KAPLAN HIGHER EDUCATION ACADEMY

KAPLAN HIGHER EDUCATION INSTITUTE

Study Guide

Diploma Programme

Fundamentals of Computer Systems

v2.0

Message to Student

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

Dear Kaplan Student,

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

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

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

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

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

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

Krishna Rajulu
Academic Dean, Kaplan Singapore

Kaplan Desired Graduate Attributes

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

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

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

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

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

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

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

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

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

Independent and reflective practitioner

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

Embedded within the desired graduate attributes are the following skills:

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

Table of Contents

About this module

This module presents fundamental topics in understanding a computer system. It will cover the **five (5)** basic operations of a computer, the systems architecture , including the microprocessor, and the different types of memory storage. An introduction to the representation of numbers (binary, octal and hexadecimal) in a computer, Boolean algebra and logic circuits that are essential to understanding how the microprocessor works is included. The basic functions of an operating system in managing the computer systems is also covered. The module also introduces good program design, programming styles, and structured program development using the 'C' high-level programming language.

Module Learning Outcomes

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

- Describe the FIVE (5) basic operations of a computer;
- Describe the components of a microprocessor and how they work together in executing an instruction;
- Describe the different type of memory storage;
- Describe the FIVE (5) basic functions of an operating system;
- Show practical skills in number system conversions, simplifications of Boolean expression;
- Explain, design, and implement logic circuit diagrams;
- Use good programming practices in writing 'C' program;
- Design and write 'C' programs to solve problems.

Overview of Learning Resources

Recommended reading:

Computer Systems: A Programmer's Perspective, Global Edition 3/e, ISBN 9781292101767 by Randal E. Bryant and David R. O'Hallaron, published by Pearson Education Limited © 2016

Discrete Mathematics, Global Edition 8/e, ISBN 9781292233703 by Richard Johnsonbaugh, published by Pearson Education Limited © 2019

C How to Program, Global Edition 8/e, ISBN 9781292110974 by Paul Deitel and Harvey Deitel, published by Pearson Education Limited © 2016

Other sources:

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

Scheme of Work

FT	PT	TOPICS
1	1	01 System functional Components, Characteristics, Performance, Interactions and use in IT • Hardware Organization of a SystemThe Central Processing Unit (CPU) • The Central Processing Unit (CPU) • CPU - putting all the components together • The data storage hierarchy
2		02 Operating Systems • The operating system manages the hardware • Task Management • Threads and multi-threading • Memory Management - Virtual Memory • File Management • User-Interface • Utilities
3	2	03 Computer Memory and Storage • Basic building block of computer storage • Distinguish between computer's memory and storage • Registers memory • Cache memory • RAM memory • Other types of memory • Data storage • Disk Capacity • Connecting Input/Output (I/O) devices
4	3	04 Digital Logic • Introduction to number systems and conversions • Boolean algebra and Synthesis of circuits • Applications
5		
6	4	05 Introduction to C Programming • A simple C program: Printing a line of text • Another simple C program: Adding two integers • Basic I/O • The C basic language syntax • Memory concepts • Basic Data types • Arithmetic in C • Decision Making: Equality and Relational Operators • Secure C programming Revision Topics 1-5
7	5	06 Structured Program Development in C • Algorithms • Pseudocode • Formulating Algorithms with top-down, stepwise refinement • Control Structure • The if statement • The if.. else statement • The switch.. case statement Revision Topics 1-5

8	5	07 C Program Control • Iteration Essentials • Counter-controlled repetition • The for() loop structure • The While loop structure • The do.. while loop structure • Nested loop • Logical operators • Confusing the relational operator Equality and the Assignment operators
9	6	08 C Functions Part I • Modularizing Programs in C • Functions in C • Function Definitions • The Math library Functions
10		09 C Functions Part II • Function Prototypes • Function call stack and stack frames • Header Files • Passing arguments to function • Storage classes and scope rules
11		
12	7	10 Arrays • Arrays in C • Defining Arrays • Calculating the size of array • Static and Dynamic Arrays • Array bound checking in C • Working with arrays • Using character arrays to store and manipulate Strings
13		
14		Revision Topics 6-10 Assignment Draft Submission and Consolidation

