operating system based on the user per system,
2. Explain types of operating system based on the mode of operation
3. Explain the functions of the operating system,
4. Explain the flaws in the operating system

Main Content

SAQ 1

Types of Operating System

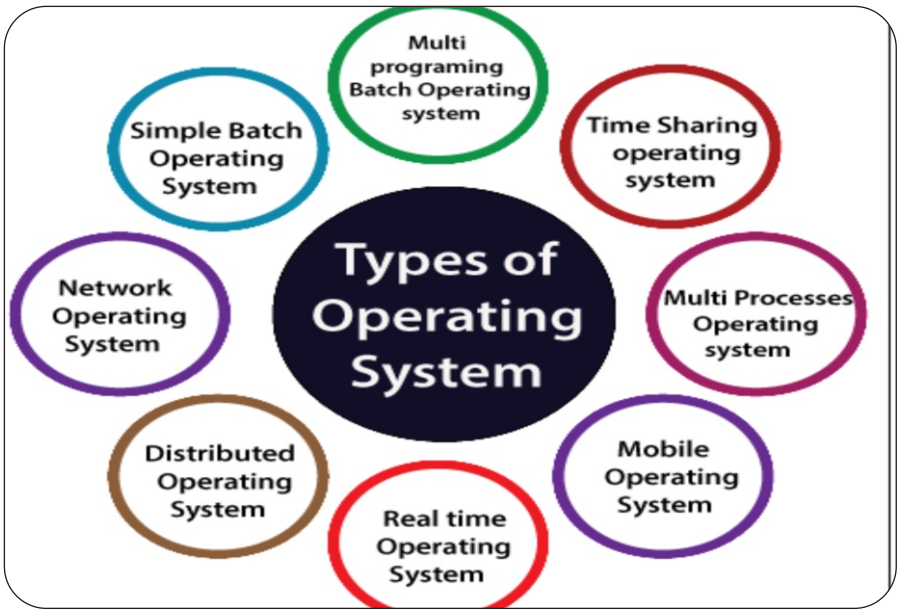
2 mins

A.Types of Operating Systems Based on the Types of Computer they Control and the Sort of Applications they Support

Based on the types of computers they control and the sort of applications they support, there are generally four types within the broad family of operating systems. The broad categories are as follows:

Real-Time Operating Systems (RTOS)

They are used to control machinery, scientific instruments, and industrial systems. You should bear in mind that an RTOS typically has the very little user-interface capability, and no end-user utilities, since the system will be a sealed box when delivered for use. A very important part of an RTOS is managing the resources of the computer so that a particular operation executes in precisely the same amount of time every time it occurs. In a complex machine, having a part move more quickly just because system resources are available may be just as catastrophic as having it not move at all because the system is busy. RTOS can be hard or soft. A hard RTOS guarantees that critical tasks are performed on time. However, soft RTOS is less restrictive. Here, a critical real-time task gets priority over other tasks and retains that priority until it completes.



02 | various type of operating system base on mode of type of computer control

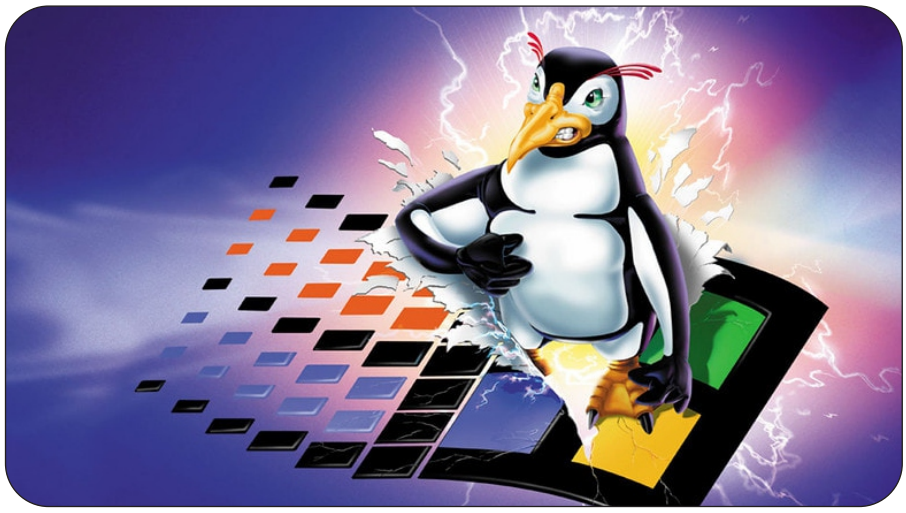
Single-User, Single-Tasking Operating System

As the name implies, it is important you know this operating system is designed to manage the computer so that one user can effectively do one thing at a time. The Palm OS for Palm handheld computers is a good example of a modern single-user, single-task operating system.

Single-User, Multi-Tasking Operating System

This is the type of operating system most of us use on our desktop and laptop computers today. **Windows 98** and the **Mac O.S.** are both examples of an operating system that will let a single user have several programs in operation at the same time.

03 | single user operating system



Multi-User Operating Systems

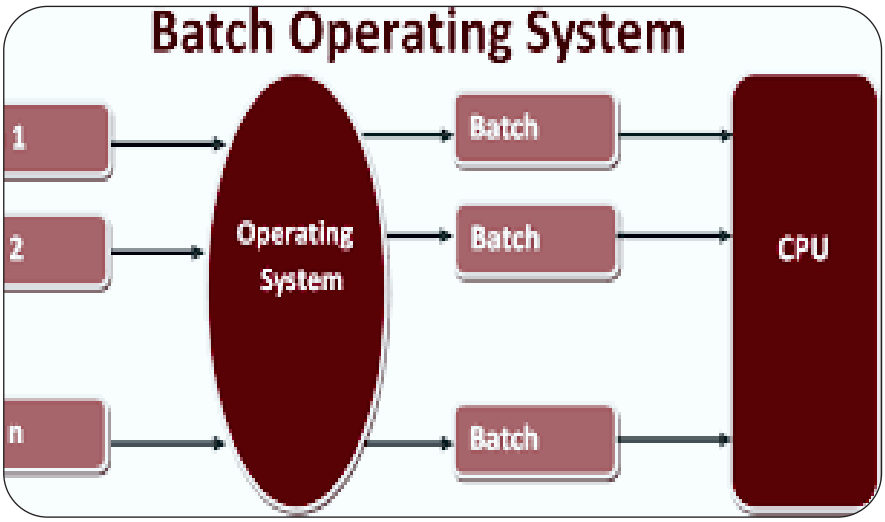
I want you to note a multi-user operating system allows many different users to take advantage of the computer's resources simultaneously. The operating system must make sure that the requirements of the various users are balanced, and that each of the programs they are using has sufficient and separate resources so that a problem with one user doesn't affect the entire community of users. **Unix, VMS**, and mainframe operating systems, such as **MVS**, are examples of multi-user operating systems. It's important for you to differentiate here between multi-user operating systems and single-user operating systems that support networking. **Windows 2000** and **Novell Netware** can each support hundreds or thousands of networked users, but the operating systems themselves are not true multi-user operating systems. The system administrator is the only user for Windows 2000 or Netware. The network support and the entire remote user logins the network enables are, in the overall plan of the operating system, a program being run by the administrative user.

Types of Operating System based on the Nature of Interaction that takes place between the Computer User and His/Her Program during its Processing

2 mins

SAQ 2

It good you try to understand that modern computer operating systems may be classified into three groups, which are distinguished by the nature of interaction that takes place between the computer user and his or her program during its processing. The three groups are: called batch, time-shared, and real-time operating systems.

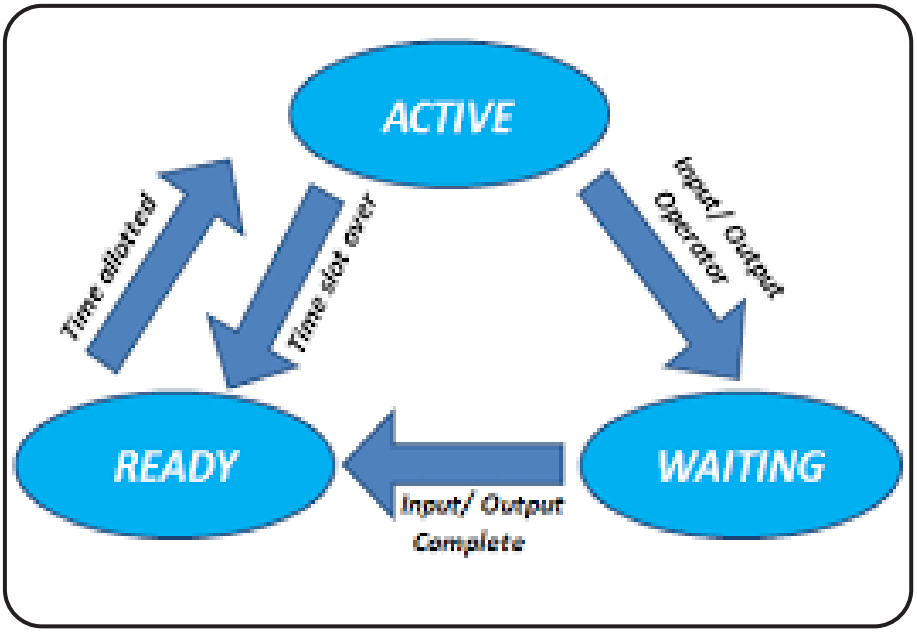


04 | Batch Operating System

Batch Processing OS

In a batch processing operating system environment, I want you to bear in mind users submit jobs to a central place where these jobs are collected into a batch and subsequently placed on an input queue at the computer where they will be run. In this case, the user has no interaction with the job during its processing, and the computer's response time is the turnaround time (i.e., results are ready for a return to the person who submitted the job).

05 | Time-Sharing Operating System

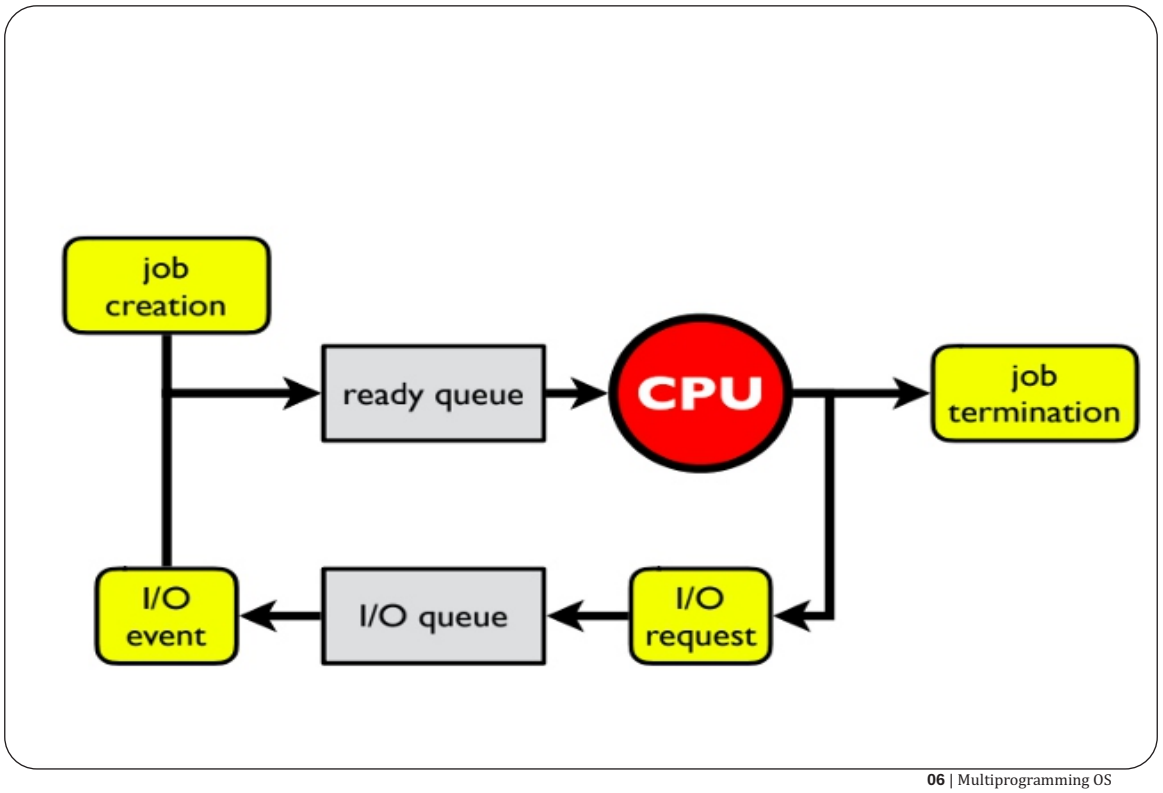


Time-Sharing OS

Time-sharing operating systems provide another mode for delivering computing services. In this environment, I want you to know a computer provides computing services to several or many users concurrently on-line. Here, the various users are sharing the central processor, the memory, and other resources of the computer system in a manner facilitated, controlled, and monitored by the operating system. The user, in this environment, has nearly full interaction with the program during its execution, and the computer's response time may be expected to be no more than a few seconds.

Real-Time OS

The third classes of operating systems, real-time operating systems, are designed to service those applications where response time is of the essence to prevent error, misrepresentation, or even disaster. Examples of real-time operating systems are those which handle airline reservations, machine tool control, and monitoring of a nuclear power station. The systems, in this case, are designed to be interrupted by an external signal that requires the immediate attention of the computer system. Many computer operating systems are hybrids, providing for more than one of these types of computing services simultaneously. It is especially common to have a background batch system running in conjunction with one of the other two on the same computer.



06 | Multiprogramming OS
source: Thanoshan MV

Other Types of OS based on the Definition of the System/Environment

2 mins

Is very important you bear in mind that many other definitions are important to gaining a better understanding and subsequently classifying operating systems:

Multiprogramming Operating System

A multiprogramming operating system is a system that allows more than one active user program (or part of a user program) to be stored in the main memory simultaneously. Thus, it is evident that a time-sharing system is a multiprogramming system, but note that a multiprogramming system is not necessarily a time-sharing system. A batch or real-time operating system could, and indeed usually do, have more than one active user program simultaneously in main storage. Another important, and all too similar, the term is 'multiprocessing.' A multiprocessing system is a computer hardware configuration that includes more than one independent processing unit. The term multiprocessing is generally used to refer to large computer hardware complexes found in major scientific or commercial applications.

Network Operating Systems

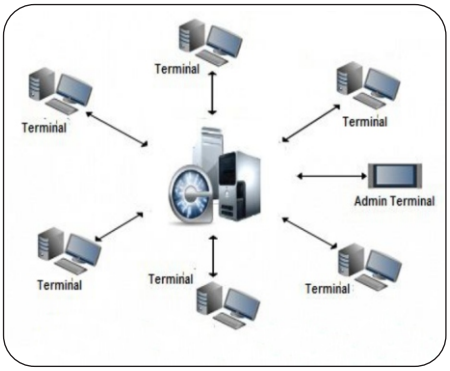
A networked computing system is a collection of physically interconnected computers. The operating system of each of the interconnected computers must contain, in addition to its stand-alone functionality, provisions for handling communication and transfer of programs and data among the other computers with which it is connected. In a network operating system, the users are aware of the existence of multiple computers and can log in to remote machines and copy files from one machine to another. Each machine runs its local operating system and has its user (or users). Network operating systems are designed with more complex functional capabilities. Network operating systems are not fundamentally different from single-processor operating systems. They need a network interface controller and some low-level software to drive it, as well as programs to achieve remote login and remote file access, but these additions do not change the essential structure of the operating systems.

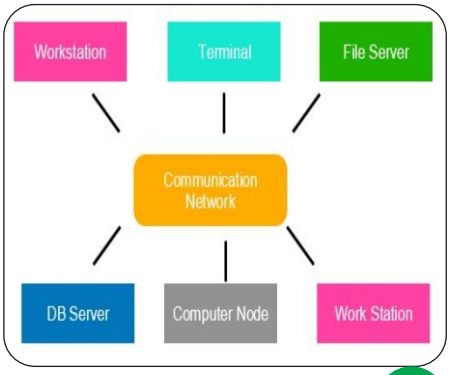
Distributed Operating Systems

A distributed computing system consists of many computers that are connected and managed so that they automatically share the job processing load among the constituent computers, or separate the job load as appropriate, particularly configured processors. Such a system requires an operating system which, in addition to the typical stand-alone functionality, provides coordination of the operations and information flow among the component computers. The distributed computing environment and its operating systems, like the networking environment, are designed with more complex functional capabilities. However, a distributed operating system, in contrast to a network operating system, is one that appears to its users as a traditional uniprocessor system, even though it is composed of multiple processors. In a truly distributed system, users should not be aware of where their programs are being run or where their files are located; that should all be handled automatically and efficiently by the operating system.

07 | Network Operating System

08 | Distributed Operating System





SAQ 3

Functions of Operating Systems

1 min

You should try to take note of the major functions of operating systems which are mentioned below;

- Processor management: An operating system deals with the assignment of the processor to different tasks being performed by the computer system.
- Memory management: An operating system deals with the allocation of main memory and other storage areas to the system programs as well as user programs and data.
- Input/output management: An operating system deals with the coordination and assignment of the different output and input device while one or more programs are being executed.
- File management: An operating system deals with the storage of files of various storage devices to another. It also allows all files to be easily changed and modified through the use of text editors or some other file manipulation routines.

Also, I want you to note other functions of the operating systems which are;

- Establishment and enforcement of a priority system: The operating system determines and maintains the order in which jobs are to be executed in the computer system.
- The automatic transition from job to job as directed by special control statements.
- Interpretation of commands and instructions.
- Coordination and assignment of compilers, assemblers, utility programs, and other software to the various uses of the computer system.
- Facilitates easy communication between the computer system and the computer operator (human). It also establishes data security and integrity.

Operating Systems Flaws

1 min

It may interest you to know operating systems are written by human programmers who make mistakes. Therefore there can be errors in the code even though there may be some testing before the product is released. Some companies have better software quality control and testing than others so that one may notice varying levels of quality from the operating system to the operating system. It is important you note errors in operating systems cause three main types of problems:

SAQ 4

- **System crashes and instabilities** - These can happen due to a software bug typically in the operating system, although computer programs being run on the operating system can make the system more unstable or may even crash the system by themselves. It is important you know this varies depending on the type of operating system. A system crash is an act of a system freezing and becoming unresponsive, which would cause the user to need to reboot.

- **Security flaws** - Some software errors leave a door open for the system to be broken into by unauthorized intruders. As these flaws are discovered, unauthorized intruders may try to use these to gain illegal access to your system. Patching these flaws often will help keep your computer system secure. How this is done will be explained later.

- Sometimes errors in the operating system will cause the computer not to work correctly with some peripheral devices such as printers.

Summary

In this unit, you have learned that:

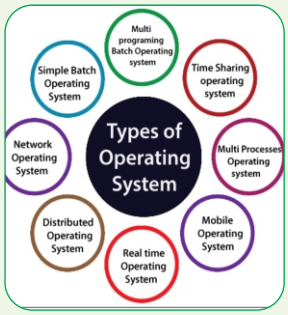
- Real-time OD are used to control machinery, scientific instruments, and industrial systems
- Single-user is designed to manage the computer so that one user can effectively do one thing at a time
- Single-user multi-tasking will let a single user have several programs in operation at the same time
- The multi-user operating system allows many different users to take advantage of the computer's resources simultaneously
- There can be errors in the code even though there may be some testing before the product is released

Activity 1

Types of Operating System

Questions

- Briefly explain the types of operating system based on mode of operation
- Mention four functions of operating system
- What are the flaws of the operating system



Self Assessment Questions

- State three (3) examples of an operating system?
- Mention three (4) operating system based on their mode of operation
- What are operating system flaws?
- State at least four (4) functions of an operating system?

Tutor Marked Assessment

- State and explain four (3) based on the nature of interaction that takes place between the computer user and program during its Processing
- Discuss the six (6) pseudocode functions of an operating system
- Briefly explain the three (3) types of flaws occur in an operating system

Further Reading

- Adele Goldberg and Kenneth S. Rubin, Succeeding with Objects: Decision Frameworks for Project Management, Addison-Wesley, 1995, ISBN 0-201-62878-3. 2013.
- [Brinch-Hansen (1973)] P. Brinch-Hansen, Operating System Principles, Prentice-Hall (1973).
- Operating System Concepts, by Abraham Silberschatz, ISBN13: 978-1118063330 9th Edition, 2013.

References

- Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dolling, Helena Gilchrist, Fiona Hayes, and Paul Jeremaes, Object-Oriented Development: The Fusion Method, Prentice-Hall, 1994, ISBN 0-13-338823-9.
- Frank Buschmann, Regine Meunier, Hans Johnert, Peter Sommerlad, and Michael Stal, Pattern-Oriented Software Architecture: A System of Patterns, Wiley, 1996, ISBN 0-471-95869-7.

Picture: Problem Solving

Photo Source: iStock

Module 2:

Introduction to Problem-Solving

Unit 1: Problem Solving Techniques

02 | Problem Solving:
source: Vectorstock

UNIT 1

Problem Solving Techniques

Introduction

In this unit, Problem Solving Techniques, I will expose you to the concept Computer System, Computer Science and how to solve computer problems. A computer is a very powerful and versatile machine capable of performing a multitude of different tasks, yet it has no intelligence or thinking power. The Intelligence Quotient (I.Q) of a computer is zero. A computer performs many tasks exactly in the same manner as it is told to do. This places responsibility on the user to instruct the computer in a correct and precise manner so that the machine can properly perform the required job. A wrong or ambiguous instruction may sometimes prove disastrous. To instruct a computer correctly, the user must have a clear understanding of the problem to be solved. Apart from this, he should be able to develop a method in the form of a series of sequential steps to solve the problem. Once the problem is well-defined, and a method of solving it is developed, then instructing the computer to solve it becomes a relatively easy task. Thus, before attempting to write a computer program to solve a given problem, it is necessary to formulate or define the problem in a precise manner. Once the problem is defined, the steps required to solve it must be stated clearly in the required order.

Learning Outcomes

At the end of this unit, you should be able to:

1. Explain the concept of a computer system
2. Discuss the concept of computer science
3. Explain the concept of Problem solving

Main Content

Computer System

4 mins

I want you to be aware that a computer is a programmable machine that receives input, stores and manipulates data, and provides output in a useful format. Concerning today's computers, the "machine" part of the computer is called the hardware, while the "programmable" part is called the software. Since computers are used everywhere, you can get involved with them from just about any field of study. However, there are specific fields that are more computer-related than others. For example, the fields of electrical engineering and computer systems engineering primarily focus on the design and manufacturing of computer hardware, while the fields of software engineering and computer science primarily focus on the design and implementation of the software.

The software itself is another important thing you need to know which could be broken down into 3 main categories:

System Software: This is designed to operate the computer's hardware and to provide and maintain a platform for running applications. (e.g., Windows, macOS, Linux, Unix, etc.)

SAQ 1

Middleware: This is a set of services that allows multiple processes running on one or more machines to interact. It is often used to support and simplify complex distributed applications. It can also allow data contained in one database to be accessed through another. Middleware is sometimes called plumbing because it connects two applications and passes data between them. (e.g., web servers, application servers).

Application Software: is designed to help you perform one or more related specific tasks. Depending on the work for which it was designed, an application can manipulate texts, numbers, graphics, or a combination of these elements. (e.g., office suites, web browsers, video games, media players, etc.)

The area of software design is huge. In this course, we will investigate the basics of creating some simple application software. If you continue your degree in computer science, you will take additional courses that touch upon the other areas of system software and middleware.

Software is usually written to fulfill some needs that the general public, private industry, or government needs. Ideally, the software is meant to make it easy for the user (i.e., the person using the software) to accomplish some task, solve some problem

or entertain him/herself. Regardless of the user's motivation for using the software, many problems will arise when trying to develop the software in a way that produces correct results, is efficient and robust, easy to use, and visually appealing. That is where computer science comes in:

Computer Sciences

It's important you know computer science is the study of the theoretical foundations of information and computation, and practical techniques for their implementation and application in computer systems. So, computer science is all about the way you input information and then performing some computations & analysis to solve a particular problem or produce the desired result, which depends on the application at hand.

Computer science is similar to mathematics in that you use both as a means of defining and solving some problems. Computer-based applications often use mathematical models as a basis for how they solve the problem at hand.

In mathematics, we often express a solution in terms of formulas and equations. In computer science, we express the solution in terms of a program: A program is a sequence of instructions that can be executed by a computer to solve some problem or perform a specified task. However, computers do not understand arbitrary instructions written in English, French, Spanish, Chinese, Arabic, Hebrew, Yoruba, Hausa, Igbo, etc. Instead, computers have the languages that they understand. Each of these languages is known as machine language. The programming language is easy for users/programmers to communicate with the system instead of 0s and 1s.

A programming language is an artificial language designed to automate the task of organizing and manipulating information and to express problem solutions precisely. A programming language “boils down to” a set of words, rules, and tools that are used to explain (or define) what you are trying to accomplish. There are many different programming languages, just as there are many different "spoken" languages. Traditional programming languages were known as structural programming languages (e.g., C, Fortran, Pascal, Cobol, Basic). Since the late 80's however, object-oriented programming languages have become more popular (e.g., JAVA, C++, C#)

It may interest you to know there are also other types of programming languages, such as functional programming languages and logic programming languages. According to the Tiobe index (i.e., a good site for ranking the popularity of programming languages), as of February 2011, the 10 most actively used programming languages were (in order of popularity): Java, C, C++, PHP, Python, C#, VisualBasic, Objective-C, Perl, Ruby

When you are thinking of jobs and careers, many people think that computer science

covers anything related to computers (i.e., anything related to Information Technology). However, computer science is not an area of study that pertains to IT support, repairing computers, nor installing and configuring networks. Nor does it have anything to do with simply using a computer such as doing word-processing, browsing the web, or playing games. The focus of computer science is on understanding what goes on behind the software and how software/programs can be made more efficiently.

I want you to carefully note the four general areas that is considers crucial to the discipline of computer science by the Computer Sciences Accreditation Board (CSAB) identifies:

1. Theory of computation - investigates how specific computational problems can be solved efficiently

2. Algorithms and data structures - investigates efficient ways of storing, organizing and using data

3. Programming methodology and languages - investigates different approaches to describing and expressing problem solutions

4. Computer elements and architecture - investigate the design and operation of computer systems

However, it is important you note they also identify other important fields of computer science:

1. software engineering
2. computer networking & communication
3. parallel computation
4. computer-human interaction
5. operating systems
6. artificial intelligence
7. database systems
8. distributed computation
9. computer graphics
10. numerical & symbolic computation

I want you to know there are aspects of each of the above fields, which can fall under the general areas mentioned previously. For example, within the field of database systems, you can work on theoretical computations, algorithms & data structures, and programming methodology. As you continue your studies in computer science, you will be able to specialize in one or more of these areas that interest you. This course, however, is meant to be an introduction to programming computers with an emphasis on problem-solving.

SAQ 2

Procedure (Steps Involved in Problem Solving)[SAQ 3]

4 mins

It may interest you to know a computer cannot solve a problem on its own. One has to provide step by step solutions of the problem to the computer. The task of problem-solving is not that of the computer. It is the programmer who has to write down the solution to the problem in terms of simple operations which the computer can understand and execute. If you want to solve a problem with the computer, you have to pass through certain stages or steps.

Regardless of the area of study, computer science is all about solving problems with computers. The problems that we want to solve can come from any real-world problem or perhaps even from the abstract world. We need to have a standard systematic approach to solving problems.

Problem Solving is the sequential process of analyzing information related to a given situation and generating appropriate response options. It is good you know there are 6 steps that you should follow to solve a problem:

1. Understand the Problem

2. Formulate a Model

3. Develop an Algorithm

4. Write the Program

5. Test the Program

6. Evaluate the Solution

I want you to consider a simple example of how the input/process/output works on a simple problem:

Example: Calculate the average grade for all students in a class.

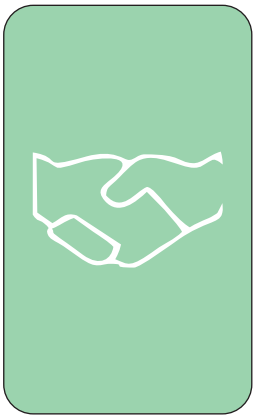
1. Input: get all the grades … perhaps by typing them in via the keyboard or by reading them from a USB flash drive or hard disk.

2. Process: add them all up and compute the average grade.

3. Output: output the answer to either the monitor, to the printer, to the USB flash drive or hard disk, or a combination of any of these devices.

As you can see, the problem is easily solved by simply getting the input, computing something, and producing the output. Let us now examine the 6 steps to problem-solving within the context of the above example.

STEP 1: Understand the Problem:



It sounds strange, but the first step to solving any problem is to make sure that you understand the problem that you are trying to solve. You need to know:
What input data/information is available?
What does it represent?
What format is it in?
Is anything missing?
Do I have everything that I need?
What output information am I trying to produce?
What do I want the result to look like …a text, a picture, a graph …?
What am I going to have to compute?

STEP 2: Formulate a Model:

Now we need to understand the processing part of the problem. Many problems break down into smaller problems that require some kind of simple mathematical computations to process the data. We need to know the model (or formula) for computing the average of a bunch of numbers. If there is no such "formula," we need to develop one. Often, however, the problem breaks down into simple computations that we well understand. Sometimes, we can look up certain formulas in a book or online if we get stuck.

STEP 3: Develop an Algorithm:

Now that we understand the problem and have formulated a model, it is time to come up with a precise plan of what we want the computer to do.

STEP 4: Write the Program:

Now that we have a precise set of steps for solving the problem, most of the hard work has been done. We now have to transform the algorithm from step 3 into a set of instructions that can be understood by the computer. Writing a program is often called "writing code" or “implementing an algorithm.” So, the code (or source code) is the program itself.

The computer requires precise instructions to understand what you are asking it to do. For example, if you removed one of the semi-colon characters (;) from a program, the computer would become confused as to what you are doing because the (;) is what it understands to be the end of an instruction. Leaving one of them off will cause your program to generate what is known as a compile error.

Compiling is the process of converting a program into instructions that can be understood by the computer.

The longer your program becomes, the more likely you will have multiple compile errors. You need to fix all such compile errors before continuing to the next step.

STEP 5: Test the Program:

Once you have a program written that compiles, you need to make sure that it solves the problem that it was intended to solve and that the solutions are correct.

Running a program is the process of telling the computer to evaluate the compiled instructions.

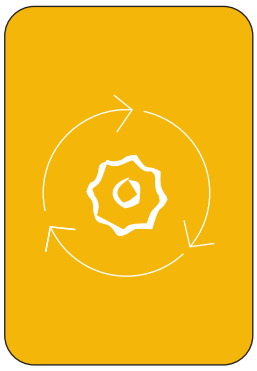
When you run your program, if all is well, you should see the correct output. It is possible, however, that your program works correctly for some set of data input but not for all. If the output of your program is incorrect, it is possible that you did not convert your algorithm properly into a proper program. It is also possible that you did not produce a proper algorithm back in step 3 that handles all situations that could arise. Maybe you performed some instructions out of sequence. Whatever happened, such problems with your program are known as bugs.