Assessment Matters

Assessment Overview

Assessment 1: CA Quiz

Weightage: 20% (20 marks)
Date: To be confirmed
Duration: 10 minutes per quiz
Format: 5 MCQs per topic

Assessment 2: Individual Assignment

Weightage: 80% (100 marks)
Date: Lesson 14 (FT) / Lesson 7 (PT)
Citation Format: APA v.7
References: Module Specific

Important Policies

Penalties for Plagiarism

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

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

Penalties for late submissions

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

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

Assignments and Kaplan Learning Management System

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

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

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

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

This page intentionally left blank

Study Guide

Topic 01 – System functional Components, Characteristics, Performance, Interactions and use in IT

Lesson Learning Outcomes

1. Identify the **FIVE (5)** basic operations of a computer
2. Explain the types of hardware associated with each operation
3. Explain how they work together to accomplish a user task
4. Explain the role of the Central Processing Unit (CPU)
5. Describe the components of the CPU
6. Explain how they work together in executing an instruction
7. The data storage hierarchy

1.1 Hardware Organization of a System

A computer system is an electronic device that accepts input from the user, processes it and display the results. The computer systems allow the user to accomplish a wide range of tasks from preparing a management report to spending endless hours playing online games. All computer systems are comprised of five (5) basic operations: input, process, output, storage, and communication.

The input operation:

The input operation provides a means to facilitate the user to communicate with the computer system. For example, the keyboard allows the user to express his intention to the computer directly. The keyboard is an input device that translates the human language into a language that the computer can understand. This is accomplished by referring to a translation table, i.e., ASCII (for English only) or Unicode (for other languages). The ASCII table paired each character to its assigned computer language. Other input devices that feeds data into the computer include the mouse, scanner, graphic tablet, bar code reader, touch screen etc.

The output operation:

The output operation communicates the result of the processing in a form that the user can understand. The processed information is in the form of 1s and 0s, is not ready for human consumption and needs to be transform into the human language. Like the input operation, this is accomplished by referring to the ASCII or Unicode tables. Output devices include the display monitor, speaker, printer, multimedia projector, graphics card etc. Please take note that some hardware devices have combined both the input and output operations, i.e., your smart phone screen is both an input and output device.

The process operation:

The output operation performs basic arithmetic operations and logical comparison on the user's data and convert them into the desired information. The Central Process Unit (CPU) or microprocessor accomplishes this. The CPU focus mainly on processing the user's data and depends on the other four (4) operations to support it. Examples of CPU includes the Itanium family of microprocessor from Intel and Opteron family of microprocessor from AMD etc.

The storage operation:

The storage operation allows data and instructions to be stored before, during and after processing. Data and instructions entered by the user are stored temporarily in storage (memory) before being passed to the CPU for processing and the output operation. There are TWO (2) types of memory: Primary and Secondary. Primary memory is temporary and volatile, i.e., they do not retain their contents after you have turned off your computer. Examples of primary memory includes RAM, Cache and Register etc. Secondary memory is permanent and non-volatile, i.e., they do not lose its contents even after you have turned off the computer. Examples of secondary memory includes ROM, hard disk, USB memory stick etc. In general, primary memories are faster and more expensive than secondary memories.

The communication operation:

The output operation allows the user to transmit and receive data and/or instructions from other computers or devices on the network. Examples of communication devices includes the network interface card (NIC) and modem on your computer.

Putting them together

- Input operation: you created a document
- Process operation: you issued commands to change the fonts, paragraphs, insert pictures etc.
- Output: you are looking at the display screen while editing your document
- Storage: you saved a copy of the document in your hard disk
- Communication: you emailed a copy of the document to your friend.

1.2 The Central Processing Unit (CPU)

The most important component of the computer system is the Central Processing Unit (CPU). The CPU is responsible for fetching, decoding, and executing of the instructions that constitute a software program. These instructions are built into the CPU and are collectively known as the instruction set. Each family of CPU has their own instruction set. The main components inside the CPU are the Control Unit (CU), the Arithmetic Logic Unit (ALU), Registers and Cache memories. The CPU is connected via electronic pathways (system bus) to other components of the computer system.