Bugs are problems/errors with a program that causes it to stop working or produce incorrect or undesirable results.

You should fix as many bugs in your program as you can find. To find bugs effectively, you should test your program with many test cases (called a test suite). It is also a good idea to have others test your program because they may think up situations or input

data that you may never have thought of. The process of finding and fixing errors in your code is called debugging, and it is often a very time-consuming "chore" when it comes to being a programmer. If you take your time to follow problem-solving steps 1 through 3 carefully, this should greatly reduce the number of bugs in your programs, and it should make debugging much easier.

STEP 6: Evaluate the Solution:



Once your program produces a result that seems correct, you need to re-consider the original problem and make sure that the answer is formatted into a proper solution to the problem. It is often the case that you realize that your program solution does not solve the problem the way that you wanted it to. You may realize that more steps are involved. For example, if the result of your program is a long list of numbers, but you intended to determine a pattern in the numbers or to identify some features from the data, then simply producing a list of numbers may not suffice. There may be a need to display the information in a way that helps you visualize or interpret the results concerning the problem. Perhaps a chart or graph is needed. It is also possible that when you examine your results, you realize that you need additional data to solve the problem fully. Or, perhaps you need to adjust the results to solve the problem more efficiently (e.g., your game is too slow).

It is important to remember that the computer will only do what you told it to do. It is up to you to interpret the results in a meaningful way and determine whether or not it solves the original problem. It may be necessary to re-do some of the steps again, perhaps going as far back as step 1 again, if data was missing.

Summary

In this unit, you have learned that:

- A computer is a programmable machine that receives input, stores and manipulates data, and provides output in a useful format
- The software can be broken down into 3 main categories
- Computer science is the study of the theoretical foundations of information and computation
- Computer science is similar to mathematics in that both are used as a means of defining and solving some problem
- A computer cannot solve a problem on its own. One has to provide step by step solutions of the problem to the computer.

Activity

Problem Solving Techniques

Questions

- Computer science is the study of the theoretical foundations of information and computation. Extensively explain
- What are the procedures involved in solving a problem?

Self Assessment Questions

Formulate a model and then develop an algorithm for each of the following problems.

1. What is the computer system?
2. What is computer science?
3. Define problem-solving concerning computer science.

Tutor Marked Assessment

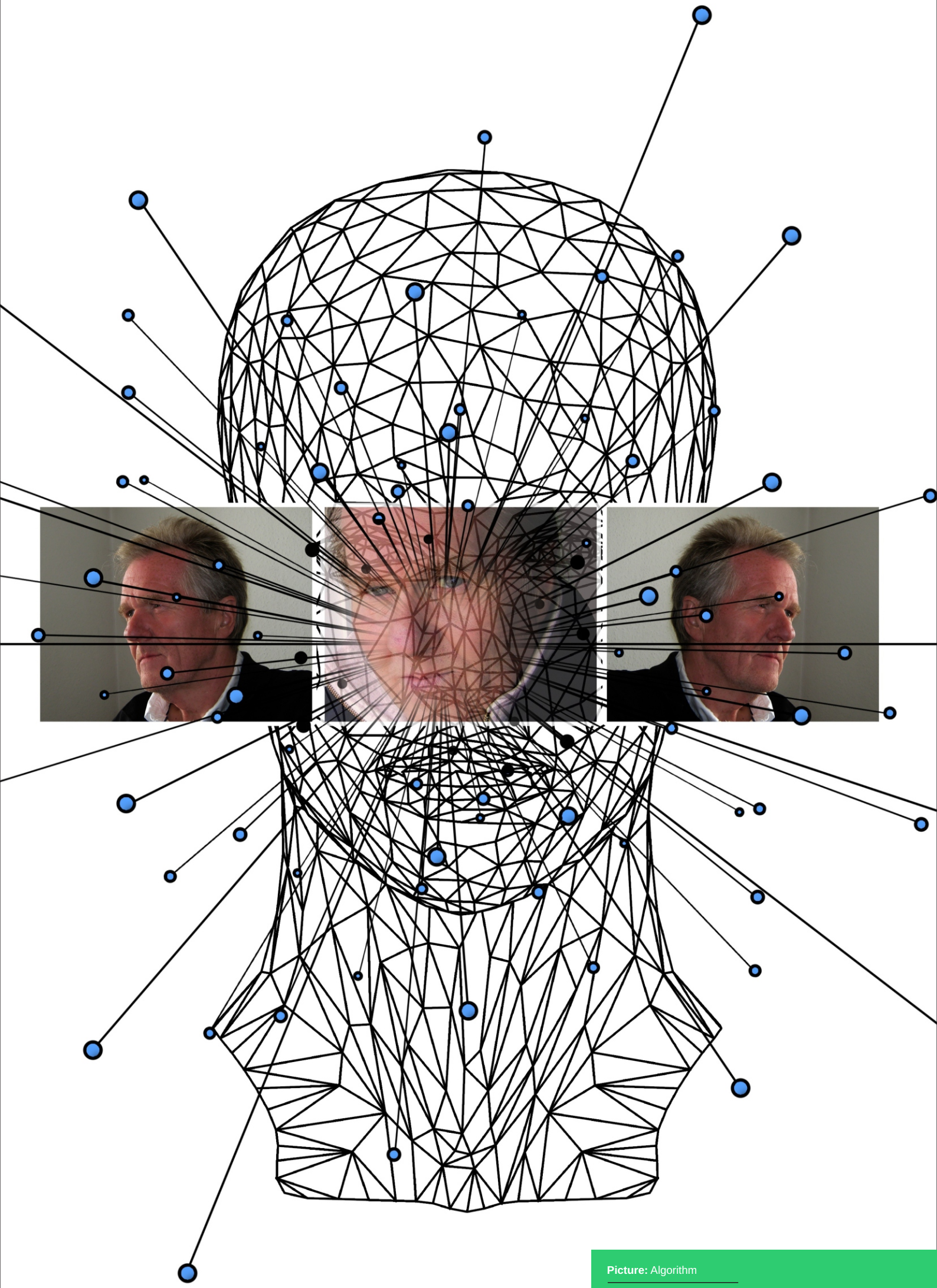
- Write the steps involved in solving a problem

Further Reading

- Michael Trombetta QBASIC for Students / Edition 1, McGraw-Hill Higher Education, 1993
- Wallace Wang Beginning Programming for Dummies by Wallace Wang, 2001

References

- Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dolling, Helena Gilchrist, Fiona Hayes, and Paul Jeremaes, Object-Oriented Development: The Fusion Method, Prentice-Hall, 1994, ISBN 0-13-338823-9.
- Frank Buschmann, Regine Meunier, Hans Johnert, Peter Sommerlad, and Michael Stal, Pattern-Oriented Software Architecture: A System of Patterns, Wiley, 1996, ISBN 0-471-95869-7.



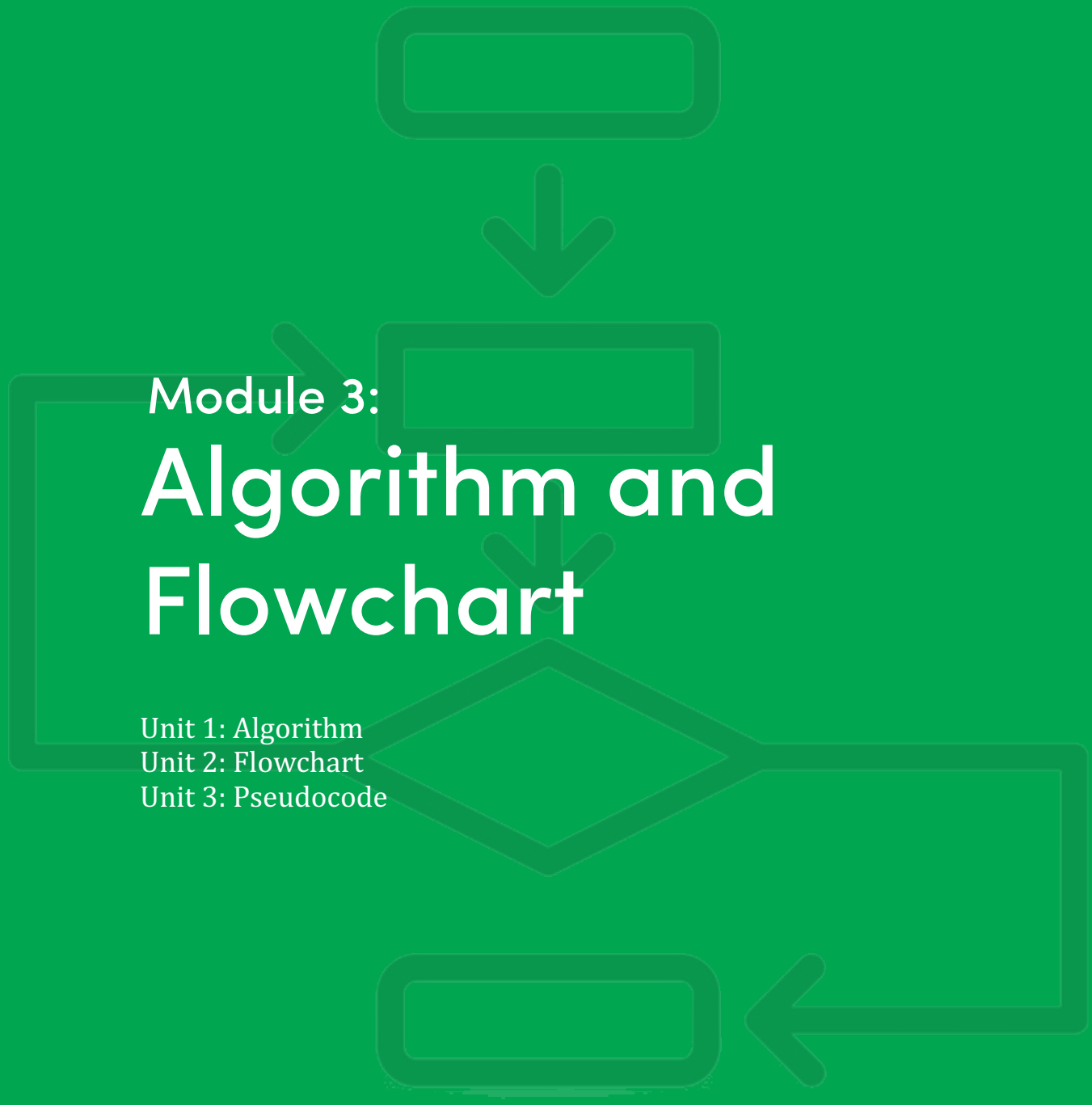
Picture: Algorithm

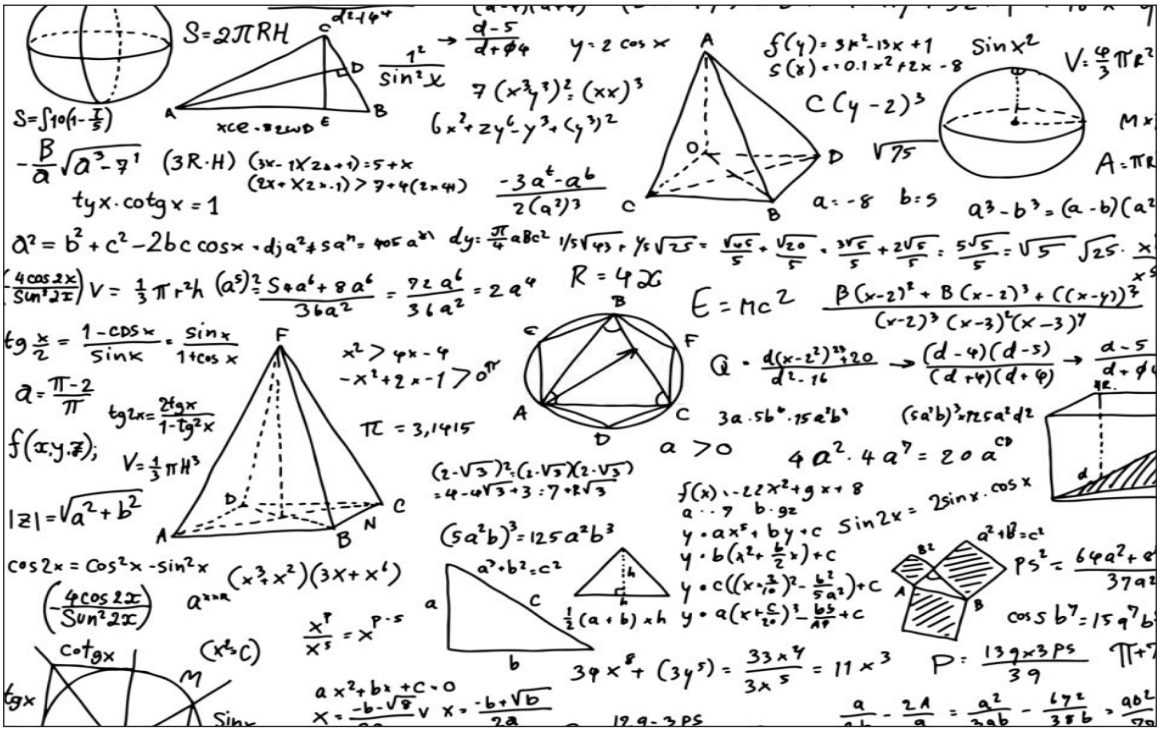
Photo Source: Pixabay

Module 3:

Algorithm and Flowchart

Unit 1: Algorithm
Unit 2: Flowchart
Unit 3: Pseudocode





UNIT 1

Algorithm

Introduction

I welcome you to another unit of this lesson; programming is the process of taking an algorithm and encoding it into a notation, a programming language so that a computer can execute it. Although it is important you know many programming languages and many different types of computers exist, the important first step is the need to have the solution. Without an algorithm, there can be no program.

It may interest you to know that computer science is not the study of programming. Programming, however, is an important part of what a computer scientist does. Programming is often the way that we create a representation of our solutions. Therefore, this language representation and the process of creating it becomes a fundamental part of the discipline. Algorithms describe the solution to a problem in terms of the data needed to represent the problem instance and the set of steps necessary to produce the intended result. Programming languages must provide a notational way to represent both the process and the data. To this end, languages provide control constructs and data types. Control constructs allow algorithmic steps to be represented in a convenient yet unambiguous way. At a minimum, algorithms require constructs that perform sequential processing, selection for decision-making, and iteration for repetitive control. As long as the language provides these basic statements, it can be used for algorithm representation

Learning Outcomes

When you have studied this unit, you should be able to:

1. Define an algorithm
2. Explain the steps involved in algorithm development
3. Explain the properties of the algorithm
4. State the types of algorithm
5. Solve some simple problems using an algorithm

Main Content

SAQ 1

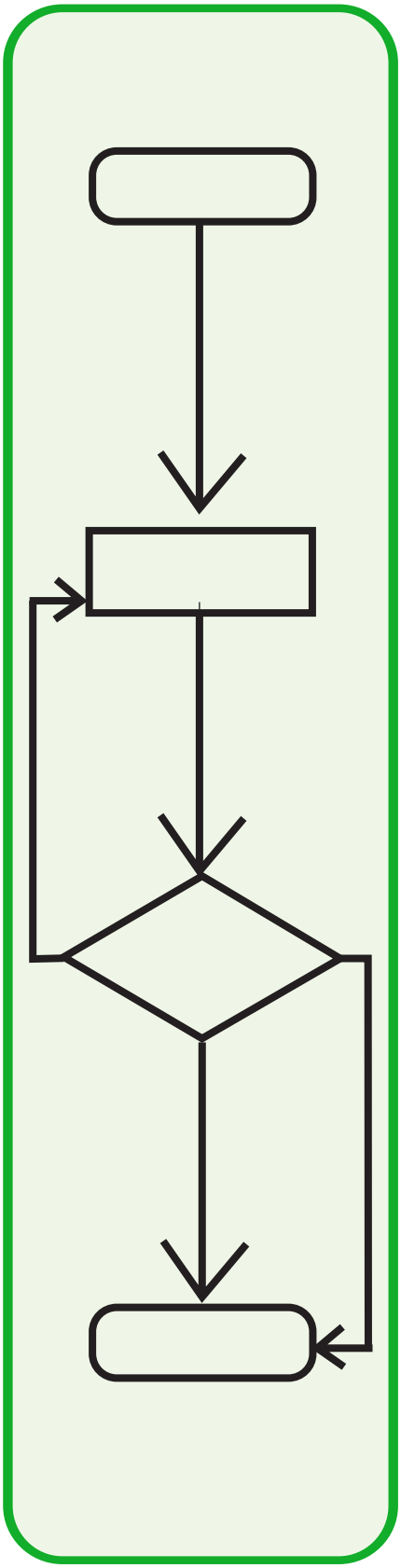
Definition

2 mins

Do you know that a set of sequential steps usually written in Ordinary Language to solve a given problem is called Algorithm? It may be possible to solve the problem in more than one way, resulting in more than one algorithm. The choice of various algorithms depends on factors like reliability, accuracy, and easy to modify. The most important factor in the choice of algorithm is the time required to execute it after writing code in High-level language with the help of a computer. The algorithm which will need the least time when executed is considered the best.

So, what is a programming algorithm? You can think of a programming algorithm as a recipe that describes the exact steps needed for the computer to solve a problem or reach a goal. We all have seen food recipes - they list the ingredients needed and a set of steps on how to make the described meal. Well, an algorithm is just like that. In computer lingo, the word for a recipe is a procedure, and the ingredients are called inputs. Your computer looks at your procedure, follows it to the letter, and you get to see the results, which are called outputs. A programming algorithm describes how to do something, and your computer will do it exactly that way every time. Well, it will, once you convert your algorithm into a language it understands!

However, it is important you note that a programming algorithm is not a computer code. It is written in simple English (or whatever the programmer speaks). It does not beat around the bush--it has a start, middle, and an end. You will probably label the first step 'start' and the last step 'end.' It includes only what you need to carry out the task. It



does not include anything unclear; often called ambiguous in computer lingo, that someone reading it might wonder about. It always leads to a solution and tries to be the most efficient solution we can think up. It is often a good idea to number the steps, but you don't have to. Instead of numbered steps, some folks use indentation and write in pseudo code, which is a semi-programming language used to describe the steps in an algorithm. But we won't use that here since simplicity is the main thing. Other folks just use a diagram called a flowchart, which we will discuss soon.

Example of the programming algorithm

Okay, you probably wish you could see an example, right? So, what exactly does an algorithm in programming look like? Well, asking a user for an email address is probably one of the most common tasks a web-based program might need to do, so that is what we will use here for an example. An algorithm can be written as a list of steps using text, or as a picture with shapes and arrows called a flowchart. We will make one of each which you will see here:

Step 1: Start

Step 2: Create a variable to receive the user's email address

Step 3: Clear the variable in case it is not empty

Step 4: Ask the user for an email address

Step 5: Store the response in the variable

Step 6: Check the stored response to seeing if it is a valid email address

Step 7: Not valid? Go back to Step 3

Step 8: End

SAQ 2

Steps involved in Algorithm development

1 min

An algorithm can be defined as "a complete, unambiguous, finite number of logical steps for solving a specific problem "

Identification of input

Step 1. Identification of input: For an algorithm, there are quantities to be supplied called input, and these are fed externally. The input is to be identified first for any specified problem.

Identification of output

Step 2: Identification of output: From an algorithm, at least one quantity is produced, called for any specified problem.

Identification of the processing operations

Step 3: Identification of the processing operations: All the calculations to be performed to lead to output from the input area to be identified in an orderly manner.

Processing Definiteness

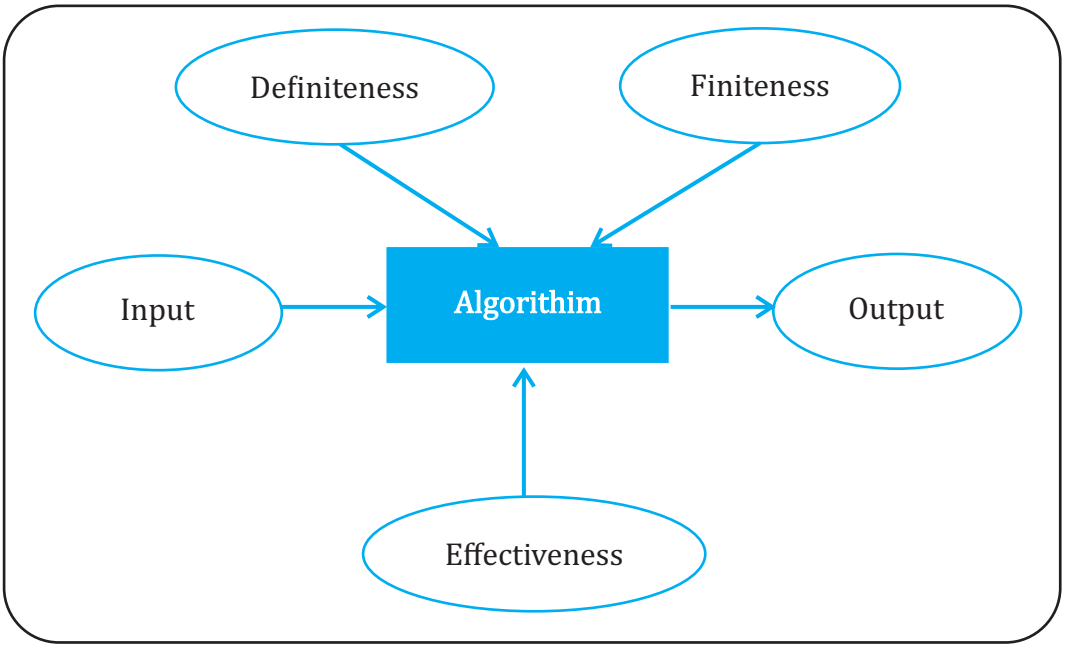
Step 4: Processing Definiteness: The instructions composing the algorithm must be clear, and there should not be any ambiguity in them.

Processing Finiteness

Step5: Processing Finiteness: If we go through the algorithm, then for all cases, the algorithm should terminate after a finite number of steps.

Possessing Effectiveness

Step 6: Possessing Effectiveness: The instructions in the algorithm must be sufficiently basic, and in practice, they can be carried out easily.



Properties of Algorithm

3 mins

An algorithm specifies a series of steps that perform a particular computation or task. It may interest you to know algorithms were originally born as part of mathematics, the word "algorithm" comes from the Arabic writer Muḥammad ibn Mūsā al-Khwārizmī, – but currently, the word is strongly associated with computer science. Algorithms resemble recipes. Recipes tell you how to accomplish a task by performing several steps. For example, to bake a cake, the steps are: preheat the oven; mix flour, sugar, and eggs thoroughly; pour into a baking pan; and so forth.

SAQ 3

However, it's very important you know that "algorithm" is a technical term with a more specific meaning than "recipe," and calling something an algorithm means that the following properties are all true. Studying algorithms is a fundamental part of computer science. There are several different characteristics of an algorithm that are useful to know: An algorithm must possess the following properties

Finiteness

If an algorithm requires its inputs (called a precondition), that requirement must be met. For example, a precondition might be that an algorithm will only

accept positive numbers as an input. If preconditions are not met, then the algorithm is allowed to fail by producing the wrong answer or never terminating.

Definiteness

You must state each step of the algorithm precisely and unambiguous. An algorithm is an unambiguous description that makes clear what has to be implemented. In a recipe, a step such as "Bake until done" is ambiguous because it does not explain what "done" means. A more explicit description such as "Bake until the cheese begins to bubble" is better. In a computational algorithm, a step such as "Choose a large number" is vague: what is large? 1 million, 1 billion, or 100? Does the number have to be different each time, or can the same number be used on every run?

Effectiveness

You must also make each step to be effective, in the sense that it should be primitive and easily convertible into a program statement) can be performed exactly in a finite amount of time. Most algorithms are guaranteed to produce the correct result. It is rarely useful if an algorithm returns the largest number 99% of the time, but 1% of the time, the algorithm fails and returns the smallest number instead.

Generality: The algorithm must be complete in itself so that it can be used to solve problems of a specific type for any input data. An algorithm is guaranteed to terminate and produce a result, always stopping after a finite time. If an algorithm could potentially run forever, it would not be very useful because you might never get an answer.

Input/output

Each algorithm must take zero, one or more quantities, as input data produce one or more output values. An algorithm can be written in English like sentences or in any standard representation sometimes. The algorithm written in English like languages is called Pseudo Code. An algorithm expects a defined set of inputs. For example, it might require two numbers where both numbers are greater than zero. Or it might require a word or a list of zero or more numbers. It might output the larger of the two numbers, an all-uppercase version of a word, or a sorted version of the list of numbers.

Example

1. Suppose we want to find the average of three numbers, the algorithm is as follows

Step 1 Start

Step 2 Read the numbers a, b, c

Step 3 Compute the sum of a, b and c

Step 4 Divide the sum by 3

Step 5 Store the result in variable d

Step 6 Print the value of d

Step 7 End of the program

Types of Algorithm

 4 mins

An algorithm is a set of self-contained sequence of instructions or actions that contains finite space or sequence, and that will result to a specific problem in a finite amount of time.

It is logical and mathematical approach for us to solve or crack a problem using any possible method. We have many types of an algorithm but the most fundamental types of the algorithm are:

1. Recursive algorithms
2. Dynamic programming algorithm
3. Backtracking algorithm
4. Divide and conquer algorithm
5. Greedy algorithm
6. Brute Force algorithm
1. Randomized algorithm

Simple Recursive Algorithm

Solves the base case directly and then recurs with a simpler or easier input every time (A base value is set at the starting for which the algorithm terminates).

We use it to solve the problems which can be broken into simpler or smaller problems of the same type.

Example

To find factorial using recursion, here is the pseudo-code:

```
Fact(x)
   If x is 0     /*0 is the base value and x is 0 is base case*/
      return 1
   return (x*Fact(x-1)) /* breaks the problem into small problems*/
```

Dynamic Programming Algorithm

I want you to know that a dynamic programming algorithm (also known as a dynamic optimization algorithm) remembers the past result and uses it to find a new result. This means it solves complex problems by breaking them down into a collection of simpler sub problems, then solving each of those sub problems only once, and storing their solution for future use instead of recomputing their solutions again.

Example

Fibonacci sequence, here is the pseudo-code:

```
Fib(n)
   if n=0
   return 0
else
   prev_Fib=0,curr_Fib=1
repeat n-1 times  /*if n=0 it will skip*/
   next_Fib=prev_Fib+curr_Fib
   prev_Fib=curr_Fib
   curr_Fib=new_Fib
return curr_Fib
```

Backtracking Algorithm

How about we learn backtracking using an example, so let's say we have a problem "Monk," and we divide it into four smaller problems "M, R, A, A." It may be the case that the solution to these problems did not get accepted as the solution of "Monk."

We did not know on which one it depends. So, we will check each one of them one by one until we find the solution for "Monk."

So basically, we attempt solving a sub problem, but if we do not reach the desired solution, undo whatever we have done and start from scratch again until we find the solution.

Example

Queens Problem

Divide and Conquer Algorithm

Divide and conquer algorithm consists of two parts; first of all, it is good you know that it divides the problems into smaller sub problems of the same type and solves them recursively and then combines them to form the solution of the original problem.

Example

Quicksort, Merge sort

Greedy Algorithm

A greedy algorithm is an algorithm that solves the problem by taking an optimal solution at the local level (without regard for any consequences) with the hope of finding an optimal solution at the global level.

We use a greedy algorithm to find the optimal solution, but you don't need to find the optimal solution by following this algorithm.

Like there are some problems for which an optimal solution does not exist (currently), these are called nondeterministic polynomial time (NP) complete problem.

Example

Huffman tree, counting money

Brute Force Algorithm

A brute force algorithm simply tries all the possibilities until a satisfactory solution is found.

We use such types of the algorithm to find the optimal (best) solution as it checks all the possible solutions.

We also use it to for finding a satisfactory solution (not the best), simply stops as soon as a solution to the problem is found.

Example

Exact string-matching algorithm

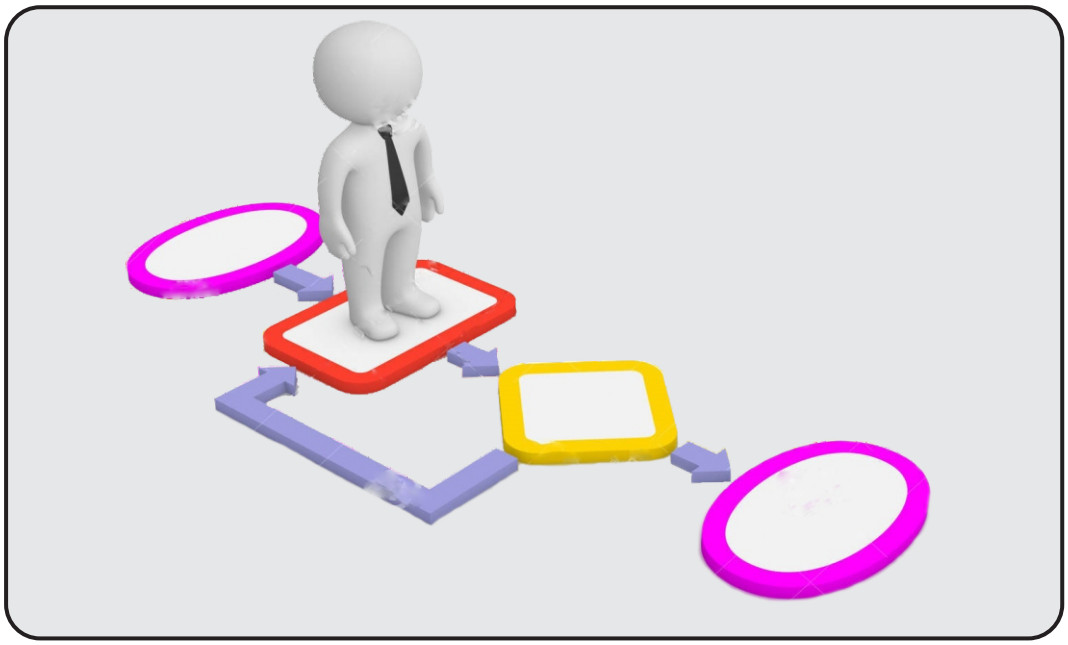
Randomized Algorithm

A randomized algorithm uses a random number at least once during the computation to make a decision.

Example

Quicksort

As we use a random number to choose the pivot point.



Algorithms for Simple Problem

2 mins

Write an algorithm for the following

SAQ 4

Write an algorithm to calculate the simple interest using the formula.

Simple interest = P*N* R/100.

Where P is principle Amount, N is the number of years, and R is the rate of interest.

Step 1: Read the three input quantities' P, N, and R.

Step 2: Calculate simple interest asSimple interest = P* N* R/100

Step 3: Print simple interest. Step 4: Stop.

Write an algorithm to find the area of the triangle.

Let b, c be the sides of the triangle ABC and A the included angle between the given sides.

Step 1: Input the given elements of the triangle, namely sides b, c, and the angle between the sides A.

Step 2: Area = (1/2) *b*C* sin A

Step 3: Output the Area

Step 4: Stop.

Write an algorithm to find the largest of three numbers

X, Y, Z. Step 1: Read the numbers X, Y, Z.

Step 2: if (X > Y) Big = X

else BIG = Y

Step 3: if (BIG < Z) Step 4: Big = Z

Step 5: Print the largest number i.e. Big

Step 6: Stop.

Write down an algorithm to find the largest data value of a set of given data values

Algorithm largest of all data values:

Step 1: LARGE 0

Step 2: read NUM

Step 3: While NUM > = 0 do

3.1 if NUM > LARGE

3.1.1 then

3.1.1.1 LARGE NUM

3.2. read NUM

Step 4: Write, "largest data value is" LARGE Step 5: end.

Write an algorithm that will test whether a given integer value is prime or not.

Algorithm prime testing:

Step 1: M 2

Step 2: read N

Step 3: MAX *SQRT (N) Step 4: While M < = MAX do

4.1 if (M* (N/M) = N

4.1.1 then

4.1.1.1 go to step 7

4.2. M M + 1

Step 5: Write "number is prime" Step 6: go to step 8

Step 7: Write "number is not a prime"

Step 8: end.

Summary

In this unit, you have learned that:

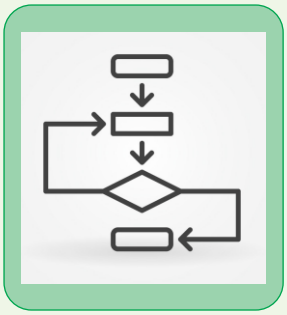
- A set of sequential steps usually written in Ordinary Language to solve a given problem is called Algorithm
- It may be possible to solve a problem in more than one way, resulting in more than one algorithm
- An algorithm specifies a series of steps that perform a particular computation or task
- Each step of the algorithm must be precisely and unambiguously stated
- The algorithm must be complete in itself so that it can be used to solve problems
- There are many types of algorithm
- An algorithm must terminate in a finite number of steps

Activity 1

Algorithm

Questions

- Define Algorithm
- What are the properties of an Algorithm
- Write an Algorithm to find the largest of three numbers



Self Assessment Questions

1. Define an algorithm?
2. State and explain the steps involved in designing an algorithm
3. Give five (5) characteristics of the flowchart
4. Use an algorithm to calculate the sum of five numbers

Tutor Marked Assessment

Use an algorithm to solve the following problems

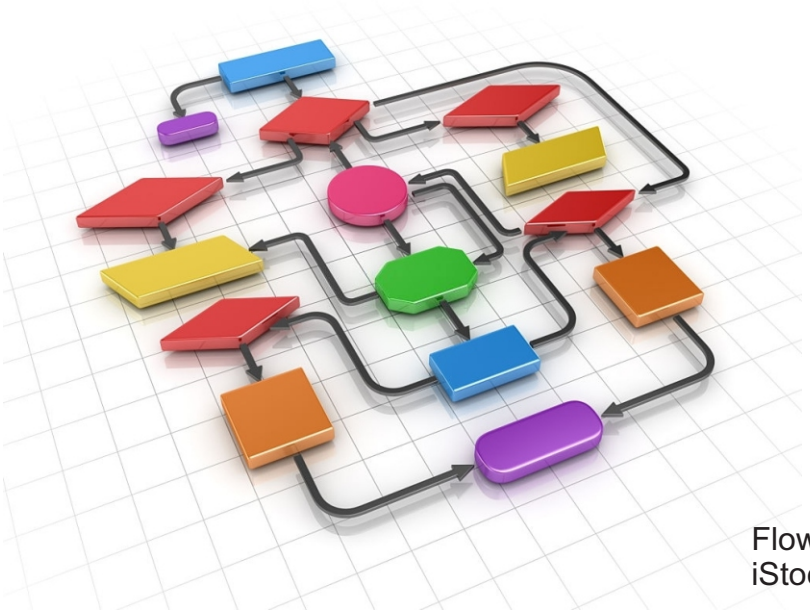
- To solve factorial of N number
- Write an algorithm that will test whether a given integer value is prime or not.

Further Reading

- Michael Trombetta QBASIC for Students / Edition 1, McGraw-Hill Higher Education, 1993
- Wallace Wang Beginning Programming for Dummies by Wallace Wang, 2001
- Selim G. Akl. The Design and Analysis of Parallel Algorithms. Prentice-Hall, 1989.
- Richard J. Anderson and Gary L. Miller. Deterministic parallel list ranking. In John H. Reif, editor, 1988 Aegean Workshop on Computing, volume 319 of Lecture Notes in Computer Science, pages 81-90. Springer-Verlag, 1988.
- Richard J. Anderson and Gary L. Miller. A simple randomized parallel algorithm for list-ranking. Unpublished manuscript, 1988.

References

- Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert E. Tarjan. Faster algorithms for the shortest path problem. Technical Report 193, MIT Operations Research Center, 1988.
- Howard H. Aiken and Grace M. Hopper. The automatic sequence controlled calculator. In Brian Randell, editor, The Origins of Digital Computers, pages 203-222. Springer-Verlag, third edition, 1982.
- M. Ajtai, J. Komlós, and E. Szemerédi. An O(n log n) sorting network. In Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, pages 1-9, 1983.



Flowchart
iStock

UNIT 2

Flowchart

Introduction

You can use a flowchart to visually present a flow of data diagrammatically through processing systems. This means seeing a flow chart, you can know the operations performed and the sequence of these operations in a system. Algorithms are nothing but a sequence of steps for solving problems. So, a flow chart can be used for representing an algorithm. A flowchart will describe the operations that are required (and in what sequence) to solve a given problem. You can see a flow chart as a blueprint of a design you have made for solving a problem. For example, suppose you are going for a picnic with your friends then you plan for the activities you will do there. If you have a plan of activities, then you know clearly when you will do what activity. Similarly, when you have a problem solving using a computer, or in other words, you need to write a computer program for a problem, then it will be good to draw a flowchart before writing a computer program. The flowchart is drawn according to defined rules.

Learning Outcomes

When you have studied this unit, you should be able to:

1. Define flowchart,
2. Explain symbols used in a flowchart,
3. Use flowchart to describe a task,
4. Solve simple problems using a flowchart

Main Content

Definition of Flowchart

1 min

The basic definition of a flowchart defines it as a diagram that represents an algorithm. Talking broadly about this, in computer science, when we have got a problem to solve, we break that it into more manageable tasks (often called modularisation). Finally, all these small parts work together to provide a solution to the original problem. Normally, when we get a problem, we tend to break it into many processes: i.e., the steps taken to arrive at the solution. A flowchart is a visual representation of these steps. Just as the word says, it visually shows the flow of a certain program. The flowchart can be thought of as the visual representation of a pseudo-code. When we have a programming problem to solve, first, we don't think about the language we are going to write the program in. We focus on the problem first, after figuring out the needed logic to solve the problem, we write a pseudo-code and/or a flowchart which shows the path we took for the solution to our problem. Then only we program it using a certain language.

SAQ 1

It may interest you to know that a flow chart is a step by step diagrammatic representation of the logic paths to solve a given problem. Or a flowchart is a visual or graphical representation of an algorithm. Flowcharts are a pictorial representation of the methods to be used to solve a given problem, and they help a great deal in analyzing the problem and planning its solution in a systematic and orderly manner. A flowchart, when translated into a proper computer language, results in a complete program.

SAQ 2

Advantages of Flowcharts

4 mins

Advantages of using Flowcharts

As we have discussed, the flow chart is used for representing an algorithm in pictorial form. This pictorial representation of a solution/system has many advantages. These advantages are as follows:

1. **Communication:** A Flowchart can be used as a better way for the communication of the logic of a system and steps involved in the solution, to all concerned, particularly to the client of the system.

2. **Effective analysis:** A flowchart of a problem can be used for an effective analysis of the problem.

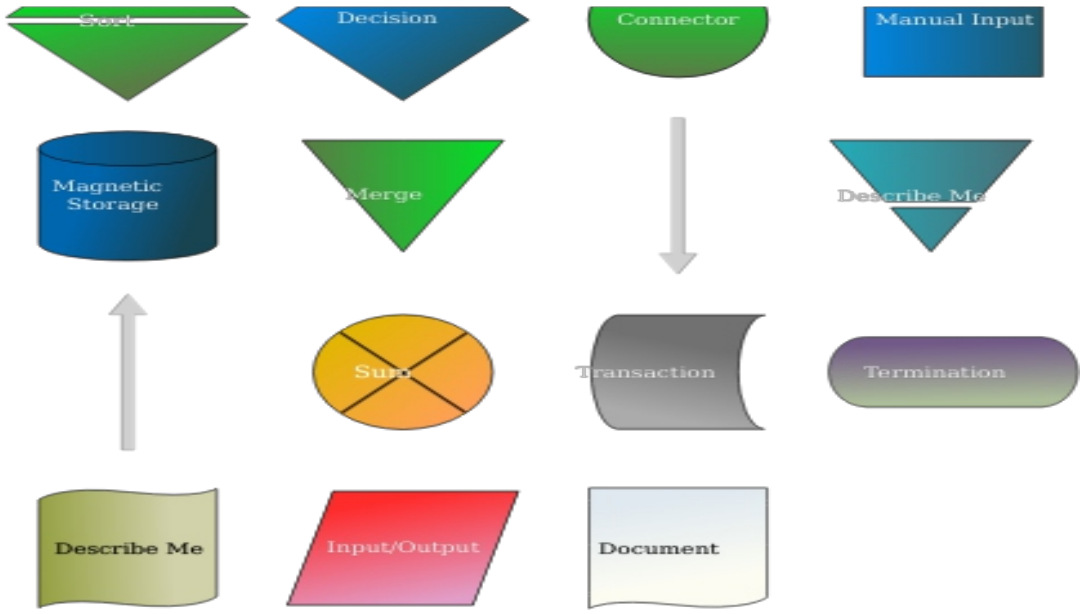
3. **Documentation of Program/System:** Program flowcharts are a vital part of good program documentation. The program document is used for various purposes, like knowing the components in the program, the complexity of the program, etc.

4. **Efficient Program Maintenance:** Once a program is developed and becomes operational, it needs time to time maintenance. With the help of the flowchart, maintenance becomes easier.

5. **Coding of the Program:** Any design of the solution of a problem is finally converted into a computer program. Writing code referring to the flowchart of the solution becomes easy.

Differences between Algorithm and Flowchart

Algorithm	Flowchart
1. A method of representing the step-by-step logical procedure for solving a problem	1. A flowchart is a diagrammatic representation of an algorithm. It is constructed using different types of boxes and symbols.
2. It contains step-by-step English descriptions, each step representing a particular operation leading to the solution of a problem	2. The flowchart employs a series of blocks and arrows, each of which represents a particular step in an algorithm
3. These are particularly useful for small problems	3. These are useful for detailed representations of complicated programs
4. For complex programs, algorithms prove to be inadequate	4. For complex programs, Flowcharts prove to be adequate



SAQ 3

Symbols we use in Flow-Charts

1 min

The shapes of the symbols we use in flowcharts are significant. Flowcharts are a common language we use to communicate processes, and it is important to be consistent in their use. This flowchart uses three different symbols to show:

- the start/endpoints of the process;
- data input or output;
- a process to be carried out.

The symbols that we use while drawing flowcharts, as given below, are as per conventions followed by the International Standard Organization (ISO).

Oval

Rectangle with rounded sides is used to indicate either the START/STOP of the program.

Input and output indicators:

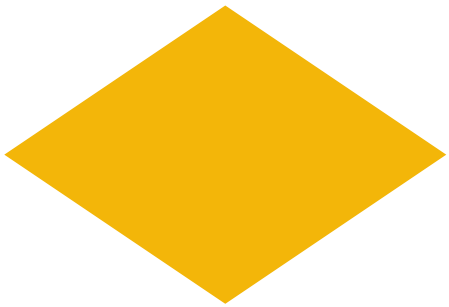
Parallelograms are used to represent input and output operations. Statements like INPUT, READ and PRINT are represented in these Parallelograms.

Process Indicators

The rectangle is used to indicate any set of processing operations, such as for storing arithmetic operations.

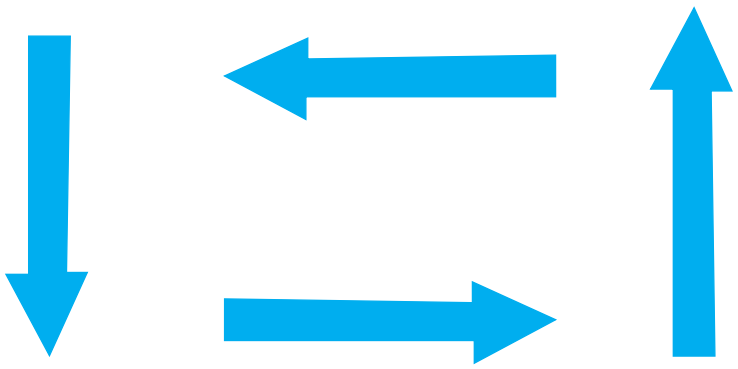
Decision Makers

The diamond is used for indicating the step of decision making and therefore known as decision box. Decision boxes are used to test the conditions or ask questions, and depending upon the answers, the computer takes the appropriate actions. The decision box symbol is



Flow Lines

Flow lines indicate the direction being followed in the flowchart. In a Flowchart, every line must have an arrow on it to indicate the direction. The arrows may be in any direction



On-Page Connectors

We use circles to join the different parts of a flowchart, and these circles are called on-page connectors. The uses of these connectors give a neat shape to the flowcharts. In complicated problems, a flowchart may run into several pages. The parts of the flowchart on different pages are to be joined with each other. The parts to be joined are indicated by the circle.

Off-page Connectors

SAQ 4

This connector represents a break in the path of a flowchart, which is too large to fit on a single page. It is similar to an on-page connector. The connector symbol marks where the algorithm ends on the first page and where it continues on the second.

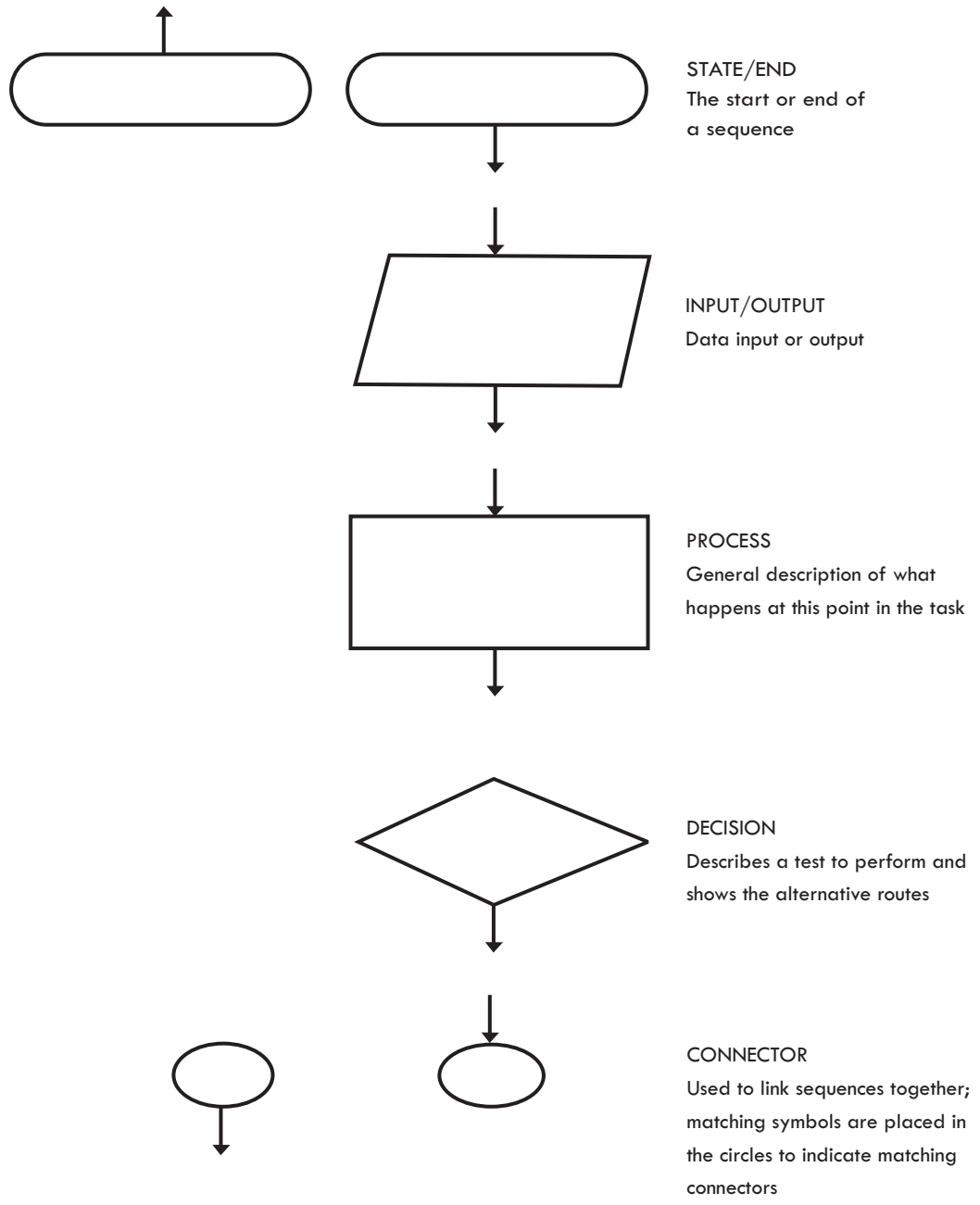


Figure 1: Sybmols used in Flowchart

Using Flowcharts to describe a task

3 mins

We design application programs to perform specific tasks. These tasks range from the relatively simple to the extremely complex. In this section, you will look at what is involved in planning a program to perform some simple tasks. To write a program, the task the program will perform has to be first written as a list of actions. The actions have to be given in an order that will ensure the task is carried out successfully.

Example One: Write down, in order, the list of actions that the computer inside the scales has to carry out to show an object's weight on display.

Discussion

The computer has to:

1. accept data from the sensor that measures the displacement of the scalepan;

2. transform the data from the sensor into data for the display;

3. send the display-formatted data to the display.

The list of actions, for example, one can be shown diagrammatically in a type of diagram called a flowchart. Figure 2 shows how I have written this simple sequence of actions as a flowchart.

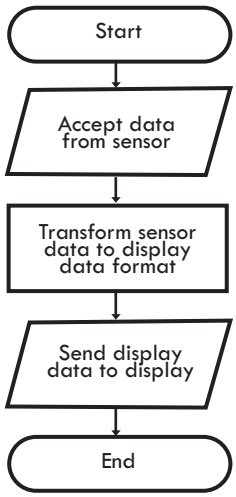


Figure 2: Flowchart for the display of the sensor

The next example I want us to look into incorporates a decision box. It involves a slightly more complex set of electronic scales: they have one additional button on the front that allows the user to select whether the weight is displayed in grams or pounds and ounces. A flowchart incorporating this choice of display format is shown in figure 3. It uses the decision box to make a choice about which piece of the program will be run. There are two exit routes from the decision box; each route is called a branch. If the user has requested that the weight should be displayed in grams, the Yes branch is followed so that the sensor data can be transformed into gram format. If the user has requested that the weight should be displayed in pounds and ounces, the No branch is followed to transform the sensor data to pounds and ounces instead. Note that once the translation to the selected output format is complete, the branches of the flowchart come together again, and the 'send to display' part of the task is run regardless of which branch was taken earlier in the program.

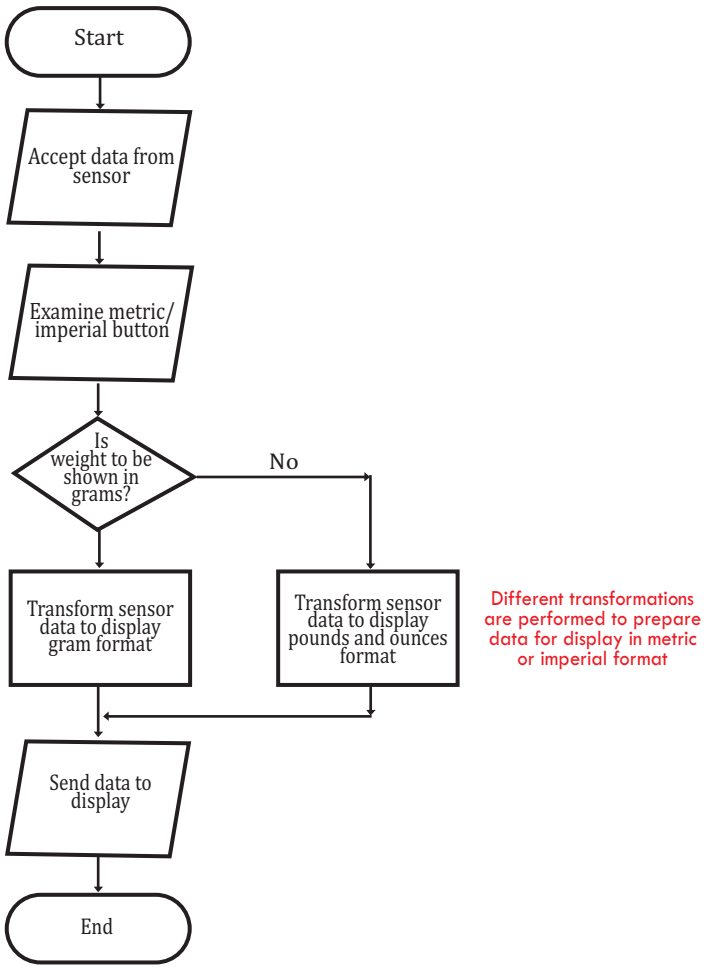


Figure 3: Flowchart incorporating this choice of display format

Now I want you to consider what happens when you weigh, for example, flour on a set of scales. You slowly add more flour to the scale pan until you reach the desired weight. As you do this, the display constantly changes, showing the weight increasing as you add more flour. To do this, the scales' computer must repeatedly examine the input and update the display each time it does so. The first two flowcharts do not implement this. They simply take one reading and need to be restarted to take another.

The flowchart can be changed so that the input is repeatedly examined, and the output repeatedly displayed utilizing what is called a loop. A loop allows a certain part of a flowchart to be carried out as many times as necessary, depending on the results of a decision. In this instance, after every 'sends to the display,' the flowchart in Figure 18 shows a loop back to examine the sensor again, provided the scales have not been switched off. Note that a loop has to start as a branch from a decision box. Spend a few moments examining the loop below; make sure that you understand how this flowchart differs from the one before and why it would enable the scales to display an increasing weight as, for example, flour was slowly added to the scale pan.

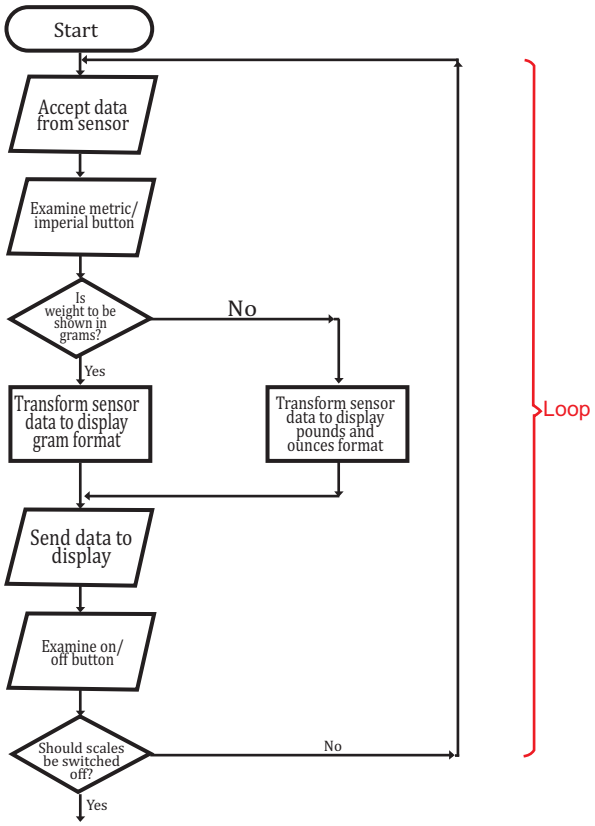


Figure 4: Flowchart for the Sensor Display

Simple Problems using Flow Chart

Draw the Flow chart for the following

Draw the Flowchart to find Roots of Quadratic equation $ax^2 + bx + c = 0$. The coefficients a, b, c are the input data

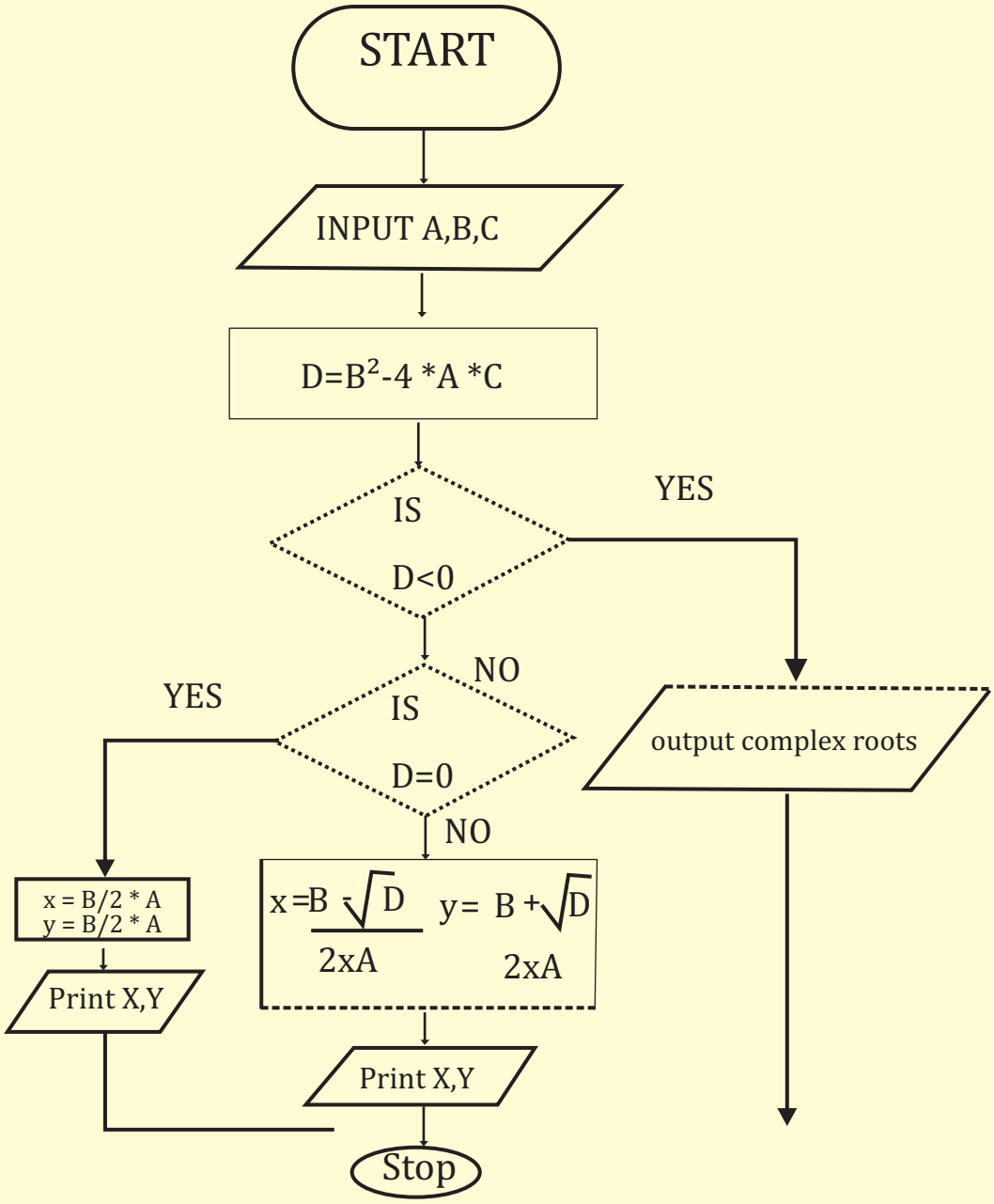


Figure 6: Flowchart for Solving Quadratic Equation

Draw a flowchart for adding the integers from 1 to 100 and to print the sum.