The Control Unit (CU):

The CU controls all the activities and coordinates the flow of data, instructions, and commands, inside the CPU. It fetches the instruction from RAM (where the instructions are previously loaded from the hard disk), checks that it is valid for execution, and loads them into the appropriate register before passing the control to the ALU for execution.

The Arithmetic login Unit (ALU):

The ALU carries out arithmetic and logical comparisons on the data according to the instruction. Arithmetic operations include the basic operations like, addition, subtraction, multiplication, and division. Logical operations include comparing data and deciding the next set of instructions to execute.

Registers and Cache memories:

Before, during and after execution, the data, instructions, and results are stored within the CPU. These are the fastest storage areas inside the CPU and are known as registers and cache memories. The registers are high-speed storage areas that temporarily store the data, instructions, and result of the processing. There are limited registers in the CPU and thus the contents of the registers are frequently swapped out to RAM to make way for other instructions and data. Cache memories are also very fast memory areas inside the CPU, the main purpose of using cache memories is for speed optimization and they are positioned between the registers and RAM memories. Frequently used data and instructions are stored in the cache instead of RAM, which is slower than cache.

1.3 CPU - putting all the components together

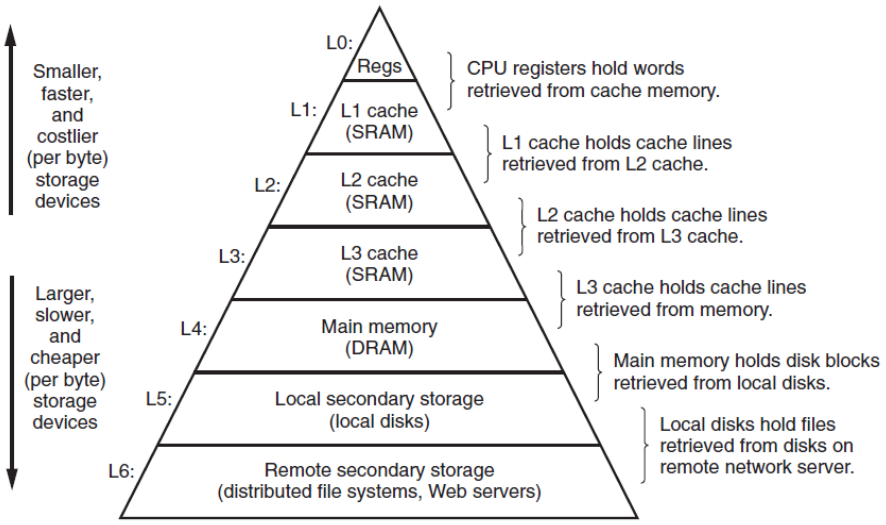
Now that we have a good understanding of the components that makes up the CPU, how does all the components in the CPU work together in executing an instruction? It is useful to discuss using the fetch-execute cycle. The fetch-execute consists of FOUR (4) stages, Fetch, Decode, Execute and Store.

Fetch stage:	The control unit fetches the next information to be executed from memory.
Decode stage:	The control unit checks if the fetch information is data, instruction, or a memory address and takes the following actions: • If it is data, it will put it into the data registers. • If it is an address, it will go and fetch the information from the memory location as specified by the address and check again. • If it is an instruction, it will check again if it is a valid instruction. ▪ If it is an invalid instruction, the CU will stop execution and display an error message. ▪ If it is a valid instruction, the CU will load it into the instruction register.
Execute stage:	The CU will inform the ALU to execute the instruction. The ALU will execute the instruction on the data.
Store stage:	The result of the execution will be stored in the appropriate register.

The fetch-execute cycle will repeat until the user terminate the program or shut down the computer.