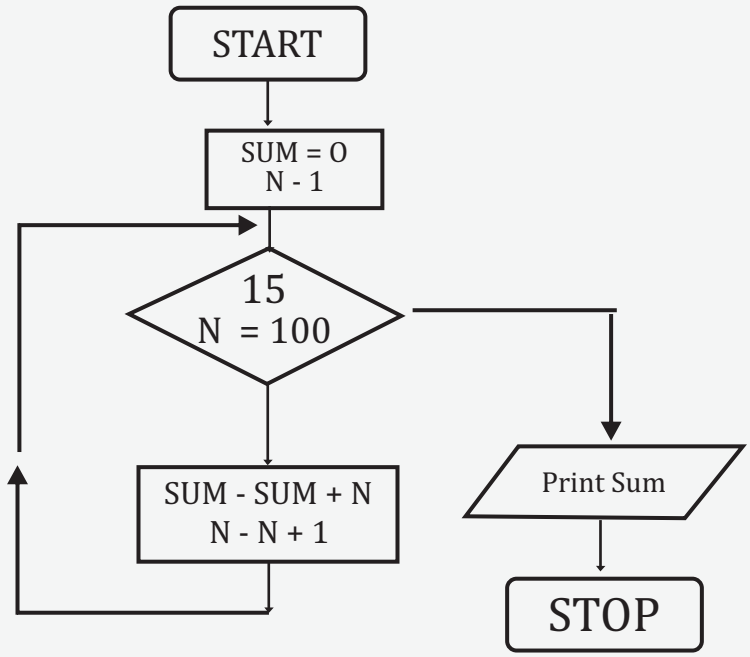


Figure 7: Flowchart for adding integers from 1 to 100

Develop a flowchart to illustrate how to make a Land phone telephone call

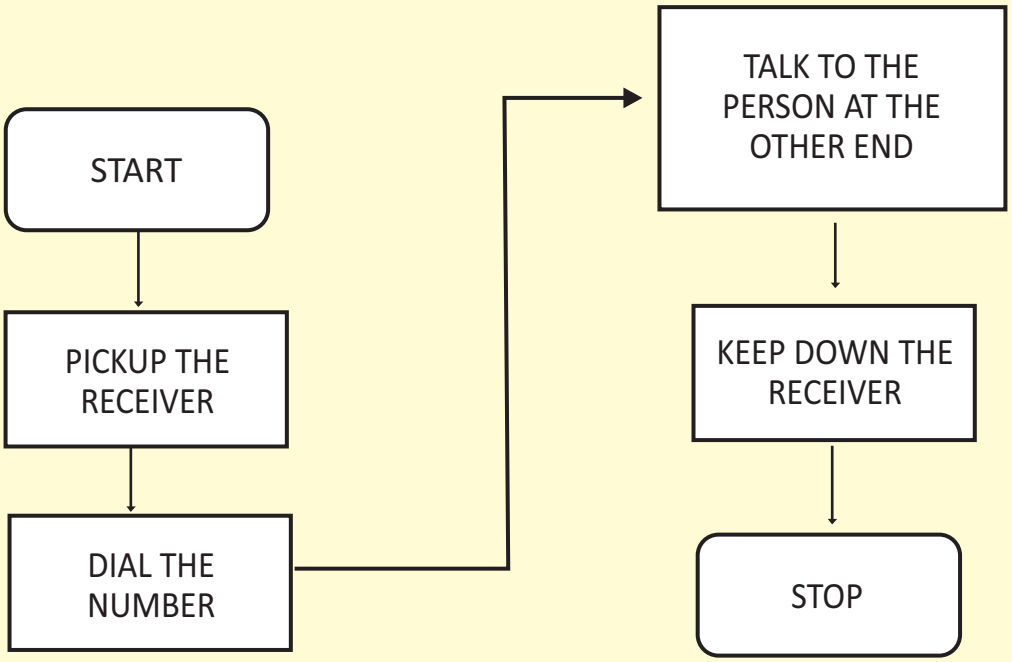
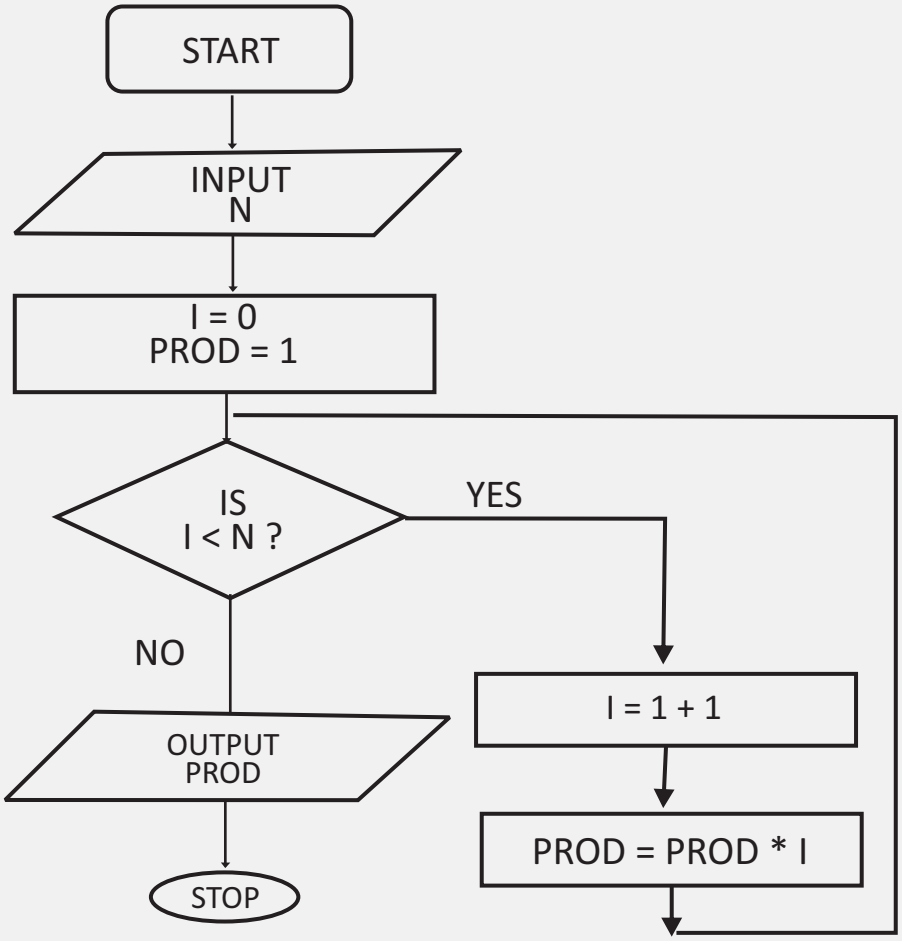


Figure 8: Flowchart for making a land phone call

Draw a flowchart to find the factorial of given positive integer N.



Flowchart for find the factorial of given positive integer N

Summary

In this unit, you have learned that:

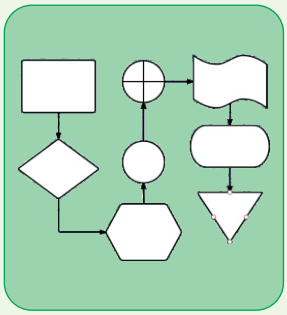
- A flow chart is a step by step diagrammatic representation of an algorithm to solve a given problem
- A flowchart is a visual or graphical representation of an algorithm
- A flowchart can be used for effective analysis of a problem
- A flowchart uses three different symbols
- Any tasks range from the relatively simple to the extremely complex

Activity 1

Flowchart

Questions

- Define a flowchart
- With the aid of diagrams, briefly discuss the symbols used in flowchart
- What are the differences between an Algorithm and a Flowchart



Self Assessment Questions

1. Define a flowchart?
2. Give five (5) advantages of a flowchart
3. State and explain the symbols used in a flowchart
4. Use a flowchart to describes how to cook yam

Tutor Marked Assessment

Use a flowchart to solve the following:

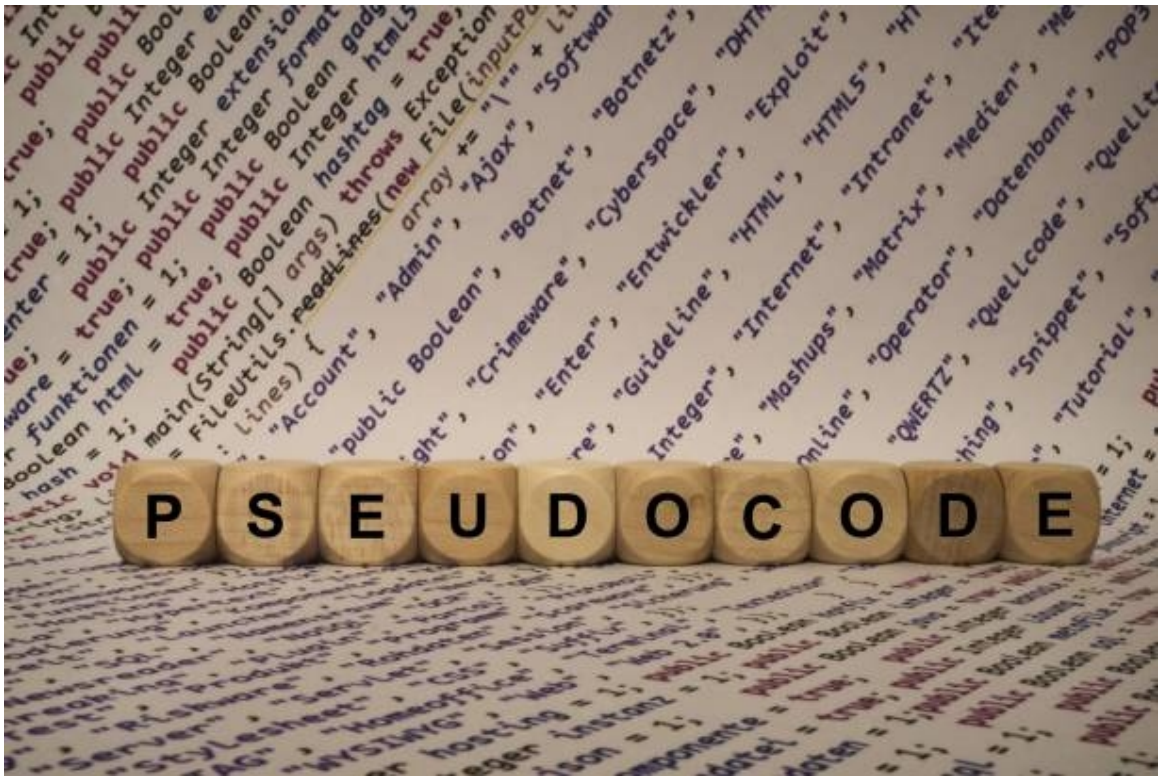
- Design a flowchart to find the sum of the first 100 natural numbers.
- Create a flowchart to find the largest of three numbers x, y, and z. Draw a flowchart to find out the biggest of the three unequal positive numbers.

Further Reading

- Michael Trombetta QBASIC for Students / Edition 1, McGraw-Hill Higher Education, 1993
- Wallace Wang Beginning Programming for Dummies by Wallace Wang, 2001
- Selim G. Akl. The Design and Analysis of Parallel Algorithms. Prentice-Hall, 1989.
- Richard J. Anderson and Gary L. Miller. Deterministic parallel list ranking. In John H. Reif, editor, 1988 Aegean Workshop on Computing, volume 319 of Lecture Notes in Computer Science, pages 81-90. Springer-Verlag, 1988.
- Richard J. Anderson and Gary L. Miller. A simple randomized parallel algorithm for list-ranking. Unpublished manuscript, 1988.

References

- Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert E. Tarjan. Faster algorithms for the shortest path problem. Technical Report 193, MIT Operations Research Center, 1988.
- Howard H. Aiken and Grace M. Hopper. The automatic sequence controlled calculator. In Brian Randell, editor, The Origins of Digital Computers, pages 203-222. Springer-Verlag, third edition, 1982.
- M. Ajtai, J. Komlós, and E. Szemerédi. An O (n log n) sorting network. InProceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, pages 1-9, 1983.



UNIT 3

Pseudocode

Introduction

In this unit, I will introduce you to pseudo-code, how it can be used to solve a real-world problem, and give examples. Pseudocode is not an algorithm, flowchart, or program itself; it is just a way of solving a problem using English like a statement to provide a solution to a situation. It is similar to every day English (in shorthand form), convenient and user friendly, although it is not an actual computer programming language. Pseudo codes are not executed on computers but rather help the programmer "think out" a program before attempting to write it in a programming language such as Basic, Visual Basic, Java, C, C++. The steps in writing a pseudo code are similar to those required to construct a flow chart.

Learning Outcomes

When you have studied this unit, you should be able to:

1. Explain pseudo code,
2. State at least five (5) advantages and three (3) disadvantages of pseudocode,
3. Solve problems using an algorithm, flowchart, and pseudocode

Main Content

Pseudocode

3 mins

SAQ 1

It is very important you bear in mind that the Pseudocode is neither an algorithm nor a program. It is an abstract form of a program. It consists of English like statements which perform the specific operations. It is defined as an algorithm. It does not use any graphical representation. In pseudo-code, the program is represented in terms of words and phrases, but the syntax of the program is not strictly followed. The informal level, high-level description of the operating principle of a computer program or other algorithm. You should bear in mind it uses the structural conventions of a programming language but is intended for human reading rather than machine reading. Pseudocode is an informal language that helps programmers develop algorithms.

Advantages and Disadvantages of Pseudocode

Programming can be a complex process when the requirements of the program are complex. The pseudo-code provides a simple method of developing the program logic as it uses every language to prepare a brief set of instructions in the order in which they appear. In the completed program, it allows the programmers to focus on the steps required to solve a

problem rather than on how to use the computer language. Some of the most significant benefits of the Pseudocode that I will want you to take note of includes:

- Since it is a language-independent, it can be used by most programmers; it allows the developer to express the design in plain and natural language.
- It is easier to develop a program from a pseudo-code as compared to the flow chart. Programmers do not have to think about syntax; we simply have to concentrate on the underline logic. The focus is on the steps to solve a problem rather than how to use the computer language.
- Often it is easy to translate pseudocode into a programming language, a step which can be accomplished by less experienced
- The uses of words and phrases in pseudo-code, which are in the lines of basic computer operations, simplify the translation from the pseudo-code algorithm to the specific programming language.
- Unlike flow charts, pseudo code is at and does not tend to run over many pages. Its simple structure and readability make it easier to modify.

You should understand the pseudocode allows programmers to work in different computer languages to talk to others; they can be reviewed by groups easier than the real code.

Disadvantages of Pseudocode

- Although the pseudo-code is a very simple mechanism to specify problem-solving logic, it has some of the limitations that are listed below:
- The main disadvantages are that it does not provide a visual representation of the programming logic.
- There are no accepted standards for writing the pseudo-code. Programmers use their styles of writing pseudocode.

The pseudo-code cannot be compiled nor executed, and there is no real formative of a syntax of rules. It is simply one step, an important one, in producing the final code.

Actions in pseudo-code

1. **Begin Action**

We use it to start a pseudo code. The general form is Begin or Start repetition

2. **Read action/ input action**

For entering values or data or input needed in solving a problem. The general form is

READ A, B

Or

INPUT A, B

3. **Repetition Action**

We use it to handle many values in code, i.e., repetition. Features like a while-do statement repeat until it can be used to carry out repetition action.

4. **Decision Action**

It is achieved by the use of if-then, else, end-if statement within an algorithm in other to make certain decisions based on some conditions.

If (Condition) Then Action

Else

End if

5. **Print Action**

Alternate action

We use it to indicate those values or outputs that are important to be recorded. Its general form is the PRINT output list. An output list may contain many variables separated by commas, m bn bn bje.g. PRINT A, B, C, D, Y, Z OR PRINT" John is a good boy."

6. **Stop Action**

Since all pseudo-code must halt, it becomes necessary to include a statement which indicates when the termination takes place. Its general form is STOP. Although STOP statement may appear many times in a program

Examples

1. Write pseudocode to sum two numbers together and output the result.

Step 1: Begin

Step 2: READ A, B

Step 3: C = A + B

Step 4: PRINT C

Step 5: stop

2. Write pseudocode to multiply two numbers and output the result if it is greater than 40

Step 1: Begin Step 2: READ A, B

Step 3: product = A * B

Step 4: if a product is > 40 THEN Step 5 ELSE Step 6 Step

5: PRINT product

Step 6: stop

3. Write pseudocode to convert a temperature in Fahrenheit to centigrade using the formula

c=(f-32) * 5/9 Begin

Read f

c=(f-32) * 5/9

PRINT" the temperature in Fahrenheit is," f

PRINT "the temperature in centigrade is," c

Stop

4. Write pseudocode to determine if a student passes an exam or not suppose the passing grade in the exam is 60

Begin

Read student's grade

If student's grade >= 60 then

Pass

Stop

Else End if

Example:

Write a pseudocode to perform the basic arithmetic operations.

Read n1, n2

Sum = n1 + n2

Diff = n1 – n2

Mult = n1 * n2

Quot = n1/n2

Print sum, diff, mult, quot

End.

Advantages and Disadvantages of Pseudocode

SAQ 2

1 min

It is important you note the advantages of pseudocode as listed below

1. It can be easily modified.
2. It gives better insight and clarity than a flowchart.
3. It can be easily written, read, and understood easily.
4. Ease of conversion to a real programming language than a flowchart.

it is important you note the disadvantages of pseudocode as listed below

1. It is not visual
2. There is no standardized style or format for pseudocode writing
3. It creates an additional level of documentation to maintain.
4. High error possibilities in converting to programming codes.

SAQ 3

Examples of Flowchart, Algorithm and Pseudocodes

2 mins

Example 1.

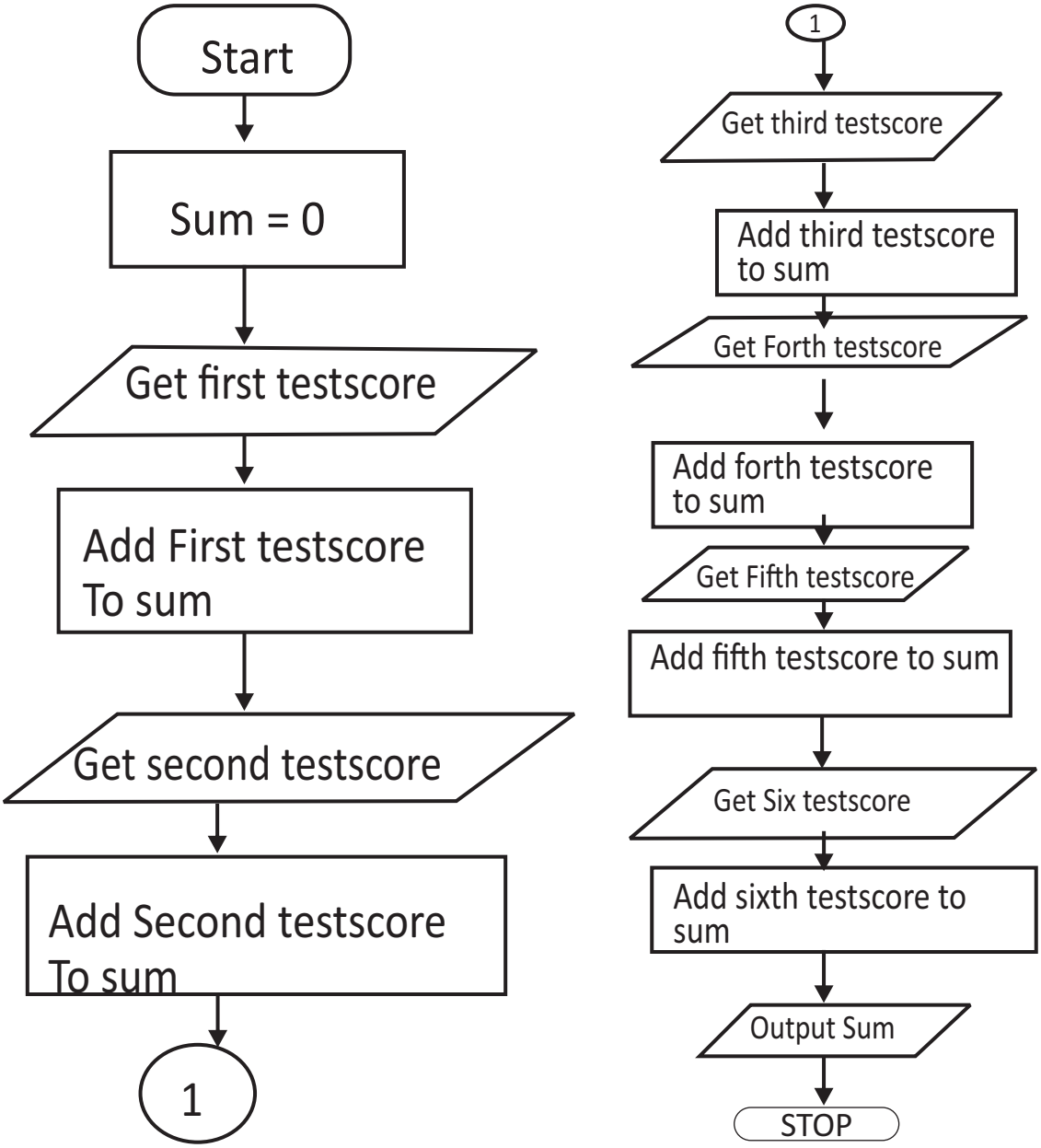
Design an algorithm and the corresponding flowchart for adding the test scores as given below:

26, 49, 98, 87, 62, 75

Algorithm

1. Start
2. Sum = 0
3. Get the first test score
4. Add the first test score to sum
5. Get the second test score
6. Add to sum
7. Get the third test score
8. Add to sum
9. Get the Forth test score
10. Add to sum
11. Get the fifth test score
12. Add to sum
13. Get the sixth test score
14. Add to sum
15. Output the sum
16. Stop

b) The corresponding flowchart is as follows:



I want you to note that the algorithm and the flowchart above illustrate the steps for solving the problem of adding six test scores. Where one test score is added to sum at a time, both the algorithm and flowchart should always have a **Start** step at the beginning of the algorithm or flowchart and at least one **stop** step at the end, or anywhere in the algorithm or flowchart. Since we want the sum of six test scores, then we should have a container for the resulting sum. In this example, the container is called **sum,** and we make sure that the sum should start with a zero value by step 2.

Example 2
Write an algorithm and draw a flowchart to convert the length in feet to a centimetre.

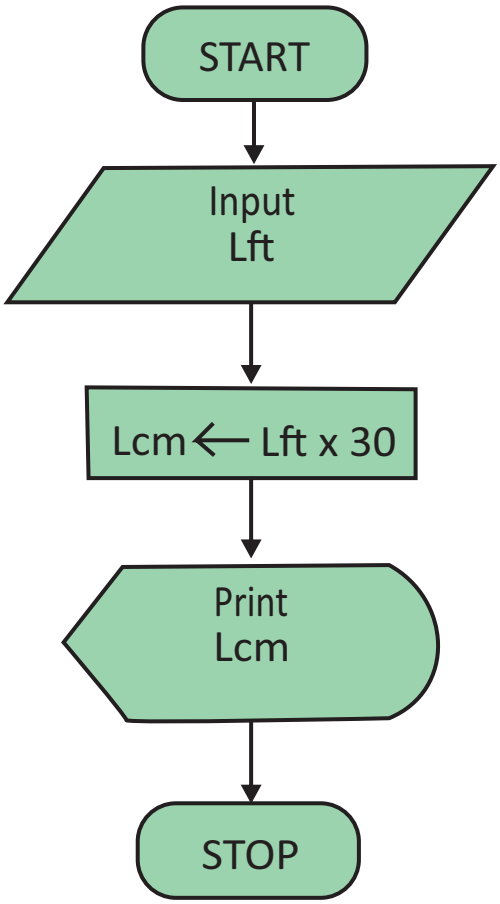
Pseudocode:

- Input the length in feet (Lft)
- Calculate the length in cm (Lcm) by multiplying LFT with
- Print length in cm (LCM)

Algorithm

- Step 1: Input Lft
- Step 2: Lcm → Lft x 30
- Step 3: Print Lcm

Flowchart



Example 3

Write an algorithm to determine a student's final grade and indicate whether it is pass or fail. The final grade is calculated as an average of four marks.

Pseudocode:

- Input a set of 4 marks
- Calculate their average by summing and dividing by 4
- if the average is below 50 Print "FAIL."

else

Print "PASS"

Algorithm

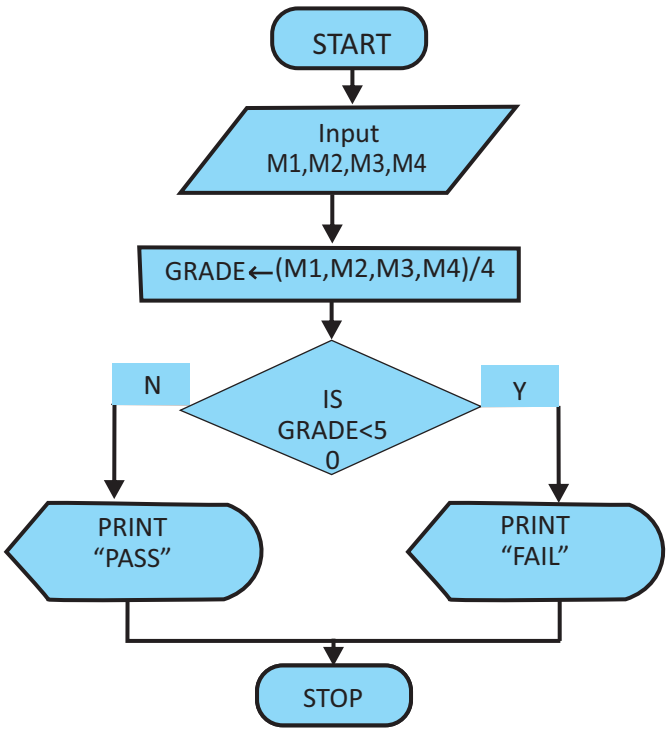
- Step 1: Input M1,M2,M3,M4
- Step 2: GRADE ¬ (M1+M2+M3+M4)/4
- Step 3: if (GRADE < 50), then Print "FAIL."

else

Print "PASS"

Endif

Flowchart



Example 4

Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

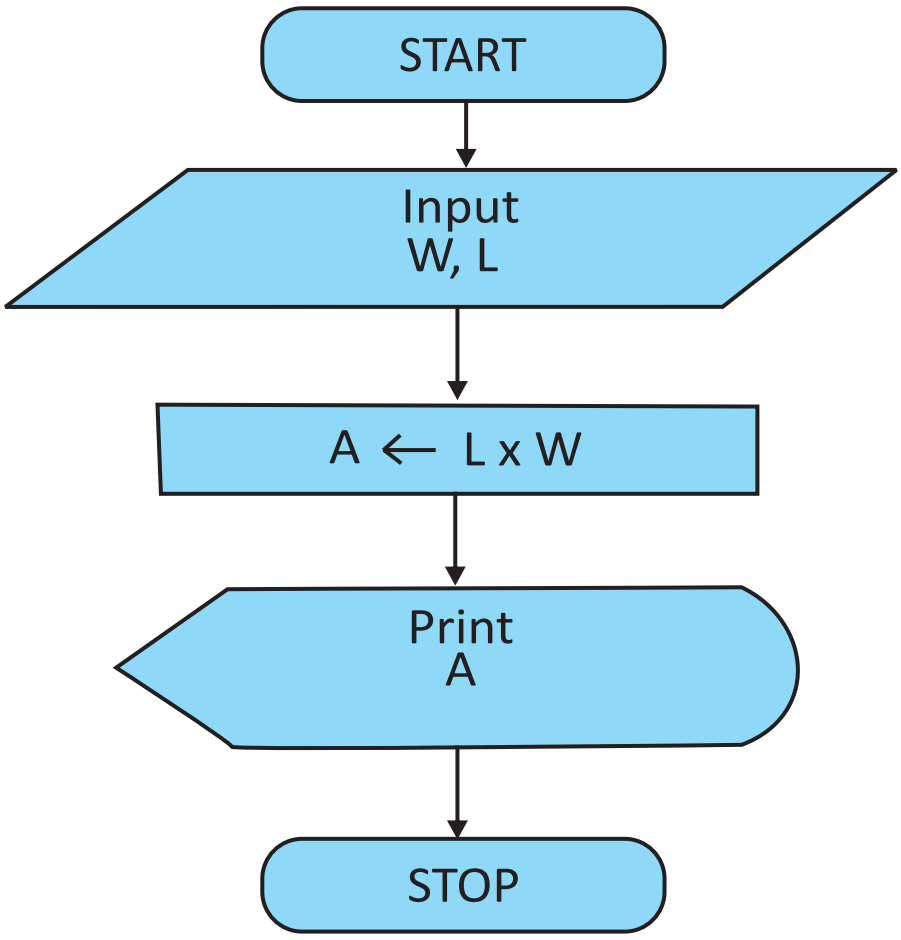
Pseudocode

- Input the width (W) and Length (L) of a rectangle
- Calculate the area (A) by multiplying L with W
- Print A

Algorithm

- Step 1: Input W,L
- Step 2: A = L x W
- Step 3: Print A

Flowchart



Example 5

Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation

$ax^2 + bx + c = 0$

Hint: d = sqrt (b^2 - 4ac), and the roots are: x1 = (–b + d)/2a and x2 = (–b – d) / 2a

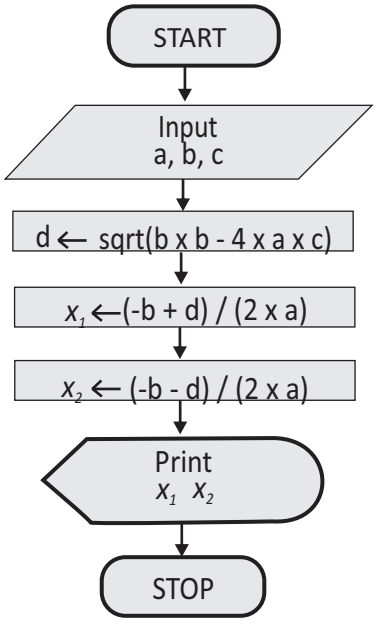
Pseudocode:

- Input the coefficients (a, b, c) of the quadratic equation
- Calculate **d**
- Calculate **x1**
- Calculate **x2**
- Print x1 and x2

Algorithm:

- Step 1: Input a, b, c
- Step 2: d =sqrt ( )
- Step 3: x1 = (–b + d) / (2 x a)
- Step 4: x2 = (–b – d) / (2 x a)
- Step 5: Print x1, x2

Flowchart

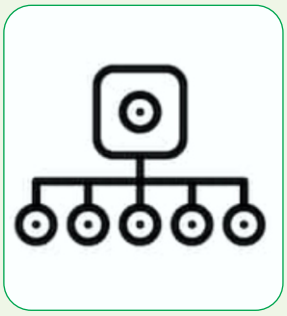


Summary

In this unit, you have learned that:

- Pseudocode is neither an algorithm nor a program
- Pseudocode consists of English like statements which perform the specific operations
- Pseudocode does not use any graphical representation

Activity 1



Questions

- Define pseudocode
- Mention three advantages and three disadvantages of pseudocode
- Give two examples of pseudocode

Pseudocode

Self Assessment Questions

1. Define Pseudocode?
2. State five (5) advantages of pseudo-code
3. Draft a pseudo code for cooking rice

Tutor Marked Assessment

Use an algorithm to solve the following problems

- Use pseudocode to solve the following:
- To find the sum of the first 100 natural numbers.
- To find the largest of three numbers x, y, and z. Draw a flowchart to find out the biggest of the three unequal positive numbers.

Further Reading

Michael Trombetta QBASIC for Students / Edition 1, McGraw-Hill Higher Education, 1993

Wallace Wang Beginning Programming for Dummies by Wallace Wang, 2001

Selim G. Akl. The Design and Analysis of Parallel Algorithms. Prentice-Hall, 1989.

Richard J. Anderson and Gary L. Miller. Deterministic parallel list ranking. In John H. Reif, editor, 1988 Aegean Workshop on Computing, volume 319 of Lecture Notes in Computer Science, pages 81-90. Springer-Verlag, 1988.

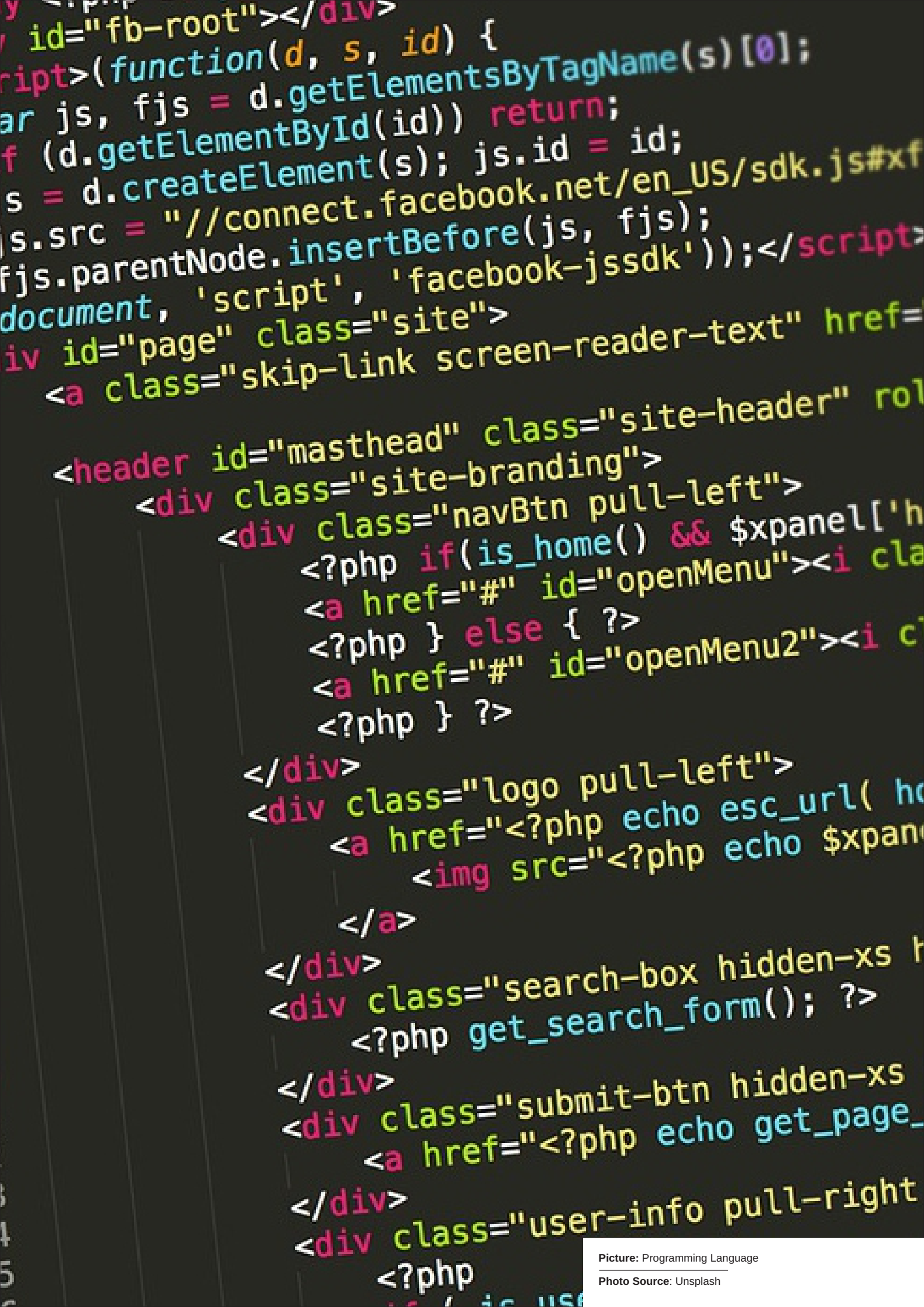
Richard J. Anderson and Gary L. Miller. A simple randomized parallel algorithm for list-ranking. Unpublished manuscript, 1988.

References

Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert E. Tarjan. Faster algorithms for the shortest path problem. Technical Report 193, MIT Operations Research Center, 1988.

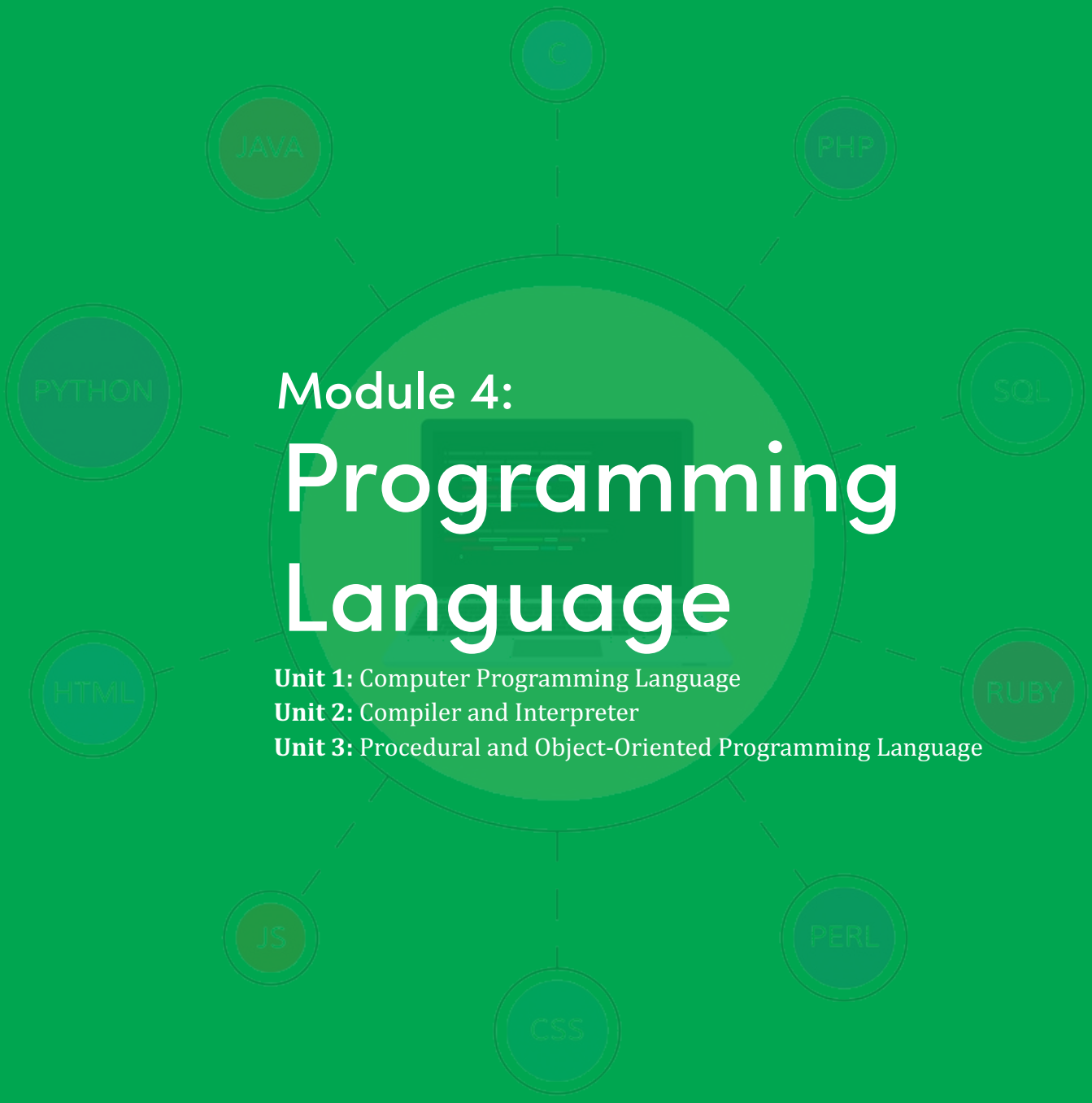
Howard H. Aiken and Grace M. Hopper. The automatic sequence controlled calculator. In Brian Randell, editor, The Origins of Digital Computers, pages 203-222. Springer-Verlag, third edition, 1982.

M. Ajtai, J. Komlós, and E. Szemerédi. An O (n log n) sorting network. InProceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, pages 1-9, 1983.



Picture: Programming Language

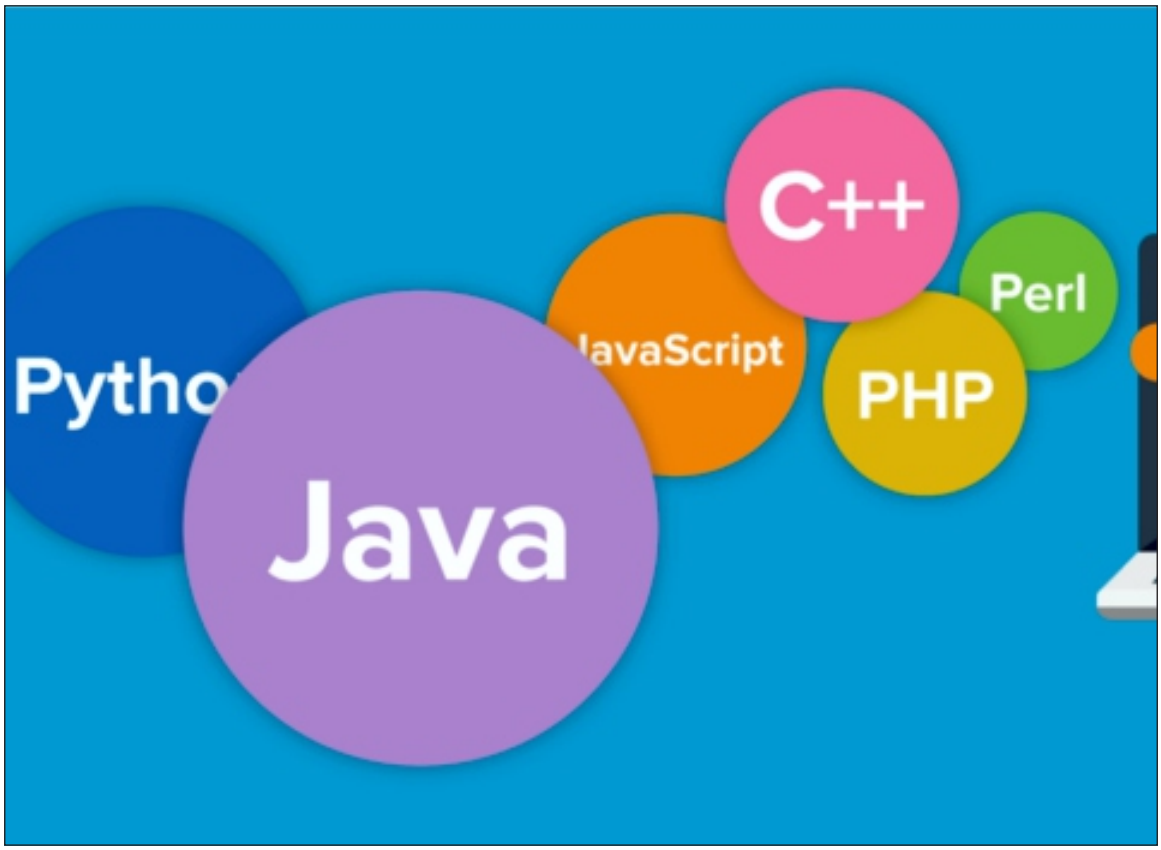
Photo Source: Unsplash



Module 4:

Programming Language

Unit 1: Computer Programming Language
Unit 2: Compiler and Interpreter
Unit 3: Procedural and Object-Oriented Programming Language



UNIT 1

Computer Programming Language

Introduction

I welcome you to this unit; I want you to know a program is a set of instructions following the rules of the chosen language. Without programs, computers are useless. A program is like a recipe. It contains a list of ingredients (called variables) and a list of directions (called statements) that tell the computer what to do with the variables.

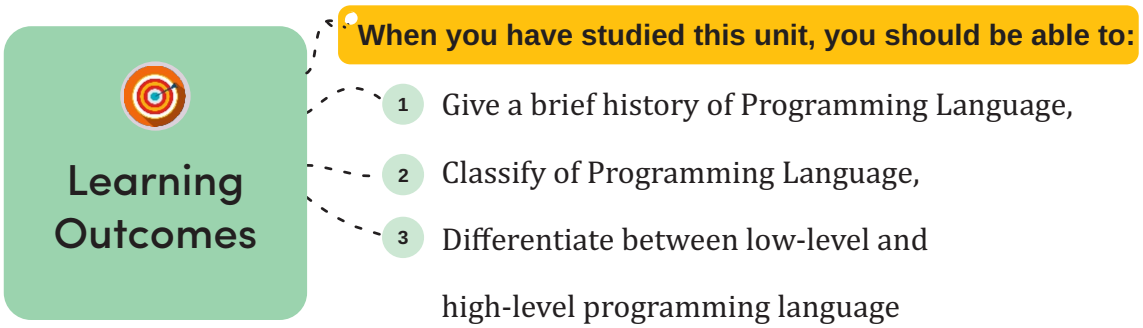
We use programming languages to create programs. A computer program is intended to perform some specific tasks through a computer or to control the behaviour of a computer. It is a vocabulary and set of grammatical rules (syntax) for instructing a computer to perform specific tasks. Programming languages can be used to create computer programs.

Using a programming language, we write instructions that the computer should perform. Instructions are usually written using characters, words, symbols, and decimal. These instructions are later encoded to the understandable computer language, i.e., binary language so that the computer can understand the instructions given by humans and can perform a specified task.

Thousands of programming languages have been created, and many are still being developed every year. Every programming language is designed for some specific purpose. FORTRAN, OCaml, Haskell are best suited for scientific and numerical computations. Whereas Java, C++, C# are best suited for designing server applications, games, desktop applications, and many more.

All data items in the computer are represented as strings of binary digits. To give these strings meaning, we need to have data types. Data types provide an interpretation for this binary data so that we can think about the data in terms that make sense concerning the problem being solved. These low-level, built-in data types (sometimes called the primitive data types) provide the building blocks for algorithm development. For example, most programming languages provide a data type for integers. Strings of binary digits in the computer's memory can be interpreted as integers and given the typical meanings that we commonly associate with integers (e.g. 23, 654, and −19). Also, a data type describes the operations that the data items can participate in. With integers, operations such as addition, subtraction, and multiplication are common. We have come to expect that numeric types of data can participate in these arithmetic operations.

The difficulty that often arises for us is the fact that problems and their solutions are very complex. These simple, language-provided constructs and data types, although certainly sufficient to represent complex solutions, are typically at a disadvantage as we work through the problem-solving process. We need ways to control this complexity and assist with the creation of solutions.



Main Content

SAQ 1

History of Programming Language

2 mins

The programming language is the language of computers. Through programming language, we can communicate with a computer system. Computers can only understand binary, but humans are not comfortable with the binary number system. Humans cannot interact fluently with computers in the language of 0's and 1's. Programming language acts as an interface between computers and humans.

Earlier, when there was no concept of programming languages, computer instructions were directly given to the computer in decimal or binary form. These instructions were given through punch cards, magnetic tapes, or switches. Later on, when the computer started growing, more and more programs were written day-by-day. Writing programs completely in binary was cumbersome and error-prone. Therefore, we developed various mnemonics for different instructions. These mnemonics are in a human-readable format. Such as ADD for adding values of two registers, JMP for conditional controls. Now, the programs were written using mnemonics and decimal values known as assembly language.

Mnemonics are short abbreviated English words used to specify computer instruction. Each instruction in binary has a specific mnemonic. They are architecture-dependent, and there is a list of separate mnemonics for different computer architectures.

Mnemonics gave relief to the programmers from writing programs directly in binary

language. However, it was still a tedious work to remember the complete list of mnemonics for various computer architectures.

During the 1950s the first high-level programming language Plankalkül was written. As the computer started expanding from scientific to business and many other fields, many more high-level programming languages were written for various specific purposes. Unlike binary and assembly, programs in high-level languages were written in English like statements. High-level languages were programmer-friendly, less error-prone, easy to write and maintain.

High-level languages were like a magic wand to the programmers. However, they lacked the facility to perform several low-level activities. This gave birth to several other programming languages with different paradigms. During the 1960s to 1980s, several popular programming languages were written for specific purposes. This was the phase when the programming languages were more influenced. Popular languages developed during this period were ALGOL, Lisp, C, Prolog, etc. Languages that we use today were either directly or indirectly inherited from this period.

Modern programming languages provide rich support for security and error checking. They provide a higher level of abstraction of hardware details

Classification of Programming Languages

Do you that thousands of programming languages have been written to date? Each was for some specific purpose. Some programming languages provide less or no abstraction from the hardware, whereas some provide higher abstraction. To separate programming languages based on the level of abstraction from hardware, they are classified into various categories.

Programming languages are classified into two main categories – Low-level language and High-level language. However, there also exists another category known as Middle-level language. Every programming language belongs to one of these categories and sub-categories.

People express themselves using a language that has many words. But Computers understand only a simple language that consists of 1s and 0s, with a 1 meaning "on" and a 0 meaning "off." This is because electrical devices seem to

fall naturally into one of two possible states. For example; they are either on or off; they are magnetized in one direction or the other; they are conducting electricity, or they are not conducting electricity. Hence, the instructions coded into a computer's memory are configured as 0‟s and 1‟s (binary notation).

As I have earlier explained the computer operates using a program coded in 0‟s and 1‟s, it would be very time consuming for a you to write a program in 0‟s and 1‟s. If the program can be written in a language approaching that of English or mathematics or a combination of both, the programmer can concentrate more on programming logic and less on programming details. Thus, the program should be less likely to contain errors. Our effort to overcome this difficulty has led to the development of new programming languages that can be well suited to the capabilities of both. Over the years, computer languages have evolved from machine language (computer's native language) to high-level languages. Also, computer programs written in the earliest programming languages were based on the underlying internal structure of the computer.

High-level languages facilitate programmers. Rather than learning the machine language, they can instruct the computer in a way that is easier to learn and understand. Languages that favour humans are termed as high-level and those oriented to machines are low-level. Presently, programming involves two levels following major levels of programming languages; Low-level Language and High-level language. binary machine language is known as the lowest level, whereas the highest level might be human language such as English.

Lower Level Languages (LLL)

Low-level language is a programming language helps us to communicate with computer's hardware components and constraints. It has no or a minute level of abstraction about a computer and works to manage a computer's operational semantics. Low-level languages are designed to operate and handle the entire hardware and instructions set architecture of a computer directly. Low-level languages can be converted to machine code without using a compiler or interpreter, and the resulting code runs directly on the processor.

You will observe that a program written in a low-level language can be made to run very quickly, and with a very small memory footprint; an equivalent program in a high-level language will be more heavyweight. Low-level

languages are simple but are considered difficult to use due to the numerous technical details which must be remembered. By comparison, a high-level programming language isolates the execution semantics of computer architecture from the specification of the program which simplifies development. We can divide low level language into two categories: Machine Language and Assembly language

- Machine Language: Machine language is the sequence of bits (machine code) that directly controls a processor, causing it to add, compare, move data from one place to another, and so forth at appropriate times. The computer microprocessor can process the machine codes directly without a previous transformation. Specifying programs at this level of detail is an enormously tedious task. Currently, programmers rarely write programs directly in machine code because it requires attention to numerous details which a high-level language would handle automatically, and also requires memorizing or looking up numerical codes for every instruction that is used.

- Assembly language: Assembly language uses structured commands (mnemonics) as substitutions for numbers allowing humans to read the code easier than looking at binary. Although easier to read than binary, assembly language is a difficult language and is usually substituted for a higher language such as C. The problem with assembly language is that it requires a high level of technical knowledge, and it is slow to write. Typically, one machine instruction is represented as one line of assembly code. Assemblers produce object files that may be linked with other object files or loaded on their own. Most assemblers provide macros.

Characteristics Lower Level Languages

1) Low-level languages, abbreviated as **LLL**, are languages close to the machine level instruction set. They provide less or no abstraction from the hardware.

2) A low-level programming language interacts directly with the registers and memory since instructions written in low-level languages are machine-dependent.

3) Programs developed using low-level languages are machine-dependent and are not portable.

4) Low-level language does not require any compiler or interpreter to translate the source to machine code. An assembler may translate the source code written in low-level language to machine code.

5) Programs written in low-level languages are fast and memory efficient. However, it is a nightmare for programmers to write, debug, and maintain low-level programs. They are mostly used to develop operating systems, device drivers, databases, and applications that require direct hardware access.

High-Level Languages (HLL)

2 mins

Have you noticed, the High-Level languages have replaced machine and assembly language in all areas of programming? Programming languages were designed to be high-level if it is independent of the underlying machine. High-level languages (also known as problem-oriented languages) enable us as a programmer to write programs that are more or less independent of a particular type of computer. We call this type of language high-level because they are closer to human languages and farther from machine languages. High-level languages are portable (machine independent) as it can be run on different machines with little or no change.

SAQ 3

Furthermore, the rules for programming in a particular high-level language are much the same for all computers, so that a program written for one computer can generally be run on many different computers with little or no alteration. Thus, we see that a high-level language offers three significant advantages over machine language: simplicity, uniformity, and portability. Higher-level languages provide a richer set of instructions and support, making the programmer's life even easier.

The languages such as BASIC, COBOL, FORTRAN, C, C++, JAVA, and Visual Basic are popular examples of high-level languages. High-level languages use translator programs such as compiler and interpreter to convert it into a

machine language program. Computer languages were first composed of a series of steps to wire a particular program; these morphed into a series of steps keyed into the computer and then executed; later these languages acquired advanced features such as logical branching and object orientation. Many programming languages require computation to be specified in an imperative form (i.e., as a sequence of operations to perform), while other languages utilize other forms of program specification such as the declarative form (i.e., the desired result is specified, not how to achieve it). Based on these programming paradigms, computer languages are classified into four main categories as follows.

- **Procedural languages:** These languages use a programming approach where a developer writes code that describes in exacting detail the steps that the computer must take to accomplish the goal.

- **Object-oriented languages:** These languages use a programming paradigm that represents concepts as "objects" that have data fields and associated procedures known as methods. Objects, which are instances of classes, are used to interact with one another to design applications and computer programs.

- **Logical Languages:** The programming paradigm in these languages is based on formal logic. A program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain.

- **Functional languages:** This paradigm was explicitly created to support a purely functional approach to problem-solving. The programming approach in these languages involves composing the problem as a set of functions to be executed. These functions are predefined blocks of code intended to behave like mathematical functions.

Advantages High-Level Languages (HLL)

- High-level languages are programmer-friendly. They are easy to write, debug, and maintain.
- It provides a higher level of abstraction from machine languages.
- It is a machine-independent language.
- Easy to learn.
- High-level programming results in better programming productivity.

Disadvantages High-Level Languages (HLL)

- It takes additional translation times to translate the source to machine code.
- High-level programs are comparatively slower than low-level programs.
- Compared to low-level programs, it is generally less memory efficient.
- Cannot communicate directly with the hardware.

Difference between Low- and High-Level Languages

- **Program speed**

  Programs in low-level language are written either in binary or assembly language. They do not require any compilation or interpretation. They interact directly with the registers and memory. Thus, they are comparatively faster than high-level languages.

  The high-level language uses English statements to write programs. Hence, they require compilers or interpreters to translate the source to machine language. They do not interact directly with the hardware. Thus, they are slower than low-level languages.

- **Memory efficiency**

  Low-level languages are memory efficient. They generally consume less memory.

  High-level languages are not memory efficient. They generally run inside a specific runtime environment. Also, several other programs are running concurrently to increase the optimal efficiency of the program which consumes memory. Thus, the overall memory consumption of high-level language is comparatively more than a low-level language.

- **Easiness**

  Low-level languages are machine, friendly languages. To write a program in a low-level language, we must know binaries or mnemonics of low-level instruction sets. Remembering various instructions sets for different architectures is nearly impossible. Thus, low-level programming is difficult to learn. Learning low-level languages requires additional knowledge and experience about the specific Machine architecture.

  High-level languages are programmer's friendly language. Programs in high-level languages are written using English statements which are much easier to remember than low-level binaries or mnemonics. Hence, high-level programming is easy to learn.

- **Portability**

  The low-level language contains low-level computer instructions set. These instructions are machine-dependent and are different for different architectures. Hence, programs developed are also machine-dependent and are not portable. High-level languages use English statements to write programs. They are further translated to machine language using a compiler or interpreter. There exists a separate compiler or interpreter for different machine architectures. That translates the source to specific machine language. Hence, high-level languages are machine independent and are portable.

- Abstraction level

  The low-level language provides less or no abstraction from the hardware.

They are the closest language to the hardware. They interact directly with the computers register and memory.

The high-level language provides a high-level of abstraction from the hardware. They run on top of the machine language. They do not interact directly with the computers register and memory. There is a layer of an operating system and other software through which they interact with the hardware.

- **Debugging and maintenance**

Low-level languages are more error-prone. Such errors could be small syntactical or big memory leaks Error detection and maintenance is a tedious and time taking process.

High-level languages are less error-prone. Almost all syntactical errors are identified using compilers or interpreters. They are generally easy to debug and maintain.

- **Additional knowledge and experience**

Low-level languages are machine-dependent. They require prior knowledge of the particular computer architecture before one can write a program for that computer.

High-level languages are machine-independent. They do not require any prior knowledge of computer architecture.

- **Applications**

Low-level languages interact directly with the hardware. They provide very less or no abstraction from the hardware. But they are blazing fast when compared to high-level languages. Thus, they are generally used to develop operating systems and embedded systems.

High-level languages provide a higher level of abstraction from the hardware. Nowadays, almost all types of software are developed using a high-level language. It is used to develop a variety of applications such as desktop applications, websites, utility software, mobile applications, etc.

Summing up the differences between low-level and high-level programming languages

Low-level language	High-level language
They are faster than high-level language.	They are comparatively lower.
Low-level languages are memory efficient.	High-level languages are not memory efficient.
Low-level languages are difficult to learn.	High-level languages are easy to learn.
Programming at a low-level requires additional knowledge of computer architecture.	Programming at a high-level does not require any additional knowledge of computer architecture.
They are machine-dependent and are not portable.	They are machine independent and portable.
They provide less or no abstraction from the hardware.	They provide high abstraction from the hardware.
They are more error-prone.	They are less error-prone.
Debugging and maintenance are difficult.	Debugging and maintenance are comparatively easier.
They are generally used for developing system software (Operating systems) and embedded applications.	They are used to develop a variety of applications such as –desktop applications, websites, mobile software, etc.

Activity 1

Computer Programming Language

Questions

- Briefly discuss the history of programming language
- Write short notes on the classifications of programming language

Self Assessment Questions

1. Explain a computer program?
2. What is a computer programming language?
3. State six (6) examples of high-level programming language

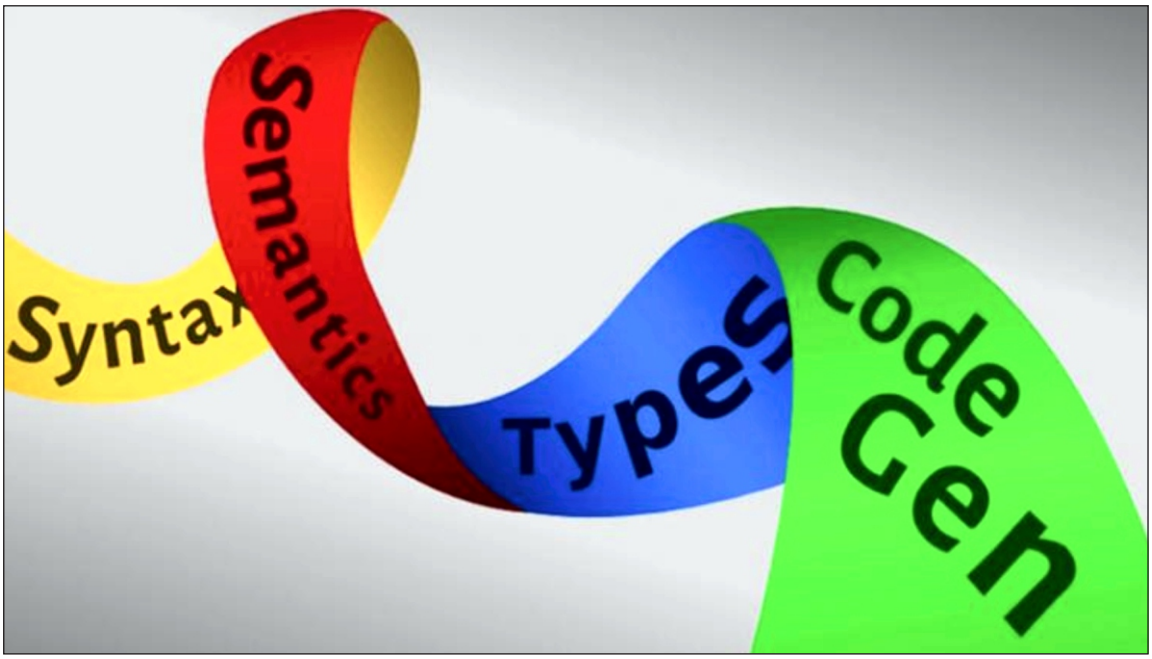
Tutor Marked Assessment

Use an algorithm to solve the following problems

- Discuss Five (5) differences between low-level and high-level programming language
- Mention four (4) advantages and three (3) disadvantages of a high-level language
- State and explain five (5) characteristics of low-level language

Further Reading

Michael Trombetta QBASIC for Students / Edition 1, McGraw-Hill Higher Education, 1993



UNIT 2

Compiler and Interpreter

Introduction

We generally write a computer program using a high-level language. A high-level language is one that is understandable by humans. It contains words and phrases from the English (or other) language. But a computer does not understand high-level language. It only understands program written in 0's and 1's in binary, called the machine code. A program written in a high-level language is called source code. We need to convert the source code into machine code, and compilers and interpreters accomplish this. Hence, a compiler or an interpreter is a program that converts a program written in a high-level language into machine code understood by the computer.

Learning Outcomes

When you have studied this unit, you should be able to:

1. Explain complier
2. Explain interpreter
3. State five (5) difference between compiler and interpreter

Main Content

Compiler

SAQ 1

7 mins

Write on your computer a program using your favorite programming language and save it in a text file called the program file.

Now, let us try to get a little more detail on how the computer understands a program written by you using a programming language. The computer cannot understand your program directly given in the text format, so we need to convert this program in a binary format, which can be understood by the computer.

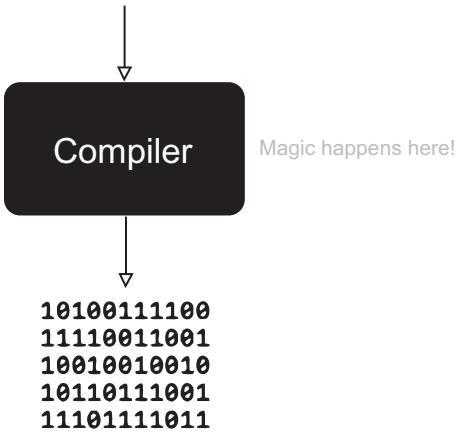
The conversion from text program to binary file is done by another software called Compiler, and this process of conversion from formatted text program to binary format file is called program compilation. Finally, you can execute a binary file to perform the programmed task.

We are not going into the details of a compiler and the different phases of compilation. Compilers are required in case you are going to write your program in a programming language that needs to be compiled into binary format before its execution.

The simplest definition of a compiler is a program that translates code written in a high-level programming language (like JavaScript or Java) into low-level code (like Assembly) directly executable by the computer or another program such as a virtual machine.