1.4 The data storage hierarchy

The registers memories in the CPU are very fast (nanoseconds) but the speed of the hard disk is very slow (typically rounds per minute). Fetching the instructions directly from the hard disk for the CPU is very slow and the overall performance of the computer will degrade. Due to the disparity in speed, the memories and storage are arranged in a hierarchy. The notion of a data storage hierarchy helps us to visualise how we can compensate for the differences in speed. More of these memories and storage characteristics will be discussed in topic 3.


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
CPU registers hold words retrieved from cache memory.
L1:
L1 cache (SRAM)
L1 cache holds cache lines retrieved from L2 cache.
L2:
L2 cache (SRAM)
L2 cache holds cache lines retrieved from L3 cache.
L3:
L3 cache (SRAM)
L3 cache holds cache lines retrieved from memory.
L4:
Main memory (DRAM)
Main memory holds disk blocks retrieved from local disks.
L5:
Local secondary storage (local disks)
Local disks hold files retrieved from disks on remote network server.
L6:
Remote secondary storage (distributed file systems, Web servers)

References

Bryant, R. E., O'Hallaron, D. R., Manasa, S., & Tahiliani, M. P. (2016). Computer systems: A programmer's perspective.

Topic 02 –Operating Systems

Lesson Learning Outcomes

1. Identify and explain the **TWO (2)** basic kinds of software
2. Explain the relationships between the user, software, and hardware
3. Explain the role of device drivers
4. Explain the role of software utilities
5. Explain the role operating systems
6. Explain the **FIVE (5)** basic functions of an operating systems

2.1 The Operating system manages the hardware

What is Software?

In general, software refers to the computer instructions that the CPU can understand to manipulate the associated computer hardware to achieve a specific goal for the user. These finite sequences of instructions are known as a program. There are many programs running in the computer system and collectively they are known as software. There are two main types of software: application and systems software:

- Application software:
  Application software is any collection of software that helps the user to accomplish a user-oriented task, i.e., editing a document (word processing), tabulating monthly expenses (spreadsheet), surfing the WWW (web browser) etc.

- Systems software:
  Systems software is a collection of software that manages the hardware of the computer and interacts with the application software to complete a specific user's task, i.e., saving a document file, sending an email etc. Systems software comprises of the following components: device drivers, utilities, and operating system.

Device Driver

When you connect a new printer to your computer, you cannot send print jobs to the printer initially. You need to install the printer's driver program first before you can print. The printer's driver program is an example of a device driver. Device driver is a category of software that allows hardware devices to interact with the operating system.

Utility software

Utility software performs specialized task to maintain the computer system to perform optimally. These utilities are typically installed and control by the users. Some examples of utility software are, file compression, backup/restore, anti-virus software etc.

Operating systems

The operating system controls the hardware, manages the available resources, and coordinates the various application programs for the user. Application programs are not permitted to access, manipulate the hardware directly, and must request the service of the

operating system to do it. Some examples of popular operating systems are Microsoft Window 10, IOS, Android, Unix, Linux etc.

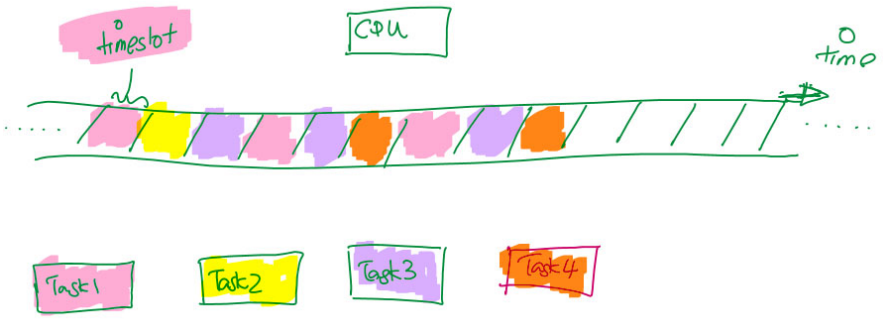
All operating systems perform FIVE (5) basic functions:

- Task management
- File management
- Memory management
- User Interface
- Utilities

2.2 Task Management

The early generation of computers can execute only one program at a time, for example, when you send a print job, you have to wait until the print job is completed before you can proceed with other tasks. In contrast, modern computer systems allow you to run different programs at the same time. Task management (also known as processor management) manages the scheduling and allocation of system resources for the programs running on the computer. A program in execution is called a process, i.e., editing a document using a work-processing program, sending a print job to the print spooler etc.

There are many processes (programs) executing at the same time and each process will competes for CPU resources. However, the CPU will divide its time (time slots) and allocate them to the different processes, i.e., the CPU multiplexes all the processes and execute each at a time for a given time slot.



2.3 Threads and multi-threading

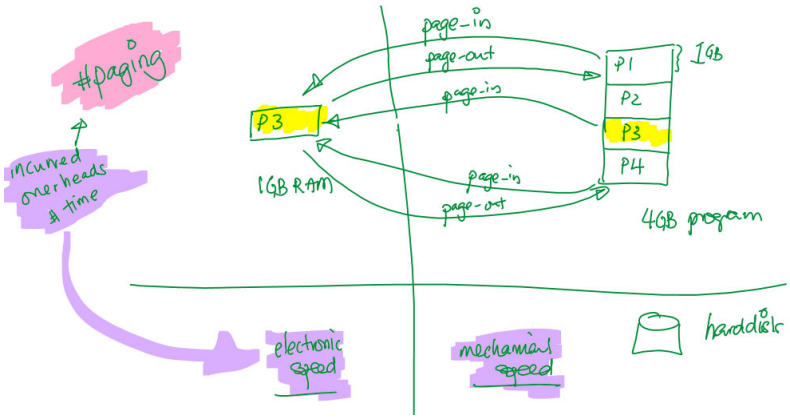
A process is a program that performs a task at a time – also known as a single thread of execution. However, modern CPUs are made of multiple cores and as such, can facilitate multiple threads of the same process to be executed simultaneously. In addition, threads of different processes can also be multiplexed.

> For example, a web browser may be implemented as a process with several threads – one for retrieving an image, another for fetching information from the Internet, another for processing the image etc.

2.4 Memory management – Virtual Memory (VM)

The operating systems manages all the resources in the system. One of the most important is memory management. Memory management controls how memory is accessed and maximizes the available memory and storage. It keeps track of where the programs and data are residing in memory. All user-installed programs (including the operating system files) resides in secondary storage, i.e., the hard disk. Due to the great disparity in speed between the CPU and the hard disk, all programs to be executed must first be loaded into RAM memory. However, some programs are very large, cannot be fully loaded into available RAM memory, and therefore cannot be executed. One solution would be to constrain the size of the program to be smaller than available RAM memory but that would be impractical as the cost of storage continued to decline. A better solution would be to implement virtual memory (VM).

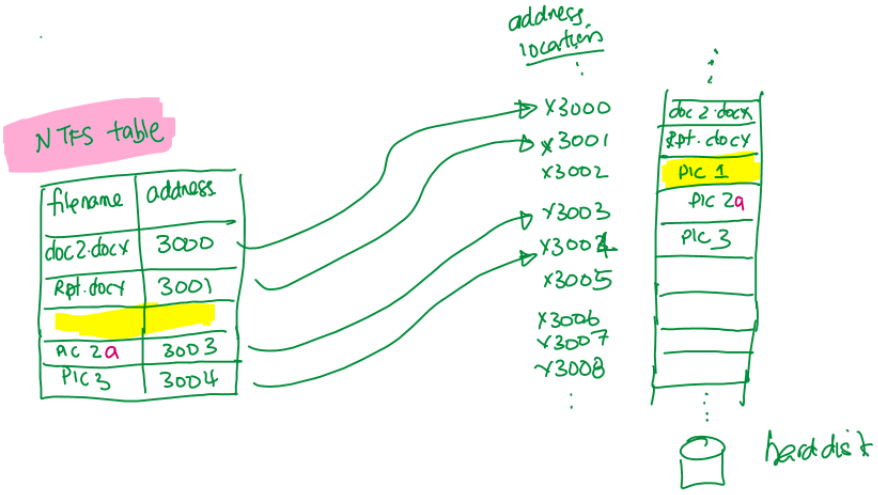
VM allocates spaces on the hard disk to supplement available RAM memory to run programs. When the operating system runs a program that is larger than available RAM, it will allocate portion of the hard disk as working storage. The program is then subdivided into smaller sections (called pages) and some of the pages will be brought into RAM while the rest will reside in the working storage area. The CPU will execute the instructions residing in RAM. When an instruction is required that is not found in RAM, it will swap some of the pages in RAM with the pages in the working storage (where the instruction is located). In this manner, VM can process large programs without the constraints of available RAM.