For example, the Java compiler converts Java code to Java Bytecode executable by the JVM (Java Virtual Machine). Other examples are V8, the JavaScript engine from Google which converts JavaScript code to machine code or GCC which can convert code written in programming languages like C, C++, Objective-C, Go among others to native machine code.

```
func  greet ( )  =  {
      Console . println ( “Hello, World! ” )
}
```



What is in the black box?

So far, we have looked at a compiler as a magic black box that contains some spell to convert high-level code to low-level code. Let us open that box and see what is inside.

A compiler can be divided into 2 parts.

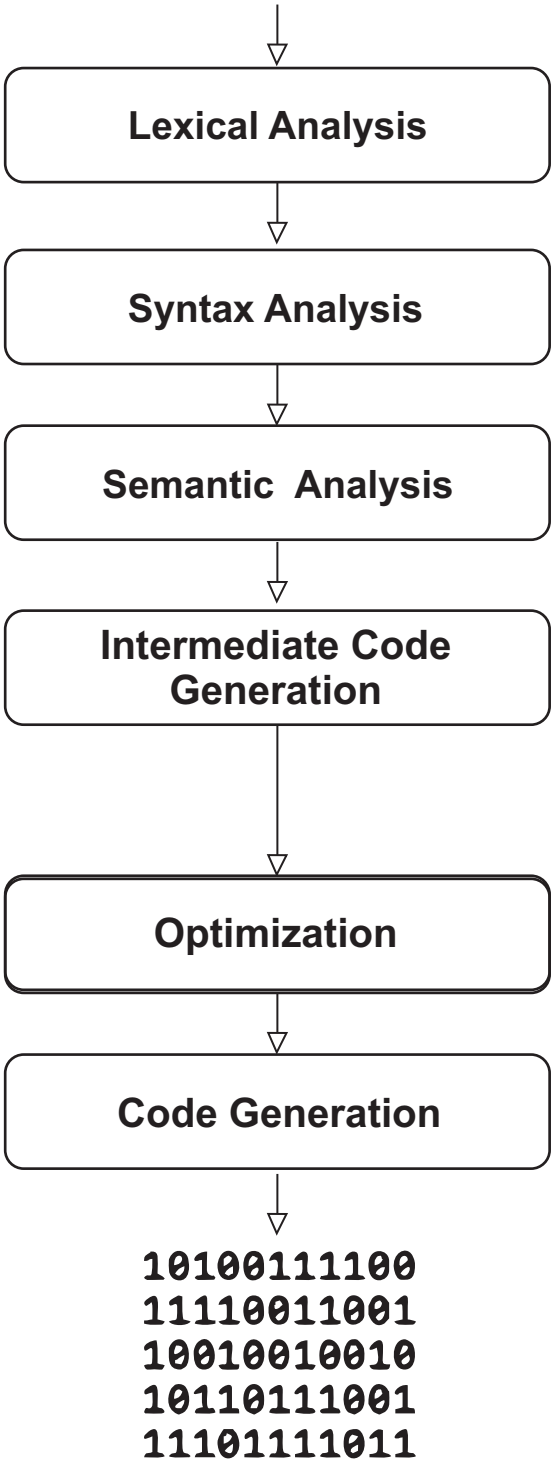
The first one generally called **the front end** scans the submitted source code for syntax errors, checks (and infers if necessary) the type of each declared variable and ensures that each variable is declared before use. If there is an error, it provides informative error messages to the user. It also maintains a data structure called **symbol table** which contains information about all the *symbols* found in the source code. Finally, if no error is detected, another data structure, an *intermediate representation* of the code, is built from the source code and passed as input to the second part.

In the second part, the **back end** uses the *intermediate representation* and the *symbol table* built by the *front end* to generate low-level code.

Both the front end and the back end perform their operations in a sequence of phases. Each phase generates a particular data structure from another data structure emitted by the phase before it.

The phases of the front end generally include **lexical analysis**, **syntax analysis**, **semantic analysis**, and **intermediate code generation** while the back end includes **optimization** and **code generation**.

```
func  greet ( )  =  {
        Console . println ( “Hello, World! ” )
}
```

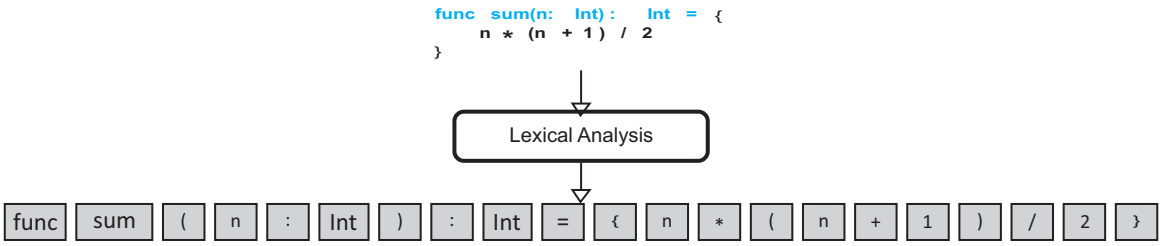


Structure of a compiler

Lexical Analysis

The first phase of the compiler is the *lexical analysis*. In this phase, the compiler breaks the submitted source code into meaningful elements called **lexemes** and generates a sequence of **tokens** from the lexemes. A *lexeme* can be thought of as a uniquely identifiable string of characters in the source programming language, for example, *keywords* such as if, while func, *identifiers*, *strings*, *numbers*, *operators* or *single characters* like (,), or:

A *token* is an object describing a *lexeme*. Along with the value of the *lexeme* (the actual string of characters of the lexeme), it contains information such as its type (*is it a keyword? an identifier? an operator?* ...) and the position (line and/or column number) in the source code where it appears.

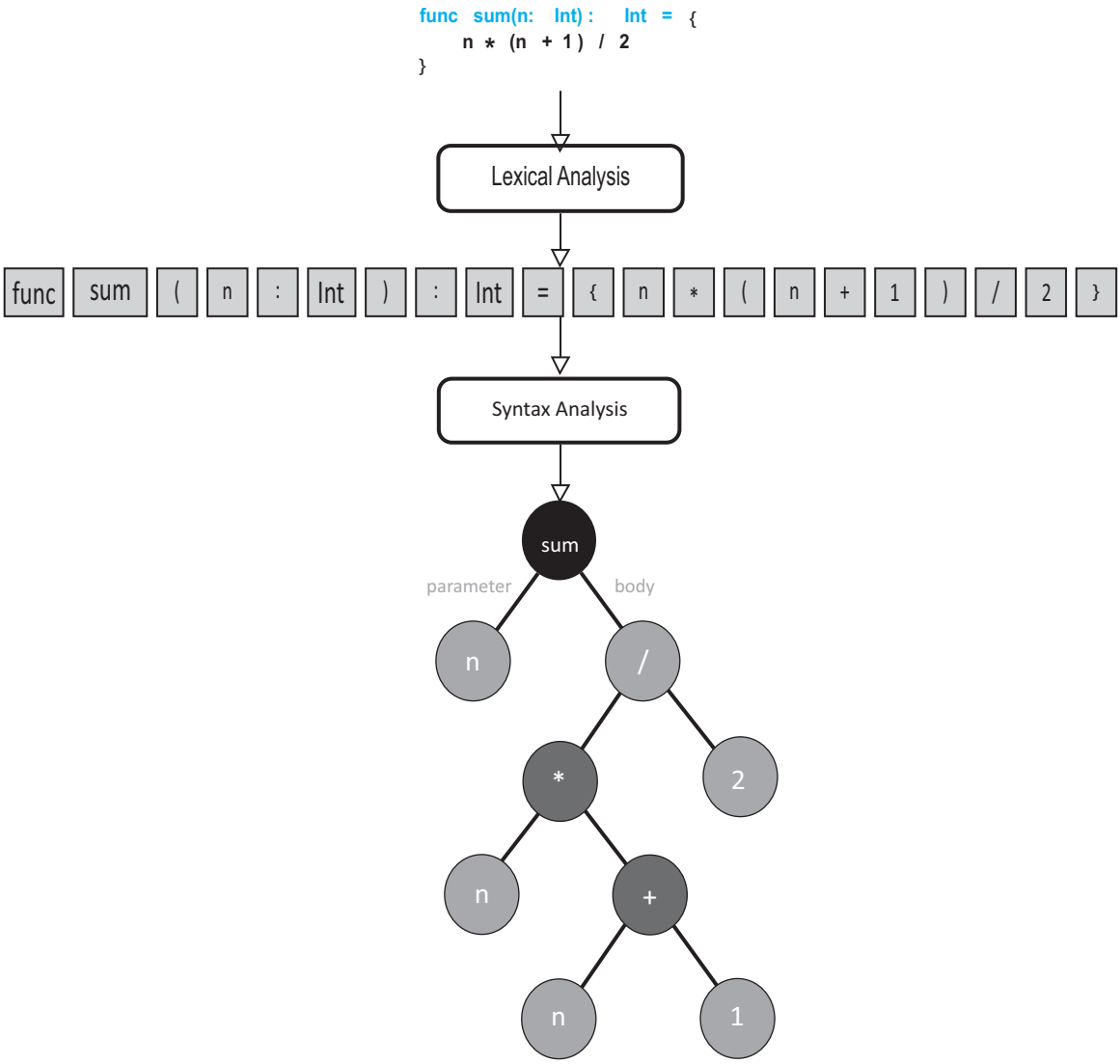


The sequence of lexemes generated during lexical analysis

If the compiler encounters a string of characters for which it cannot create a *token*, it will stop its execution by throwing an error; for example, if it encounters a malformed string or number or an invalid character (such as a non-ASCII character in Java).

Syntax Analysis

During syntax analysis, the compiler uses the sequence of *tokens* generated during the lexical analysis to generate a tree-like data structure called **Abstract Syntax Tree**, **AST** for short. The *AST* reflects the syntactic and logical structure of the program.



Abstract Syntax Tree generated after syntax analysis.

Syntax analysis is also the phase where eventual syntax errors are detected and reported to the user in the form of informative messages. For instance, in the example above, if we forget the closing brace} after the definition of the sum function, the compiler should return an error stating that there is a missing}, and the error should point to the line and column where the} is missing. If no error is found during this phase, the compiler moves to the *semantic analysis* phase.

Semantic Analysis

During semantic analysis, the compiler uses the *AST* generated during syntax analysis

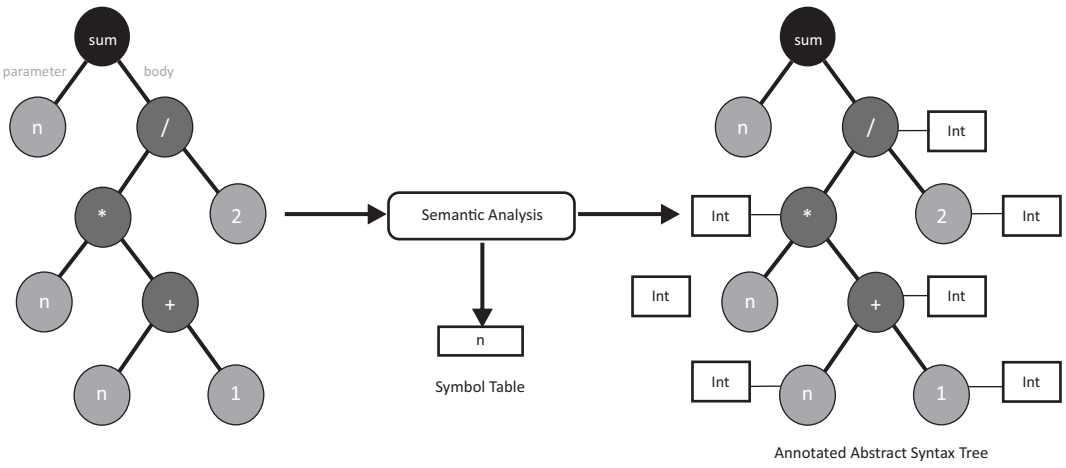
to check if the program is consistent with all the rules of the source programming language. The semantic analysis encompasses the following:

Type inference. If the programming language supports type inference, the compiler will try to infer the type of all untyped expressions in the program. If a type is successfully inferred, the compiler will **annotate** the corresponding node in the *AST* with the inferred type information.

Type checking. Here, the compiler checks that all values being assigned to variables and all arguments involved in the operation have the correct type. For example, the compiler makes sure that no variable of type String is being assigned a Double value or that a value of type Bool is not passed to a function accepting a parameter of type Double or again that we are not trying to divide a String by an Int, "Hello" / 2 (unless the language definition allows it).

Symbol management. Along with performing type inference and type checking, the compiler maintains a data structure called **symbol table** which contains information about all the symbols (or names) encountered in the program. The compiler uses the *symbol table* to answer questions such as *Is this variable declared before use? Are there 2 variables with the same name in the same scope? What is the type of this variable? Is this variable available in the current scope?* And many more.

The output of the semantic analysis phase is an **annotated AST** and the **symbol table**.

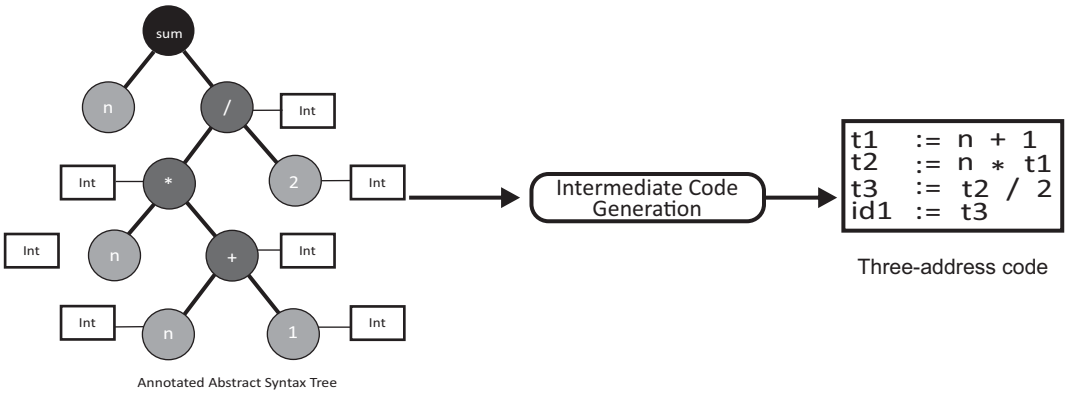


Intermediate Code Generation

After the semantic analysis phase, the compiler uses the *annotated AST* to generate an intermediate and machine-independent low-level code. One such intermediate representation is the three-address code.

The *three-address code* (*3AC*), in its simplest form, is a language in which an instruction is an assignment and has at most 3 operands.

Most instructions in *3AC* are of the form a: **=** b **<operator>** c or a: **=** b.



The above drawing depicts a *3AC* code generated from an *annotated AST* created during the compilation of the function

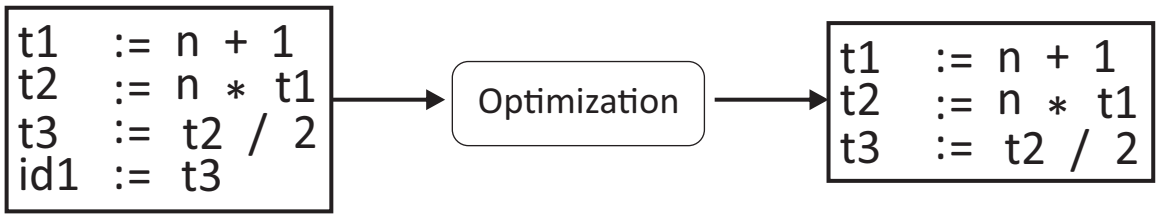
```
func sum (n: Int): Int = {
  n * (n + 1) / 2
}
```

The intermediate code generation concludes the *front-end* phase of the compiler.

Optimization

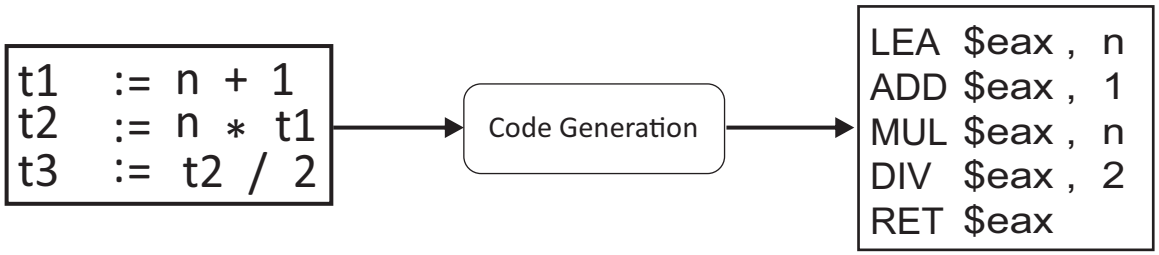
In the optimization phase, we use the first phase of the *back end*, the compiler uses different optimization techniques to improve on the intermediate code generated by making the code faster or shorter, for example.

For example, a very simple optimization on the *3AC* code in the previous example would be to eliminate the temporary assignment t3: = t2 / 2 and directly assign to id1 the value t2 / 2.



Code Generation

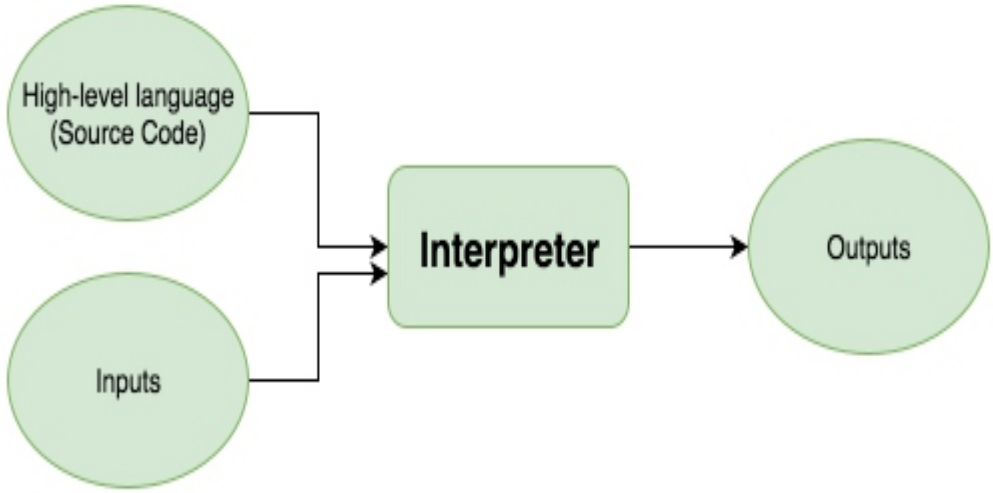
In this last phase, the compiler translates the optimized intermediate code into a machine-dependent code, *Assembly*, or any other target low-level language.



Computer vs Interpreter

You will notice interpreters and compilers are very similar in structure. The main difference is that an interpreter directly executes the instructions in the source programming language while a compiler translates those instructions into efficient machine code.

An interpreter will typically generate an efficient intermediate representation and immediately evaluate it. Depending on the interpreter, the intermediate representation can be an *AST*, an *annotated AST* or a machine-independent low-level representation such as the *three-address code*.



Interpreter

3 mins

An interpreter is a computer program that is used to execute program instructions written using one of the many high-level programming languages directly. The interpreter transforms the high-level program into an intermediate language that it then executes, or it could parse the high-level source code and then performs the commands directly, which is done line by line or statement by statement.

SAQ 2

We implement Programming languages are in two ways: interpretation and compilation. As the name suggests, an interpreter transforms or interprets a high-level programming code into code that can be understood by the machine (machine code) or into an intermediate language that can be easily executed as well. The interpreter reads each statement of code and then converts or executes it directly. In contrast, an assembler or a compiler converts high-level source code into native (compiled) code that can be executed directly by the operating system.

In most cases, a compiler is more favourable since its output runs much faster compared to a line-by-line interpretation. However, since interpretation happens per line or statement, it can be stopped in the middle of execution to allow for either code modification or debugging. Both have their advantages and disadvantages and are not mutually exclusive; this means that they can be used in conjunction as most integrated development environments employ both compilation and translation for some high-level languages.

Since an interpreter reads and then executes code in a single process, it is very

useful for scripting and other small programs. As such, it is commonly installed on Web servers, which run a lot of executable scripts.

Do you know that interpreters were used as early as 1952 to ease programming within the limitations of computers at the time (e.g. a shortage of program storage space, or no native support for floating-point numbers)? We use Interpreters to translate between low-level machine languages, allowing code to be written for machines that were still under construction and tested on computers that already existed. The first interpreted high-level language was Lisp. Lisp was first implemented in 1958 by Steve Russell on an IBM 704 computer. Russell had read John McCarthy's paper and realized (to McCarthy's surprise) that the Lisp eval function could be implemented in machine code. The result was a working Lisp interpreter which could be used to run Lisp programs, or more properly, "evaluate Lisp expressions".

We have other programming languages such as Python, PHP, and Perl which do not need any compilation into binary format; rather we use an interpreter to read such programs line by line and execute them directly without any further conversion.

An interpreter is simpler than a compiler. It includes the parser, but instead of the code generator, the interpreter goes through the internal representation of the source code (such as an abstract syntax tree) and 'executes' the code directly. Although in principle we can compile or interprets any language, languages that are usually compiled tend to be dynamically typed and scoped, while compiled languages are statically typed and lexically scoped.

Difference between interpreter and compiler

We use interpreter to translates high-level instructions into an intermediate form, which can be executed. In contrast, a compiler translates high-level instructions directly into machine language. Compiled programs generally run faster than interpreted programs. The advantage of an interpreter, however, is that it does not need to go through the compilation stage during which machine instructions are generated. This process can be time-consuming if the program is long. The interpreter, on the other hand, can immediately execute high-level programs.

For this reason, interpreters are sometimes used during the development of a

program, when a programmer wants to add small sections at a time and test them quickly. Also, interpreters are often used in education because they allow students to program interactively.

Both interpreters and compilers are available for most high-level languages. However, BASIC and LISP are specially designed to be executed by an interpreter. Besides, page description languages, such as PostScript, use an interpreter. Every PostScript printer, for example, has a built-in interpreter that executes PostScript instructions

The differences between an interpreter and a compiler are given below:

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes a large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby uses interpreters.	Programming languages like C, C++ use compilers.

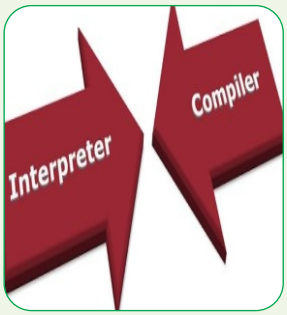
Summary

In this unit, you have learned that:

- A compiler or an interpreter is a program that converts program written in a high-level language into machine code
- Compiler convert from text program to binary file
- The compiler can be divided into two parts
- An interpreter is a computer program that is used to execute program

  Instructions written using one of the many high-level programming languages directly

Activity 1



Questions

- Define Compiler
- Give three examples of Interpreter
- Mention four differences between compiler and interpreter

Compiler and Interpreter

Self Assessment Questions

1. Define is a compiler?
2. Explain interpreter?
3. State four (4) differences between compiler and interpreter?

Tutor Marked Assessment

Use an algorithm to solve the following problems

- Itemise and explain the process of a compiler
- What is an interpreter with given examples?
- Demise and discuss five (5) differences between compiler and interpreter

Further Reading

Torben Mogensen, Introduction to Compiler Design, Springer;

2011 edition (Aug. 2, 2011), 319 pages

William M. Waite and Gerhard Goo, Compiler Construction, Springer;

Softcover reprint of the original 1st ed. 1984 edition (May 24, 2013),

447 pages

Operating System Concepts, by Abraham Silberschatz,

ISBN13: 978-1118063330 9th Edition, 2013.

References

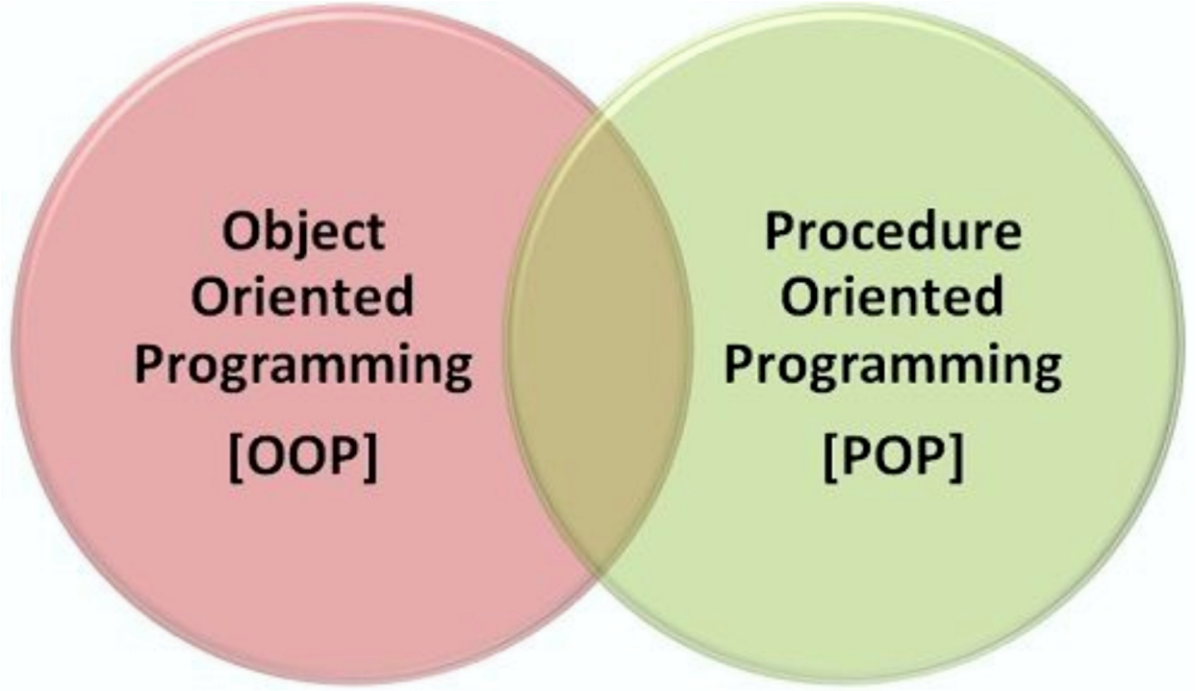
Chen, D., J. R. Youssef, and G. Zacharewicz. (2015). 'Towards an Enterprise Operating System – Requirements for Standardisation, Proceedings of IWEI 2015 (IWEI Workshops 2015), May 2015.

Leadbeater, P. F., 1999. “Systems Architecture.” Enterprise Model Execution and Integration, CEN/TC310, ENV13550 40 (2–3): 305–310

Aho, Alfred V., Hopcroft, J. E., and Ullman, Jeffrey D. (1974). The Design and Analysis of Computer Algorithms.

Addison Wesley, Reading, MA. Aho, Alfred V., and Johnson, Stephen C. (1976). Optimal code generation for expression trees. Journal of the ACM, 23(3):488{501.

Aho, Alfred V., Johnson, Stephen C., and Ullman, Jeffrey D. (1977). Code generation for machines with multi register operations. Journal of the ACM, pages 21{28.



UNIT 3

Procedural Programming and Object-oriented Programming

Introduction

In this unit, I will introduce you to procedural and objects-oriented programming principles, how it can be used in a programming language to solve a real-world problem, and given examples. The differences between both procedural and object-oriented programming languages were taught in this unit.

Learning Outcomes

At the end of this lesson, you should be able to:

1. Explain procedural programming language,
2. Explain object-oriented programming language,
3. State at least five (5) differences between procedural and object-oriented programming language

Main Content

SAQ 1

Procedural Programming

2 mins

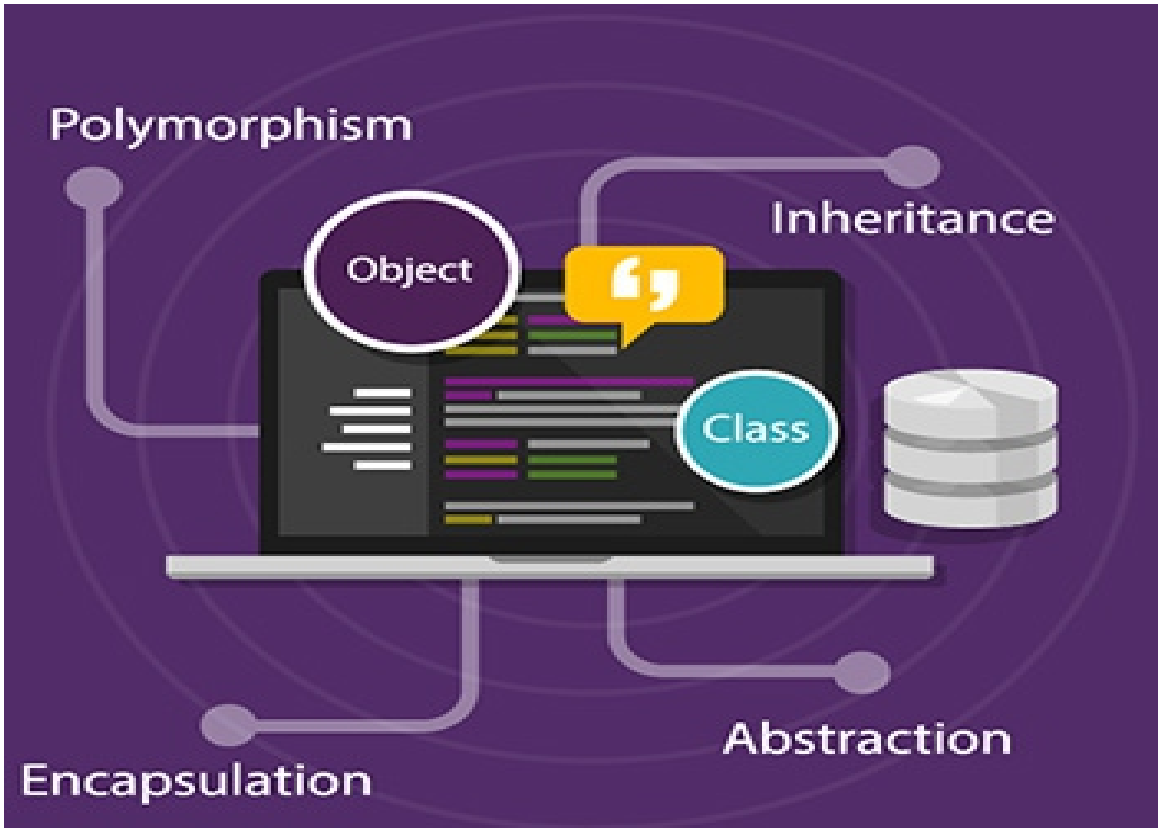
Programs are made up of modules which are parts of a program that can be coded and tested separately and then assembled to form a complete program. In procedural languages (i.e. C), these modules are procedures, where a procedure is a sequence of statements. In C, for example, procedures are a sequence of imperative statements, such as assignments, tests, loops, and invocations of sub-procedures. These procedures are functions that map arguments to return statements.