When VM is implemented, parts of the program are in physical RAM while the rest are in the hard disk as virtual memory. By swapping, the pages from RAM to hard disk will incur a speed overhead and thus the overall performance of the computer will be slowed down eventually.

2.5 File management

The file management function manages the creation, deletion and access controls of files and data in the storage devices. It ensures that files and data in storage are available when needed and protected from unauthorized access. When a file is created, an entry is created in a special system table (FAT or NTFS), that includes the file name and the physical address in the storage where the file is written into. The file name and the physical address forms a map for all the files and data stored in that storage device. When a file is deleted, the entry for the deleted file is removed from the system table. The Window file explorer takes file map (FAT/NTFS) and display them graphically on the screen for the user.



2.6 User interface

The user interface provides a means for the user to articulate his intention to the computer system. There are different types of user interface, the most common being:

- Graphical user interface (GUI)
- Command line interface
- Menu driven interface

Graphical user interface (GUI) display icons, windows, pull down menus on the screen and allows the users to directly manipulate them (touch pad, screen, or mouse) to express their intention. Most useful for novice users.

Command line interface requires the user to enter text-based commands to the computer to perform basic tasks. An example is the Window command prompt. It is not meant for novice user as it places a heavy burden on the user cognitive system – you need to memorise all the commands by heart.

Menu driven interface display a list of options that the user can select from, usually in a menu like fashion. An example would be the ATM machine where the user is provided with the limited set of functions to select from to conduct his banking needs.

2.7 Utilities

Utilities software helps the operating system to maintain the computer system to operate in the most optimal state. Some examples of such utilities include the disk defragmenter, task manager, software and hardware monitor that comes with your Window 10 operating system.

Reference List

Silberschatz, A., Galvin, P. B., & Gagne, G. (2014). Operating system concepts.

Topic 03 – Computer Memory and Storage

Lesson Learning Outcomes

1. Understand the basic building block of storage
2. Distinguish between computer's memory and storage
3. Explain volatile and non-volatile storage
4. Identify the different levels of storage in a computer system
5. Discuss the storage characteristics of each level
6. Explain how the different levels of storage works together to support the CPU
7. Other types of memories

3.1 Basic building block of computer storage

In topic 1, we briefly looked at the data storage hierarchy. In this topic, we will examine in more detail the components in the data storage hierarchy. The basic building block of storage is a binary digit (bit). A bit represents an electronic circuit that is either open or close (two possible states). Information is stored in the computer using these electronic circuits. However, a bit by itself is insufficient to represent any meaningful information; therefore, 8 bits are combined to form a byte and can be used to represent the English character set (A,B,C… Z, a, b, c… z, 0, 1, 2… 9, +, -, *, / etc).

3.2 Distinguish between computer's memory and storage

In the context of this module, memory refers to short-term electronic storage area, i.e., registers, cache, RAM etc. Storage refers to long-term electronic storage area, i.e., mechanical hard disk or solid-state storage device. Each memory or storage location has an address for referencing.