The design method used in procedural programming is called Top DownDesign. This is where you start with a problem (procedure) and then systematically break the problem down into subproblems (subprocedures). This is called functional decomposition, which continues until a subproblem is straightforward enough to be solved by the corresponding subprocedure. The difficulty with this type of programming is that software maintenance can be difficult and time-consuming. When changes are made to the main procedure (top), those changes can cascade to the subprocedures of main, and the sub-sub procedures and so on, where the change may impact all procedures in the pyramid.

One alternative to procedural programming is object-oriented programming. Object-oriented programming is meant to address the difficulties with procedural programming. In object-oriented programming, the main modules in a program are classes, rather than procedures. The object-oriented approach lets you create classes and objects that model real-world objects.

A procedural language is a type of computer programming language that specifies a series of well-structured steps and procedures within its programming context to compose a program. It contains a systematic order of statements, functions, and commands to complete a computational task or program.

Procedural language segregates a program within variables, functions, statements, and conditional operators. Procedures or functions are implemented on the data and variables to perform a task. These procedures can be called/invoked anywhere between the program hierarchy and by other procedures as well. A program written in procedural language contains one or more procedures.



Object-Oriented Programming

SAQ 2

2 mins

Object-oriented (OO) programming is a programming paradigm that uses abstraction (in the form of classes and objects) to create models based on the real-world environment. An object-oriented application uses a collection of objects, which communicate by passing messages to request services. Objects are capable of passing messages, receiving messages, and processing data. Object-oriented programming aims to try to increase the flexibility and maintainability of programs. Because programs created using an OO language are modular, they can be easier to develop, and simpler to understand after development.

Object-oriented programming is centred on creating objects rather than procedures/ functions. Objects are made up of procedures that manipulate that data. Data in an object are known as attributes. Procedures/functions in an object are known as methods.

Object-oriented programming combines data and behaviour (or method). This is called encapsulation.

Data hiding is the ability of an object to hide data from other objects in the

program. Only an object's methods should be able to manipulate its attributes directly. Other objects are allowed to manipulate an object's attributes via the object's methods. This indirect access is known as a programming interface.

OOP features include the following:

Encapsulation: This makes the program structure easier to manage because each object's implementation and state are hidden behind well-defined boundaries.

Polymorphism: This means abstract entities are implemented in multiple ways.

Inheritance: This refers to the hierarchical arrangement of implementation fragments.

Object-oriented programming allows for simplified programming. Its benefits include reusability, refactoring, extensibility, and efficiency.

This benefit also implies it:

Save development time (and cost) by reusing code –once an object class is created it can be used in other applications

Easier debugging –classes can be tested independently

Differences between procedure and object-oriented programming

3 mins

Difference between Procedure Oriented Programming (POP) & Object-Oriented Programming (OOP)

	Procedure Oriented Programming	Object- Oriented Programming
Divided Into	In POP, the program is divided into small parts called functions.	In OOP, the program is divided into parts called objects.
Importance	In POP, Importance is not given to data but to functions as well as a sequence of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a real world.
Approach	POP follows a Top- Down approach.	OOP follows a Bottom- Up approach.
Access Specifiers	POP does not have any access specifier.	OOP has access to specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and functions.
Data Access	In POP, the most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data cannot move easily from function to function; it can be kept public or private so that we can control the access of data.
Data Hiding	POP does not have any proper way of hiding data, so it is less secure.	OOP provides Data Hiding so provides more security.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.

SAQ 3

Summary

In this unit, you have learned that:

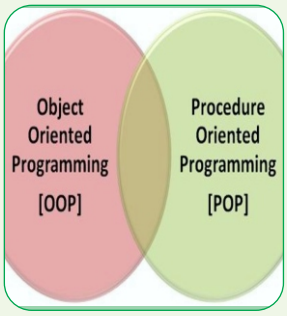
- Programs are made up of modules which are parts of a program that can be coded and tested separately
- The object-oriented application uses a collection of objects
- A procedural language is a type of computer programming language that specifies a series of well-structured steps and procedures

Activity 1

Compiler and Interpreter

Questions

- Define Procedural Programming Language
- Mention five differences between POP and OOP



Self Assessment Questions

1. Define procedural programming language?
2. What is an object-oriented language?
3. Itemize five differences between POP and OOP

Tutor Marked Assessment

Use an algorithm to solve the following problems

- Discuss five (5) differences between procedural and object-oriented languages
- Discuss four (4) characteristics of procedural languages

Further Reading

Michael Trombetta QBASIC for Students / Edition 1, McGraw-Hill Higher Education, 1993

Wallace Wang Beginning Programming for Dummies by Wallace Wang, 2001

References

- Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dolling, Helena Gilchrist, Fiona Hayes, and Paul Jeremaes, Object-Oriented Development: The Fusion Method, Prentice-Hall, 1994, ISBN 0-13-338823-9.

- Frank Buschmann, Regine Meunier, Hans Johnert, Peter Sommerlad, and Michael Stal, Pattern-Oriented Software Architecture: A System of Patterns, Wiley, 1996, ISBN 0-471-95869-7.

- Adele Goldberg and Kenneth S. Rubin, Succeeding with Objects: Decision Frameworks for Project Management, Addison-Wesley, 1995, ISBN 0-201-62878-3.

QBasic

Module 5:

Introduction to Basic Language (QBASIC)

Unit 1: BASIC Programming Language
Unit 2: Control Structures
Unit 3: System Built-in Functions and String Manipulation

```
10 INPUT "What is your name: ", U$
20 PRINT "Hello "; U$
30 INPUT "How many stars do you want: ", N
40 S$ = ""
50 FOR I = 1 TO N
60 S$ = S$ + "*"
70 NEXT I
80 PRINT S$
90 INPUT "Do you want more stars? ", A$
100 IF LEN(A$) = 0 THEN GOTO 90
110 A$ = LEFT$(A$, 1)
120 IF A$ = "Y" OR A$ = "y" THEN GOTO 30
130 PRINT "Goodbye "; U$
140 END
```

UNIT 1

BASIC Programming Language

Introduction

Dr John G. Kemeny and Thomas Kurtz developed the BASIC language at Dartmouth in 1964. BASIC stood for "Beginner's All-Purpose Symbolic Instruction Code". Their objective: to create a simplified computer language for teaching students how to program.

Learning Outcomes

At the end of this lesson, you should be able to:

1. Install QBASIC
2. List six (6) characteristics of QBASIC
3. State five (5) advantages and four (4) disadvantages of QBASIC
4. Explain BASIC character set
5. Explain QBASIC programming contents
6. Solve problems using a Basic program

Main Content

How to install QBASIC Programming Language

SAQ 1

2 mins

Before you can create a program in QBasic, you need the QBasic interpreter. It is available from your Windows 95 (or 98) CD, or you can download it below.

To access QBasic from the Windows 95 CD:

1. Insert the CD into your CD-ROM drive.

2. Click "browse this CD" (if the menu screen doesn't come up, then browse the CD from My Computer. 3. Go to the \OTHER\OLD MSDOS directory.

4. Open a program called QBASIC.EXE (this is version 1.1 of the QBasic interpreter).

To access QBasic from the Windows 98 CD:

1. Insert the CD into your CD-ROM drive.

2. Click "browse this CD" (if the menu screen doesn't come up, then browse the CD from My Computer. 3. Go to the \TOOLS\OLD MSDOS directory.

4. Open a program called QBASIC.EXE (this is version 1.1 of the QBasic interpreter).

Download it here (right-click and press "Save As"):

QBASIC.ZIP (323 KB) - QBasic 1.1 interpreter and sample programs

UNZIP32.EXE (90 KB) - Extracts the ZIP file

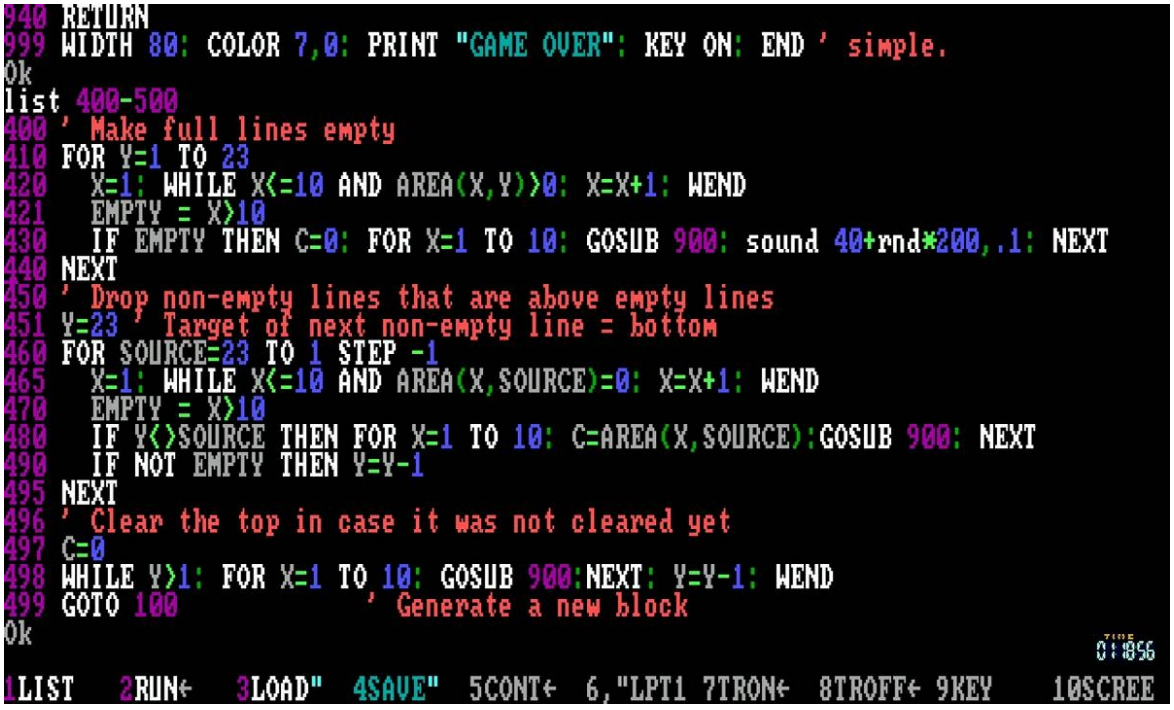
To unzip the QBASIC.ZIP file with UNZIP32.EXE:

a. Go to the Start Menu

b. Click Run...

c. Type the following (this loads MS-DOS): command

d. Enter the following in DOS (assuming you saved QBASIC.ZIP to C:\QBASIC):
cd c:\qbasic unzip32 -n qbasic.zip



Nature and Characteristics of BASIC

2 mins

Do you know that BASIC is an acronym for beginners' all-purpose symbolic instruction code? It is the name of a high-level programming language developed at Dartmouth College, New Hampshire USA under the auspices of professors John G Kemeny and Thomas E. Kutz during their academic years-1963-64. The language was developed to teach beginners the basic constructs of programming theory and as such, one of the easiest programming languages to learn.

SAQ 2

As a high-level programming language, the program code written in BASIC must be translated to its equivalence in machine code before it can be executed on the computer. A BASIC interpreter (i.e. translator software) is needed for this purpose. We call the program source code before the interpreter translates it while the translated version, ready for execution, is called the object code. However, the interpreter is not the only translator used in programming. Compilers are used by several other high-level programming languages such as C++, Pascal, etc. while assembly languages use assemblers. Java has both interpreter and compiler

An interpreter translates the sources code line by line and statement by statement. Complier does not generate intermediate object code from the source code but translates and executes the sources code simultaneously.

Advantages and Disadvantages of BASIC Programming Language

4 mins

Merits of BASIC Programming Language

1. BASIC is the simplest programming language.
2. BASIC is friendly, easy to teach and to learn compared to other HLL languages. It is flexible and easy to code or modifies existing code
3. It has simplified grammar (syntax) and relatively fewer numbers of statements
4. It has a multipurpose language suitable for interactive environments
5. Its interpreter is small and portable
6. It can be programmed on a terminal
7. It is very common and readily available on many micro and minicomputers
8. It is the most widely used time-sharing language in U.S.A and Canada
9. It comes automatically with every new microcomputer

SAQ 3

Demerits of BASIC Programming Language

1. It has a slow execution pace as translation is combined with execution.
2. Many versions of BASIC are unstructured because of statements such as GOTO.
3. It has limited portability as different versions require different interpreters for source code translation.
4. There are different versions of the BASIC Programming Language. More than 50 versions ("dialects") of BASIC exist today as a result of variations in hardware systems. All these variations are expected to conform to the universally accepted standard set out by the American National Standard Institute (ANSI). Some of these versions are ANSI BASIC, VISUAL BASIC, GW-BASIC, QUICK BASIC, TURBO BASIC, BASICA, MFBASIC, CASSETTE BASIC, APPLE BASIC, etc.

+	"	>	$
-	'	<	%
*	`	(	&
/	;	)	!
\	:	.	#
^	=	?	Blank Space

BASIC Character Set

13 mins

A character set refers to the composite number of different characters that are being used and supported by computer software and hardware. It consists of codes, bit patterns, or natural numbers used in defining some particular character.

SAQ 4

Alphabets Letters: A-Z; a-z

Digits Numbers: 0-9

Special characters:?, (,), +, =, *, &, ", %, $, n, !, >,

QBASIC Programming

Are you aware that of all of the high-level languages, BASIC (Beginners All-purpose Symbolic Instruction Code) is probably the easiest to learn? The beginning user soon discovers that small programs can be written and quickly entered into the computer, producing interesting results. The goal of the unit is to learn how to write computer programs in QBASIC Language.

QBASIC Language Contents

The following contents are used by QBASIC System:

1. Letters of the alphabet (A, B, C, .Z).

2. Digit Numbers (0, 1, 2……..9).

3. Characters set (+,-,*, /, =,^, ( ), , >=,=< ,<>,@,#,!,?,",%).

4. Special Words (go to, if, read, print, input).

Constants In QBASIC

Constants In QBASIC division into two types:

1. Numeric Constants: there are two types of numeric constants:

a. Real: the numbers used may be written in a decimal form such as (6.9,-52.76, 0.095, 3269.0)

b. Integer: Whole numbers may be written without the decimal point such as (89, - 132, 7698)

c. Exponential Form: this form requires a number followed by the letter E, such as (2.8E05, 0.57E-03, 0.07E-9, and 29.8E7).

b) String Constants. A string consists of a sequence of characters enclosed in double quote marks. Strings usually consist of names or addresses or cities such as "Computer", "Baghdad".

Variables in QBASIC

Again, Variables in QBASIC are classified into two types:

1. Numeric Variables: the variables used to represent numbers are the letters of the alphabet; these may be followed by a digit or letter. including 0 .for example: (A,b,c…..,A0,b1,c2,Ab,ba,bv,zx).

2. String Variables: a string variable must consist of a letter of the alphabet followed by the $ character, such as (A$, BC$, A2$, ZW$).

Arithmetic Expressions

We use an expression in QBASIC to perform calculations. In general, an expression may consist of a single constant or a single variable, or of arithmetic operation involving two or more constants or two or more variables, or of any arithmetic combination of constants and variables. Although complicated mathematical expressions will not occur frequently, some skill is required to translate mathematical expressions into QBASIC expressions. In performing this task, it is necessary to know that the QBASIC compiler uses the following order of precedence in performing arithmetic operations:

Order	Symbol	Remark
first	( )	Operations within parentheses are performed first.
second	^	Exponentiation is performed before other arithmetic operations
third	*, /	Multiplication and division are equal in the order of precedence.
forth	+ , -	Addition and subtraction are performed last and are equal in order of precedence.

Suppose (a=5. b=4, c=2, i=2, j=1) Execute the following expressions in order of precedence:

i. a+b+c/12

ii. a+b/c–I*j+a^2a*b–c/I+j+a^b+c^3j^b–I/c*j+20

iii. j/(I+c)*j/I+c

iv. (a*(b+I)*c)^2

v. (j+I*(b/c–a^2))+i

vi. (I+j)/a?I+j/a

vii. (j*(a^2–10))/2

viii. i+j^2–3*a

ix. ((I+j)/3*a)^2/2

Operators

We use an Operators also known as symbols to indicate the type of mathematical operations. QBASIC has to perform on the data or values of the variable. In QBASIC, there are four types/branches of operators used on the coding program, namely: Arithmetic, Relational, Logical, and String operators.

Arithmetic Operators

We use arithmetic operator to represent mathematical sign to perform simple and practical mathematical calculations like addition, subtraction, division, multiplication, exponents, etc.

Example 1:

Operation	Operators	Example	Results
Addition	+	5 + 8	13
Subtraction	-	8 – 6	2
Multiplication	*	5 * 4	20
Division	/	8/2	4
Integer division	\	9\2	4
Exponential	^	4^3	64
Modular division	Mod	7 Mod 3	1

Relational Operators:

They are operators used to perform a comparison on two values of the same typed statements. E.g.

Statement 1 (for P)	Statement 2 (for Q)	Results (P & Q)
F	T	F
T	F	F
F	F	F
T	T	T

Not Operator:

It operates on one statement operand and returns true if the logical statement returns false. E.g.

Statement 1 (for A)	Result (for Not A)
F	T
T	F

String Operator

A string joins two or more strings data (sentences) together to form a simple or compound sentence. E.g.

String data (for A$)	String data (for B$)	Results (A$ + B$)
"Wel"	"come"	Welcome
"50"	"45"	5045

The plus sign (+) is used as the string operator as the act of combining two strings which can also be known as concatenation.

Expression

It is the combination of operators, constants, and variables that are evaluated to get a result. The result is string data, numeric data or logical value and can be stored in a variable, e.g.

(A + B) > C

A >= B + C

u = t + ½ etc.

Note, an expression can come from any side of mathematics, some from the linear simultaneous equation, algebra, statistics, trigonometry, etc.

Example of algebra expression:

It must be converted into QBASIC expression and interpretation before the result is shown at the output.

A = L X B A = L * B

P = 2(L+B) P = 2 * (L + B)

I = (P *T * R)

100 I = (P * T * R)/100 V = 4 pi r3

V = 4/3 * (PI * R ^ 3)

V = r2h = PI * r ^ 2 * h

Order of Operation

The order of operations is the order (due process) in which all algebraic expressions should be simplified.

For instance, on QBASIC:

Parenthesis

Exponents (and Roots)

Multiplication and Division

Addition and Subtraction

[PEMDAS]

Alternatively:

Brackets

Exponentiation

Division and Multiplication

Addition and Subtraction

[BEDMAS]

Note:

You should be aware that multiplication does not always come before division, and addition does not always come before subtraction, for example.

3 X 3 – 5 + 2

Your work will always start from left to right.

You work like this. First, notice that there is no parenthesis or exponents, so we move to multiplication and division. There is only one multiplication, so we do that first and end up with 9 – 5 + 2

Now we move to addition and subtraction, so finally, we do the subtraction to get 4 + 2 and finally the addition to give 6. Thus, the order becomes: [PEDMSA]

Example 2: 75/5^(3-1) Using the PEDMSA

Solve for bracket (parenthesis) (3 – 1) = 2

= 75/5^2

Solve for exponentiation (5^2) = 25

=75/25

Finally, solve for division =75/25

= 3

Example 3:

In this example, chose the student that solves his/her expression correctly.

Student 1	student 2
3 + 4 x 2	3 + 4 x 2
= 7 x 2	= 3 + 8
= 14	= 11

Student 1 is wrong, and student 2 is right in the order of operation.

Library Functions

Each basic interpreter has within it the capability of calculating certain special functions, sometimes called library functions. These functions are indicated by three letters naming the function, followed by argument enclosed in parentheses. The argument may be a number, a variable, or an expression. In the following table, library functions as might be found in most basic interpreters.

Function	Description of function
ABS(x)	The absolute value of x, $\|x\|$.
SGN(x)	=(1 or 0 or 1) for (x<0 or x=0 or x>0).
SQR(x)	The square root of x.
INT(x)	The largest integer not exceeding x.
RND(x)	Create a random number value between 0 and 1.
Sin(x)	The sin of x, x in radians
Cos(x)	Cosine of x, x in radians
Tan(x)	Tangent of x, x in radians
Atn(x)	The arctangent of x, x in radians
Exp(x)	The value of ex
Log(x)	The natural logarithm of x ,ln(x)
Mod	Modulus-Rest of division. Ex:10mod3=1
Div	Integer division. Ex:10div3=3

In QBASIC, a variable is a place on the computer memory, which has a name and stores data temporarily. On the other hand, a variable is an entity that stores data needed to be used in a program. Each program defines a different number of variables. The value of a variable can be changed during the execution of the program. On declaring (i.e. typing) a variable, one will need to determine the type of data that QBASIC will store inside the variable.

There are mainly two types of variables declaration. They are: -

a) String variable

b) Numeric variable

String Variable:

String variable stores string data (i.e. quotation statements) with a string variable name declareD on that same line whereby mathematical and logical operations cannot be done on it as directed by the declaration symbols of a dollar sign ($).

Numeric Variable:

We use a numeric variable to store number and data inside variable names that are mathematical and logical operations and can easily be done on it as directed by its declaration symbols typed.

Example of numeric variable declaration:

We can express them under four headings, namely, Integer, long integer, single-precision, and double-precision variable.

For integers, it stores whole short number with the sign of percentage (%) typed to the variable name. Long integer stores a large range of whole numbers with ampersand (&) declaration, for single precision, it stores single decimal numbers with or without a declaration of the exclamation mark (!), and for double-precision, it stores large decimal numbers declared with the use of hash sign (#)

Rules for Typing a Variable to QBASIC

1. Variable names typing can have a maximum of 40 characters in length typed together without space.
2. Variable names can have alphabets, numbers, and alphanumeric on line numbers.
3. Variable name must begin with a letter
4. Variable names cannot begin with fn or Fn alphabets, for example, fnames$, fnum, etc. It is because fn is a keyword in BAISC and cannot be used as a variable name. Don't use it as a variable name.
5. Variable names cannot be served words.
6. Variable names may be ended with type declaration characters like $, %, &, ! and #

SAQ 4

The Qbasic Language statement: 5 mins

The REM Statement

The general form of the REMARK statement is REM [remark] where remark may be any remark the programmer chooses to make to describe the program, to identify various parts of a program, or even to serve as a blank line between parts of the program. For example REM this program is named calculate Rem program written in 2010-03-30 Rem input module Rem output module

Print Statement

The general form of the Print statement is Print expression (, or ;) expression, etc., where expression is any number, any string of words, any variable, or any combination of numbers and variables in an algebraic expression. Commas or semicolons must be used to separate expressions. For example:

Print

Print X

Print a,b,c

Print s;t;u;r;v

Print "the sum is";s9

Print a$, B$,c$

Remark: the symbol " " used in a print statement for print strings.

Input Instructions

There are three types of input instructions as follows:

1. Let statement.

2. Input statement.

3. Read / Data statement.

Let statement.

The general form of the Let statement is:

Let variable = expression

Where variable may be either a numeric variable or a string variable and expression may be either an arithmetic expression or a string expression. Strings cannot be placed into numeric variables or vice versa. The following are examples of let statement:

Let X=25

Let A$="computer"

Let R3=5*x*(X-9)

Let M$=N$

Example

Write a program to compute the following functions:

A=X2 + Y2

B=A2 + XY

C= (A+B)2

Solution

```
Let x=5
Let y=7
A=X^2+Y^2
B=A^2+X*Y
C=(A+B)^2
Print a,b,c
```

Input statement

We write the input statement in the form:

Input variable [, variable...]

Where variable stands for any suitable variable chosen by the programmer, also, it is the programmers' option to use more than one variable, separating them by commas, to enter additional values. When the program is running, and control comes to an input statement, the computer prints question mark (?) and pauses so that the user may enter the proper value for that variable. This opportunity to enter a value makes the program very flexible. The following are examples of Input statement:

Example-1

Write a program to compute the real roots of an equation

$AX^2+BX+C=0$

Solution

```
REM Program for computing roots of an equation
Input A,B,C
X1=(-b+ SQR(b^2-4*a*c)) /(2*a)
X2=(-b-SQR (b^2-4*a*c)) /(2*a)
Print" the first root is"; X1
Print " the second root is"; X2
```

Example-2 Write a program to compute the area and circumference of circle its radius R.

Solution

```
REM Program for compute circle area and circumference
Pi = 3.141592
Input "radius of circleis";R
A=R^2*Pi
C= 2*pi*r
Print" the area is";A
Print"the circumference is";C
```

Read / Data statement

The general form of the Read and Data statements are:

Read variable [,variable] etc.

Data datum [,datum] etc.

Where the variable is any valid numeric variable or string variable and datum is any valid number or string constant. The following are examples of Read / Data statements:

Read X, Y, Z

Data 12.7,35,-29.75

Read A,N$,M$,T

Data 7.4, "address","telephone",66.7

Example - 1

Write a program to read student names and 6 degrees then compute and print the average of student degrees.

Solution

```
Read A$,a,b,,c,d,e,f
S=a+b+c+d+e+f
AV=S/6
Print A$,AV
Data Ali,50,65,87,97,90,70
```

Example - 2

Write a program to read 4 variables then compute the arithmetic mean and geometric mean.

Solution

```
Read x1,x2,x3,x4
Ar= (x1+x2+x3+x4)/4
Ge=(x1*x2*x3*x4)^(1/4)
Print "arithmetic mean is";ar
Print "geometric mean is";ge
Data 50,67,80,77
```

The Restore statement

The general form of the restore statement is

Restore

Whenever control executes the restore statement, the system restores the data block pointer to the first item of data. Then the next read statement starts the process of reading the data all over again from the very first data statement.

Example
Read a,b,c
Restore
Read x,y,z
Data 5,7,9
The x,y,z have the values5,7,9

Control Statements

Control statements include
1. Go To statement
2. If-Then statement
3. If Go To statement
4. On Go To statement

The Go-To Statement
The general form of the Go-To statement is:
Go To Line number
Where line number represents the next line number to which control will go instead of the following line number. The following are examples of the go-to statement: Go to 10, Go to 5. This statement is sometimes called the unconditional go to statement.

Example-1
Write a program to compute the average of 6 subjects
for several students.
Solution
Now
10 Input N$,S1,S2,S3,S4,S5,S6
AV=(S1+S2+S3+S4+S5+S6)/6
Print N$.AV
Go To 10

2. The IF ... THEN Statement
The general form of the If ... then Statement is:
IF Condition THEN Line Number Where the condition has the form:
Arithmetic expression relation Arithmetic expression
Or
String expression relation String expression
And relation represents one of the symbols from the following table:

Symbol	Example
=	A=B
<	A<B
>	A>B
<=	A<=B
>=	A>=B
<>	A<>B

Example-1
Write a program to compute the Summation (S) of values from 1 to 100 where: S = 1+2+3+.....100.
Solution

```
S=0 : I=1
5 S=S+I
I=I+1
If I <= 100
then 5
Print S
```

Example - 2
Write a program to compute the summation for even numbers from 0 to N.
Solution

```
Input N
S=0 : I=0
5 S=S+I
I=I+2
If I < N then 5
Print S
```

3. The IF ... Go To Statement

The general form of the If ... go to Statement is:

IF Condition Go To Line Number

Example-1
Write a program to compute the average of 50 students who have 6 subjects.
Solution

```
REM this program for compute the average
I=0
10 Input N$,S1,S2,S3,S4,S5,S6
AV=(S1+S2+S3+S4+S5+S6)/6
Print N$,AV
I=I+1
If I < 50 go to 10
End
```

Example-2
Write a program to compute the (N!) value.
where N!=1*2*3….N Solution

```
REM this program for compute the factorial
Input N
I=1
F=1
5 F=F*I
I=I+1
If I<=n go to 5
Print F
```

4. Compound IF ... then

The general form of compound if ...then is:

Simple relation (and, or) simple relation

The most logical operators are in following table:

Logical operators	Remark
X1 and X2	True if x1 and x2 are true otherwise false.
X1 or X2	True if either x1 or x2 or both true otherwise false.

Example-1

Write a program to compute the Y value where:

X= (A+B) / 2 ,

$Y=X^2+X^{-3}$

if A=1 or B=3

$Y=X^2+3X+5$ if A >2 and B > 4

$Y=X^3+2X^2+X$ otherwise Execute the program to N from A,B values.

Solution

```
Print " A B X Y": print "------------------"
Read N I=1
5       Read A,B
X=(A+B)/2
If A=1 or B=3 go to 10
If A>2 and B>4 go to 20
Y=X^3+2*X^2+X Go to 30
10      Y= X^2 +X-3
Go to 30
20      Y=X^2+3*X+5 print A;" ";B;" ";X;" ";Y
30      I=I+1 If I<=N go to 5
Data 5,1,3,2,4,3,5,1,2,5,4
```

Example-2

Write a program to input 3 numbers then find the maximum one. Solution

```
Input a,b,c
Max=a
If b>a and b>c then max=b
If c>a and c>b then max =c
Print "the maximum is";max
```

Example-3

Write a program ask about triangle sides, and determine if it is a right triangle, and compute it is area and circumference.

Solution

```
Input x,y,z
If x+y>z and y+z>x and x+z>y then ? "right triangle"
C=x+y+z
S=C/2
Ar=sqr(s*(s-x)*(s-y)*(s-z))
Print "the circumference is"C
Print "the area is"ar
```

Examples of BASIC Program

Example: Developed a BASIC program of the area of a rectangle whose length is 10cm and its width is 8cm.

Program

```
10 LET LENT=10
20 LET WID=8
30 LET AREA=LENT *WID
40 PRINT AREA, LENT, WID
50 END
```

Example: Given that x = 450, Develop a BASIC program that generates and displays the sine, cosine, and Tan of x.

Solution

```
10 LET X = 45
20 LET P = SINE (X)
30 LET Q = COS (X)
40 LET R = TAN (X)
50 PRINT P, Q, R
60 END
```

Example

```
10 LET AS = “FIVE HUNDRED.”
20 LET BS = “THOUSAND”
30 LET C$ = A$ + “ ” + B$ + “ NAIRA”
40 PRINTC$ 50 END
```

Summary

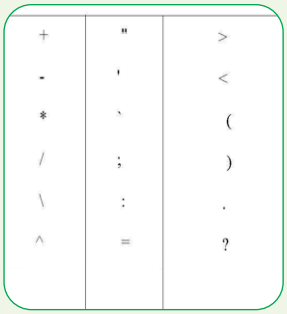
- Dr John G. Kemeny and Thomas Kurtz developed the BASIC language at Dartmouth in 1964
- BASIC stands for "Beginner's All-Purpose Symbolic Instruction Code."
- BASIC is the simplest programming language
- Variables in QBASIC classified into two types
- An expression is the combination of operators, constants and variables that are evaluated to get a result

Activity 1

Compiler and Interpreter

Questions

- What are the steps involved in installing a QBASIC
- List six rules for typing a variable to QBASIC
- Mention five contents in QBASIC



Self Assessment Questions

1. Discuss, sequentially the installation of QBASIC
2. Mention six (6) characteristics of QBASIC
3. Itemize five (5) advantages and four (4) disadvantages of QBASIC
4. State four (4) types of the BASIC character set you to know
5. Give five (5) contents in QBASIC

Tutor Marked Assessment

Use an algorithm to solve the following problems

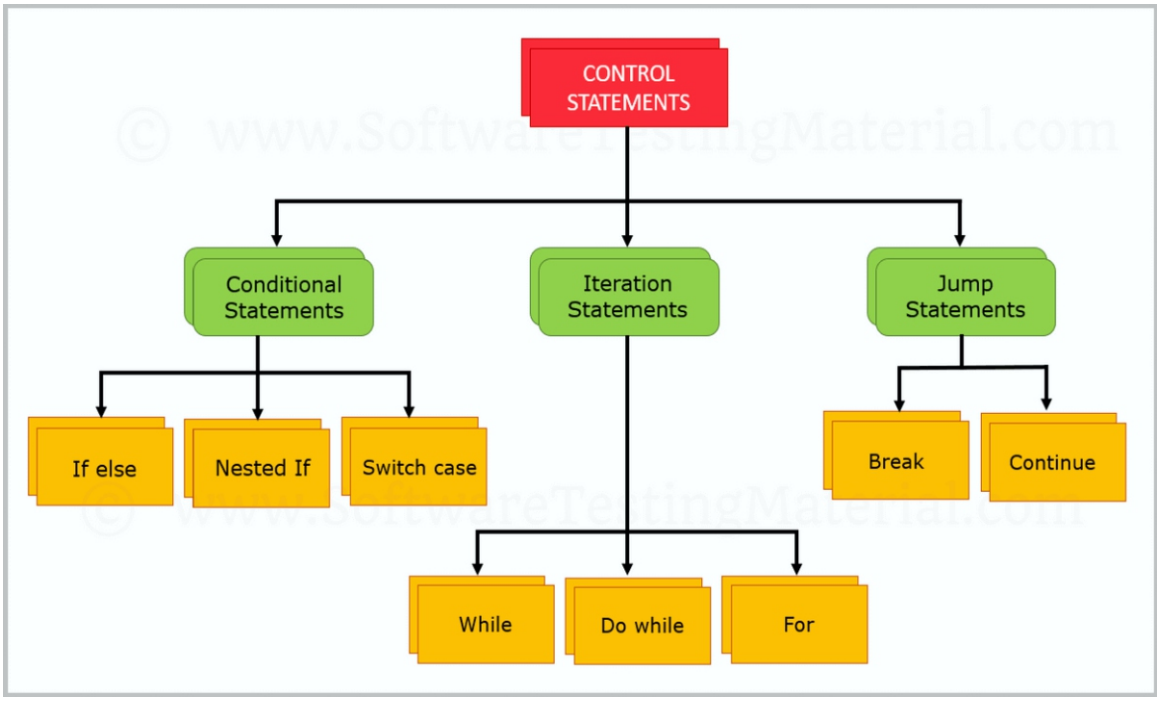
- Draft a menu list program that allows the user to make a wish/choice in a statistical analysis coding system.
- Given that the goal you set for your retirement in your current employment is N850,000. You decided to deposit your pension payment in advance to ECO BANK PLC with payment of N1,000 each month with the interest of a 5.0% interest charge.

Further Reading

- Hu X. Application Value of JAVA Programming Language in Computer Software Development. China Computer & Communication, 2017.
- Ivanova V, Sedov B, Sheynin Y, et al. Domain-specific languages for embedded systems portable software development. Open Innovations Association. IEEE, 2014:24-30.
- Michael Trombetta QBASIC for Students / Edition 1, McGraw-Hill Higher Education, 1993
  Wallace Wang Beginning Programming for Dummies by Wallace Wang, 2001

References

- Badreddin O, Forward A, Lethbridge T C. A test-driven approach for developing software languages. International Conference on Model-Driven Engineering and Software Development. IEEE, 2014:225-234.
- Ivanova, V, Sedov, B, Sheynin, Y, et al. Domain-specific languages for embedded systems portable software development Journal. 2014:24-30.
- Lavazza L, Morasca S, Tosi D. An empirical study on the effect of programming languages on productivity. ACM Symposium on Applied Computing. ACM, 2016:1434-1439.



UNIT 2

Control Structures

Introduction

In this unit, you will be taught statements that are used to implicitly or explicitly control the sequence in which program statements are to be executed are referred to as control structures. Implicitly, by default, the normal order of executing program statements sequentially. That is, statements are executed in the sequence they physically appear in the program. Usually, in programming, statement, or set of statements can be executed repeatedly or selectively based on the truth or falsity of a specified condition. Control structures can be broken down into 3 main structures, namely Sequence, Selection, and Iteration.

Learning Outcomes

When you have studied this unit, you should be able to:

1. Explain sequence control structure,
2. Explain the selection control structure,
3. Explain the iteration control structure

Main Content

SAQ 1

Sequence

5 mins

We use a control structure to represent a sequence of processing steps executed by the computer one after the other in the order as specified by the programmer. These steps can be an arithmetic operation, input/output instructions, assignment operation, sub-program call statements, etc. one basic property of a control structure is that it has only one point of entry and one point of exit.

For example:

Let us consider, the program below computes and prints out the circumference of a circle after reading its diameter from the keyboard.

```
10      PRINT "ENTER THE DIAMETER OF THE CIRCLE"
20      INPUT DIAMETER
30      LET PIE = 3.142
40      CIRCUM = PIE * DIAMETER
50      PRINT "CIRCUMFERENCE = ", CIRCUM
60      END
```

SAQ 2

Selection

3 mins

The selection control structure is a kind of control structure that involves decision making in which one of two or more sets of activities would have to be carried out depending on the truth or falsity of a specified condition. There are 2 main forms of this structure: **Single Selection and Binary Selection**.

Single selection: This is when a decision is to be made about the execution of one or more statements based on the truth or falsity of a specified condition.

The IF-THEN statement: single selection control structure is implemented in structured BASIC by the IF-THEN statement. The format for the IF-THEN statement is :

```
N       IF(Condition) THEN Imperative statement
```

Where N is a statement number (label), and the "CONDITION" represents a test that

is to be performed with the result of true or false. For Example:

25 IF AGE =0 THEN PRINT "NO AGE" 30

……..

In the above program segment, the imperative statement: PRINT "NO AGE" is carried out if S = 0 and the word NO AGE will be printed or displayed on the screen. The control is then passed to the immediate next statement (statement 30). However, if S is not equal to zero control goes straight to statement 30 without executing the imperative statement. Another format of the IF-THEN statement is: **N**

IF(Condition) THEN M

Where M and N are two different statement numbers, when the condition is true, control is transferred to statement M.

The CONDITION in the general format of the IF-THEN statement is usually a relational expression which yields the value true or false. In relational expression, two items are compared with each other using the relational operators.

Relational Operations

We use these operators to make comparisons between two data items. The data items can be numeric or non-numeric data. These relational operators include

SYMBOL	EXAMPLE	MEANING
=	A = B	An EQUALS B
>	A > B	A GREATER THAN B
<	A < B	A LESS THAN B
>=	A >= B	A GREATER THAN OR EQUALS TO B
<=	A <= B	A LESS THAN OR EQUALS TO B
<>	A <> B	A NOT EQUAL TO B

In a condition A < B, the less than is called the relational operator, and A and B are called the operands. In BASIC, the operands of a relational operator must be a legal BASIC expression.

The IF-THEN-ENDIF Statement

In this case, the IF-THEN Statement ends with the ENDIF statement. In this format, it is possible to execute many imperative statements with an IF-THEN statement. The general forms of this statement are:

IF (Condition) THEN Statement 1

Statement 2 Statement n

ENDIF

Where the condition is a specified condition that will yield a Boolean value, and statement 1, statement 2,...., statement n are different imperative statements.

For Example:

```
30      IF A = B THEN 40        A = A + 1
50      B = A – 1
60      B = B – 1
70      PRINT A, B
80      ENDIF
90      END
```

You will observe that in the example above, if A = B, the 4 imperative statements between the IF and the ENDIF will be executed after which control will be passed to statement 80. If, however, A is not equal to B, the 4 imperative statements will be skipped, and the control passed to statement 90.

Binary Selection

You will see, a choice is to be made between 2 alternative sets of actions based on the truth or falsity of a specified condition. In this structure, the given condition is evaluated. If the condition is true, the TRUE ACTION is carried out, and if false, the FALSE ACTION will be performed. Either of the true or false action blocks may represent a sequence control structure when several actions need to be performed.

THE IF-THEN-ELSE

This is a kind of binary selection control structure. It has the following format:

N IF(Condition) THEN statement 1 ELSE statement 2 For example,

10 IF A < B THEN PRINT A ELSE PRINT B

You should be aware that that multiple statements can also be executed when a choice has been made. However, the multiple statements must be separated by a colon.

For example,
10 IF A < B THEN A=2: B=3 ELSE A =3: B=2

THE IF-THEN-ELSE-ENDIF STATEMENT

This is also a kind of binary selection control structure. It has the format:

IF (Condition) THEN Statement 1

Statement 2

…… Statement N

ELSE

Statement N + 1 Statement N + 2

…….. Statement M

ENDIF

Where the CONDITION is a Boolean expression, and statement 1, statement n,……, statement m are different imperative statements.

For example,

```
10    IF A > B THEN 20          X = 2
30    Y = 3
40    ELSE
50    X = 5
60    Y = 6
70    ENDIF
```

2. The program below computes and prints the roots of a quadratic equation of the form $ax^2 + bx + c = 0$, given that real roots exists, if and only if $b^2-4ac \geq 0$.

```
10    INPUT A,B,C
20    D = B * B – 4 * A * C
30    IF D < 0 THEN
40    PRINT "COMPLEX NUMBER"
50    ELSE
60    X1 = (-B – SQR(D))/ (2*A)
70    X2 = (-B + SQR(D))/ (2*A)
80    PRINT "THE ROOTS ARE: X1="X1, " X2 = " X2
90    ENDIF
100   END
```

You should be aware the SQR in statement 60 and 70 is a built-in function that provides the non-negative square root of D.

SAQ 3

ITERATION

4 mins

We use iteration when we need to perform an action repeatedly. In this case, an action is repeatedly carried out until a given condition is satisfied. There are 2 forms of iteration control structure, i.e. Pre Tests Iteration and Post Test Iteration.

In BASIC, there are 3 main statements used for implementing the iteration control structure. They are:

- WHILE – WEND (PRE TEST ITERATION)
- REPEAT – UNTIL (POST TEST ITERATION)
- FOR – NEXT

While - Wend Statement

The pre-test iteration is usually implemented with the WHILE _WEND statement. It has the format:

WHILE (EXPRESSION) # LOOP BODY

WEND

The loop body is repeatedly executed as long as the expression is true. If the expression is not true, the execution continues with the statement following the WEND statement. It is also possible to nest the WHILE – WEND loop in another one to any level.

For example,
The program prints the first ten even numbers.

```
10      LET E = 0
20      LET K = 0
30      WHILE (K < 10)
40      E = E + 2
50      PRINT E
60      K = K + 1
70      WEND
80      END
```

REPEAT-UNTIL STATEMENT

This statement can be used to implement a post-test iteration. It has the format:

REPEAT

#LOOP BODY UNTIL (EXPRESSION)

In this case, it first executes the loop body before testing the validity of the condition. The loop body is repeated for as long as the condition is valid.

For example, let us consider the The program below prints the first 10 even integers.

```
10      LET E = 0
20      LET K = 0
30      REPEAT
40      E = E + 2
50      PRINT E
60      K = K + 1
70      UNTIL (K = 10)
80      END
```

FOR-NEXR STATEMENT

A loop can be created with the use of a FOR-TO –STEP statement. This is used to repeat a statement or set of statements a specified number of times. The number of repetitions must be known ahead of the loop creation.

e.g. FOR I=1 TO 10

This starts with the variable I like 1 and then continuously increases I by 1 until I get to 10. The STEP is used to tell the increment on the variable I each time the loop is executed.

e.g. FOR I=1 TO 10 STEP 2

With this, I am increased by 2 after each execution. The running variable, I, can be assigned negative numbers and fractions. Also, the STEP can be a negative number if the value of I is to be decreased rather than increased.

e.g. 30 FOR I=-1 TO 5 STEP 2

30 FOR I=1.5 TO 10 STEP 0.5

30 FOR I=6.5 TO 0 STEP -0.5

In closing a For loop, we use the NEXT statement. The loop consists of all the statements between the FOR-TO-STEP statement and the NEXT statement. The NEXT statement consists of a statement number, the NEXT keyword, and the running variable used.

e.g.

30 FOR I=1 TO 10 STEP 2

…

70 NEXT I

All statements between line 30 and 70 would be executed before the running variable is increased/decreased. After incrementing/decrementing, the statements are executed again until the ending value (10 in this case) is reached.

Rules to be followed in constructing a FOR loop

The running variable can appear within the set of statements in the loop, but its value cannot be altered within the loop.

If the initial and final values of running the loop are equal, and the step value is positive and non-zero, then, the loop will be executed just once.

The loop will not be executed under the following conditions:

The initial and final values of the running variable are equal, and the step size is zero.

The final value of the running variable is less than the original value, and the step size is positive.

The final value of the running variable is greater than the original value, and the step size is negative

Control can be transferred out of a loop but not into the loop. e.g.

```
30      FOR I=1 TO 10 STEP 2
40      GOTO 100
...
70      NEXT I
```

You can have the above example but cannot have this next one:

```
30      FOR I=1 TO 10 STEP 2
40      LET X=4+Y
...
70      NEXT I...
90      GOTO 40
```

This is because control is being transferred from outside the loop into the loop, which is not allowed.

Also, it is good programming practice to indent the code within the FOR loop for clarity.

```
30      FOR I=1 TO 10 STEP 2
40      LET X=4+Y
50      LET B=A+10 60 NEXT I
```

For Example, assuming we are asked to Write a program that provides the solution of an unknown number being raised to the power of another number. Let the user provide both numbers.

Solution

Here, we are using the same task as encountered previously but with a FOR loop.

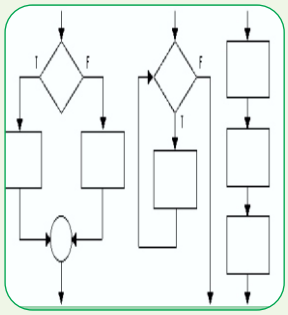
```
10 REM This program computes the exponential value of a number
20 CLS
30 PRINT "This program calculates the exponent of a number."
40 PRINT "The first number entered is the number."
50 PRINT "The second number entered is the power."
60 INPUT X, Y
70 LET EXPO=X              'let the result be x at first
80 FOR I=1 TO Y-1 STEP 1
90 LET EXPO= EXPO*X
100 NEXT I
110 PRINT X; "raised to the power of"; Y; "equals"; EXPO
120 END
```

Activity 1

Control Structures

Questions

- Define Control Structure
- What is iterative control structure



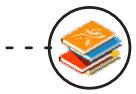
Self Assessment Questions

1. What is a control structure?
2. State the three (3) types of control structure?
3. Define the iterative control structure?

Tutor Marked Assessment

Use an algorithm to solve the following problems

- Itemize and discuss the types of control structure you know
- With given examples differentiate between IF-THEN and IF-THEN-ELSE statement

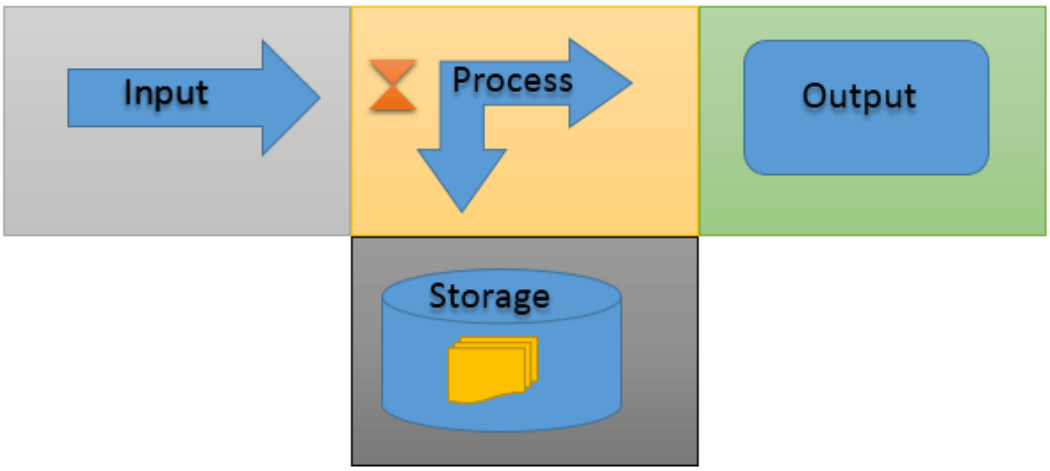


Further Reading

- Lavazza L, Morasca S, Tosi D. An empirical study on the effect of programming languages on productivity. 2016:1434-1439.
- Garrido J M. Improving Software Development for Embedded Systems[C]// Southeast Conference. ACM, 2017:231-234.
- Long M, Amp H R. An Analysis of the Algorithm for "Spending 100 Dollars on 100 Chickens" in C Programming Language. Software Engineering, 2017.
- Michael Trombetta QBASIC for Students / Edition 1, McGraw-Hill Higher Education, 1993
- Wallace Wang Beginning Programming for Dummies by Wallace Wang, 2001

References

- Bakpo. F.S. (2005). Introduction to Computer Science and Programming in QBASIC. Third edition Enugu: Magnet Business Enterprises.
- Bezanson, W.R. (1975). Teaching structured programming in Fortran with IFTRAN. SIGCSE 7(1), 196 - 199.
- Fenichel, R.R. (1970). A program to teach programming. Comm. ACM 13(3), 141- 146.
- Winetzhammer S, Westfechtel B. Compiling graph transformation rules into a procedural language for behavioural modelling. International Conference on Model-Driven Engineering and Software Development. IEEE, 2015:415-424.



UNIT 3

System Built-in Functions and Manipulation

Introduction

In this lesson, I will introduce you to a system of built-in functions and subprograms in QBASIC. Operation on string data, a detailed explanation was given on built-in string functions with given examples. I will conclude by explaining the concepts on subprogram function with given examples. Certain operations that are commonly carried out by many computer users when solving their problems have been pre-coded by computer manufacturers. For example, computing the square root, logarithm, or sine of a number are common operations in solving mathematical problems. These pre-coded routines are called system built-in functions and are normally part of every programming language.

Learning Outcomes

When you have studied this unit, you should be able to:

1. Explain built-in functions,
2. Explain string manipulation,
3. Explain concepts of subprogram functions

Main Content

System Built-in Functions

6 mins

BASIC also provides programmers with such flexibility and programming convenience. For example:

```
10      LET R = SQR (49)
20      PRINT R
30      END
```

The result of the code segment above will be 7. In statement 10, the square root function, SQR, is used to compute the non- negative square root of 49, which is 7. In this example, SQR is called the FUNCTION NAME, and 49 is called the ARGUMENT, PARAMETER OR SIGNATURE of the function. Note that the argument is enclosed within parenthesis. It generally was written as:

FUNCTION_NAME (ARGUMENT)

The argument may be a NUMBER, a valid IDENTIFIER, any legal BASIC EXPRESSION, or another function.

Common built-in mathematical functions include:

SQR(X) - Computes the non-negative square root of X.

ABS(X) - Computes the absolute value (non-negative) of X.

RND(X) - Generates a random number.

SIN(X) - Computes sine of X.

COS(X) - Computes cosine of X.

TAN(X) - Computes tangent of X.

SIN(X) - Computes sine of X.

LOG(X) - Computes natural log of X.

EXP(X) - Computes the exponential value of X. Etc.

Operations on String Data

Several operations, aside from the arithmetic operations, are defined on string data. Some of these are:

Assignment to String Variable: Strings variables can be assigned specific string values. For example, consider the bellow program.

```
10      LET A$ = "HELLO WORLD."
20      PRINT A$
30      END
```

"HELLO WORLD" will be printed on the screen without the quotes. You can also input character string by the use of reading and DATA statements.

For Example,

```
10      READ NAME$
20      DATA    "HELLO WORLD"
30      PRINT NAME$
40      END
```

This prints HELLO WORLD.

Logical Comparison: Arithmetic operations cannot be performed on string data, but Comparison can be made. For example:

```
10      LET A$ =" GOOD."
20      LET B$ = "MORNING"
30      IF A$ = B$ THEN PRINT "YES."
40      IF A$ <> B$ PRINT "NO."
50      END
```

Concatenation: Two or more strings may be concatenated (joined together) using the addition (+) symbol.

```
10      LET A$ ="GOOD"
20      LET B$ = "MORNING" 30          C$ = A$ + B$
40      PRINT C$
50      PRINT C$ + "NIGERIA"
60      END
```

Built-in String Functions

Major operations that can be carried out on string data apart from those mentioned above are done through the BASIC string functions, i.e. built-in functions whose argument must be a string.

LEN FUNCTION: This is used to get the length of (i.e. the number of characters in) a string.

```
30      LET A$ =" HELLO WORLD."

40      PRINT LEN(A$)

50      END
```

From the above code segment, the output will be 11. The LEN function returns an integer equal to the number of characters in a string argument, i.e., its length.

LEFT$: This is used to get the substring composed of the leftmost specified number of characters of a string argument.

LEFT$(A$, N) returns the leftmost N characters of A$.

For example,

```
10      A$ = "BETTER BY FAR."
20      PRINT LEFT$(A$, 3)
```

The above code segment will print BET.

RIGHT$: This is used to get the substring composed of the rightmost specified number of characters of a string argument. RIGHT$(A$, N) returns the leftmost N characters of A$.

For example,

```
10      A$ = "BETTER BY FAR."

20      PRINT RIGHT$(A$, 3)
```

The above code segment will print FAR.

MID$: This can be written as MID$(A$, N).This returns a substring at the nth position of

a string (say A$) to the end (last character) of A$. MID$(A$,N,M) is also a valid format for MID$. It returns M characters starting from the nth position of the string A$.

For example:

LET A$ = ”BETTER BY FAR” MID$(A$,8)=”BY FAR” MID$(A$,8,2)=”BY”

VAL: The VAL function gives the numerical value of strings of digits that means it only captures digits. Whenever it encounters a NON-DIGIT CHARACTER, it stops looking. For example:

A$ = ”12345”; VAL(A$) = 12345 B$ = ”789GH”; VAL(B$) = 789 C$ = ”ADE06”; VAL(C$) = 0

STR$: This function changes a numeric data to a string data. For example, NUM = 12345, therefore, STR$(NUM) will result in a string literal “12345”.

There are quite many other built-in functions that cannot be strictly classified as either mathematical functions or string functions. Few common ones are:

ASC(X$): we use this function to get the ASCII code (a numeric value) of the first character of its argument, X$. for example, if X$= “C” then PRINT ASC(X$) will print out 67 which is the ASCII code of C. in the case of ASC(“BETTER BY FAR”), 66 will be displayed on the screen due to fact that, it is only one character (the first character) that will be converted to ASCII code. ASCII stands for American Standard Code Information Interchange.

CHR$(N): This function returns a single character (ASCII code), whose ASCII value is the argument, N, where is between 0 and 255 inclusively. CHR$ is the more like the opposite of ASC$ function and is very useful in outputting ASCII control characters as part of a print statement. For example, PRINT CHR$(88) will display the character X on the screen. Since 88 is the ASCII code for character X. It should be noted that the ASCII code is case sensitive, i.e., the ASCII code for “A” is not the same as “a” and vice versa.

INKEY$: This function will assign to a string variable any character that is typed at the keyboard without having to press the Enter key. The character typed will not be displayed on the screen. For example,

```
10      CH$ = INKEY$
```

If character A is entered via the keyboard, A will be stored in CH$. The INKEY$ function

is often used whenever a program has to pause for a key to be pressed before it continues

execution.

INPUT$(N): The INPUT$ function is similar in operation to the INKEY$ function. It is used to return a string of N characters typed on the keyboard. The string typed will be displayed on the screen, but it will be stored as an N-character string typed at the keyboard.

STRING$(M, N): This function returns an M-character long string of the character whose ASCII value is N. For example,

```
10      LET C$ = STRING$(7,65)
```

The above code segment will store "AAAAAAA" in C$ since 65 is the ASCII code for A.

Concept of Subprograms

5 mins

SAQ 1

I want you to know that subprogram is a small self-contained and usually separately compiled sequence of instruction, (program) for accomplishing a well-defined task? This task is usually a component of another larger main program. A subprogram cannot execute on its own without the help of another program which acts as a main (calling) program. However, it does not have to and does not usually reside in the same contiguous memory with the main program. When a reference to a subprogram is encountered in a program that is being executed, execution will be suspended, and the transfer of control made to the subprogram. When the execution of the subprogram is complete, the control will be returned to the main (calling) program at the point its execution was suspended.

Types of Subprogram

You might have observed from our previous discussion that there are types of subprogram. These are functions and subroutines. Many HLL makes provision for certain common functions which are usually referred to as built-in functions. The mathematical functions are a typical example. When a required function is not in-built in the system, most programming languages, allow programmers to define their function called user-defined functions.

The main difference between function and subroutine is that subroutine calls are a statement within a program, whereas function calls appear as an expression within a statement in a program. Also, the function subprogram usually returns only one value and the control to the caller, whereas the subroutine can return as many values as possible together with the system control back to the caller.

User-Defined Functions

In addition to what we have discussed about standard built-in functions, the programmers make functions of his/her own, utilizing definition. The DEF statement will be used in this case. For a single line function definition, the procedure is as follow:

Choose a three-letter name beginning with FN as a name for the function. E.g. FNA, where A is the function name.

If you Follow the 3 letters with an argument enclosed within parenthesis. E.g. FNA(X). where x is the argument or parameter of the function A., it is also known as dummy(formal) argument(parameter).

Now let us define the function by writing a legal BASIC expression showing how the argument is to be used or manipulated. e.g. 2*x^2 + 3*x^2

Use the DEF statement (DEFine) and write the function definition. E.g. 10 DEF FNA(X) = 2*x^2 + 3*x^2

Statement defining functions are usually placed at the beginning of a program. Though, they may be placed elsewhere within the program. For example,

```
10        DEF FNR(A,B,C) = SQR (B^2 - 4 *A*C)
20    INPUT      A,B,C
30          LET D = FNR(A,B,C)
40          IF D < 0 THEN
50          PRINT "COMPLEX NUMBER"
60           ELSE
70         X1 = (-B + SQR(D))/(2*A)
80         X2 = (+B + SQR(D))/(2*A)
90         PRINT X1, X2
100        ENDIF
110        END
```

The number and type of the actual parameter must correspond to those of the formal parameter.

Multiple-Line Function

In a situation whereby our intended function is too complex to appear as a single and simple expression, the multiple line function method is used. In this case, the DEF statement indicates the beginning of the function, and the FNEND indicates the end of the function. All other statements (s) in between DEF and FNEND are considered to be part of the function definition. The opening statement contains only the word DEF, the function name and the arguments, if any, enclosed in parenthesis. The function name can appear in the statement of the definition. For example,

```
////// DEFINITION OF FUNCTION FACTORIAL
10    DEF FNF(N)
15    LET FNF = 1
20    IF N = 0 THEN 80
30    FACT = 1
40     FOR I = 1 TO N
50       FACT = FACT *I
60       NEXT  I
70     LET FNF = FACT
80     FNEND
//////// ANOTHER CODE SEGMENT TO CALL THE FUNCTION ABOVE
110 INPUT K
120 PRINT "FACTORIAL OF ", K ", IS", FNF(K)
130 END
```

Subroutines

A subroutine in BASIC is a section of a program that is entered by a GOSUB statement, and it is exited using a RETURN statement, which sends the program control to the statement immediately after the GOSUB. It has the format: nn GOSUB mm

where mm is a line number designating the point at which the subroutine starts. The format of the RETURN statement is in RETURN. They may be several GOSUB statements in a program. Subroutines may as well be placed anywhere in a program, but It is advisable to place them just before the END statement. For example,

```
10      INPUT K

20      GOSUB 60: REM STATEMENT 50 BEGINS THE
SUBROUTINE

30      PRINT "FACTORIAL OF", K ", IS", FACT

40      STOP

50      REM THE SUBROUTINE BEGINS HERE

60      FACT = 1

70      IF K = 0 THEN 110

80      FOR I TO K

90      FACT = FACT * I

100     NEXT I

110     RETURN

120     END
```

If you notice the statement 20 contains 2 statements. GOSUB 60 AND THE REM statement separated by a semi-colon. The semi-colon is used to separate 2 or more statements that are written in a line having the same line number.

Summary

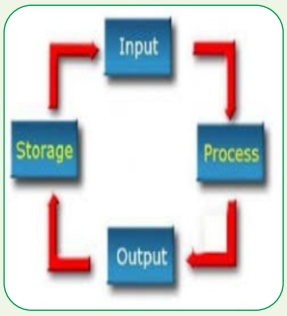
- In this lesson, you have learnt that:
- Certain operations that are commonly carried out by many computer users when solving their problems have been pre-coded by computer manufacturers
- Subprogram cannot execute on its own without the help of another program which acts as a main (calling) program
- DEF statement indicates the beginning of the function
- FNEND indicates the end of the function

Activity 1

System Built-in Functions and Manipulation

Questions

- Define subprogram
- What is subroutine



Self Assessment Questions

1. What is a subprogram?
2. Mention three (3) examples of built-in functions

Tutor Marked Assessment

Use an algorithm to solve the following problems

- Itemize and discuss three (3) types of a subprogram in BASIC
- With adequate examples for each, explain any three built-in functions in BASIC

Further Reading

- Hu X. Application Value of JAVA Programming Language in Computer Software Development. China Computer & Communication, 2017.
- Ivanova V, Sedov B, Sheynin Y, et al. Domain-specific languages for embedded systems portable software development. Open Innovations Association. IEEE, 2014:24-30.
- Michael Trombetta QBASIC for Students / Edition 1, McGraw-Hill Higher Education, 1993
- Wallace Wang Beginning Programming for Dummies by Wallace Wang, 2001

References

- Badreddin O, Forward A, Lethbridge T C. A test-driven approach for developing software languages. International Conference on Model-Driven Engineering and Software Development. IEEE, 2014:225-234.
- Ivanova, V, Sedov, B, Sheynin, Y, et al. Domain-specific languages for embedded systems portable software development Journal. 2014:24-30.
- Lavazza L, Morasca S, Tosi D. An empirical study on the effect of programming languages on productivity. ACM Symposium on Applied Computing. ACM, 2016:1434-1439.