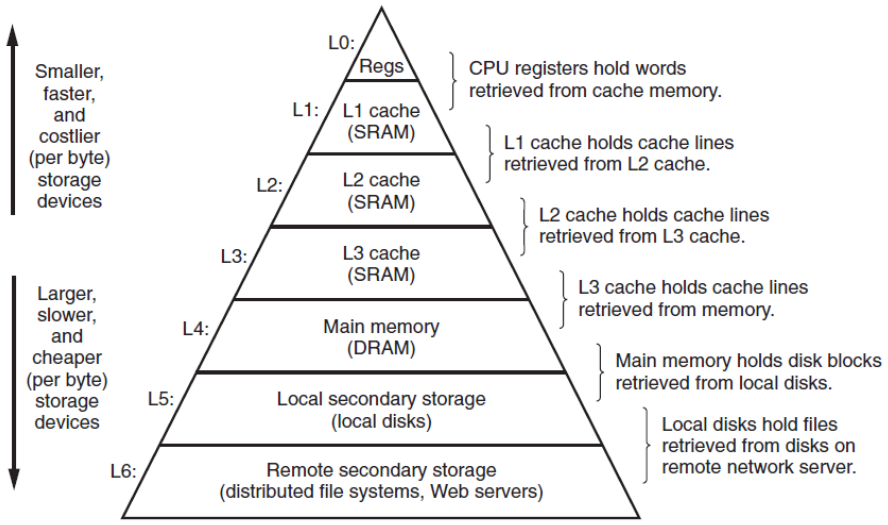
Volatile memory vs non-volatile storage

Memory is temporary and loses their contents when the computer is turn off. Their capacity is smaller; speed is faster and more expensive when compared with storage. Also known as volatile memory.

Storage is permanent and does not loses their contents even when the computer is turn off. Their capacity is bigger; speed is slower and cheaper when compared with memory. Also known as non-volatile storage.

The different levels of storage in a computer system and their characteristics

The central processing unit works at a very fast (electronic) speed; however, the hard disk is a mechanical device and is considered slow (rounds per minute or RPM). The disparity in speed does not optimise the working capability of the CPU. To resolve this disparity in speed, we need to implement some form of speed buffering to exploit the processing speed of the CPU. The speed buffering mechanism consists of layers of memory between the hard disk and the CPU.



3.3 Registers memory

Registers are volatile and are the fastest memory inside the CPU. There are a limited number of registers in the CPU and they are under the jurisdiction of the control unit. The instructions/data fetched by the control unit are loaded into the respective registers immediately before, during and after execution.

3.4 Cache memory

There is still some disparity in speed between RAM and the CPU. To compensate these speed latencies, another layer of memory is introduced – cache memory. Cache memory is very fast memory between RAM and the CPU. It stores frequently used instruction and make them immediately available to the CPU. A cache controller checks and maintain information of frequently used instructions and inform the CPU to fetch from the cache instead of RAM. In this manner, the overheads incurred in fetching these frequently used instructions/data from RAM are minimized, thereby improving the overall performance of the computer system.

At current, most CPU have three levels of cache memory Level 1 (L1), Level 2 (L2) and Level 3 (L3). While L1 and L2 are built into the CPU, level 3 is physically positioned between RAM and L1& L2 cache memories. The main objective of this structure is to allow the CPU to access the required instructions/data in the shortest possible time.

3.5 RAM memory

RAM (Random Access Memory) are volatile memory outside the CPU. Instructions fetched from the hard disk are first loaded into the RAM memory. It is like a staging area for all the programs being executed by the user. These programs must be first loaded into RAM before they can be executed. Please take note that each program is not fully loaded into RAM, only portions of the program are loaded into RAM each time. Availability of RAM can affect the performance of the computer, in general, the more RAM you have, the more programs you can load and run.

During the initial boot-up of the computer system, the supervisory part (kernel) of the operating system is loaded into RAM. When the user runs an application, the executable codes of the application are loaded into RAM as well.

There are two main categories of RAM: Static RAM (SRAM) and Dynamic RAM (DRAM). SRAM maintains the data stored in it only when power is supplied to it. On the other hand,

DRAM will lose the data unless it is constantly refreshed and as a result incurred higher overhead. As a result, SRAM is faster and more expensive. The higher SRAM speed warrants a higher position in the data storage hierarchy, whereas DRAM is typically associated with main memory.

How the different levels of storage work together to support the CPU

When the CPU needs to fetch an instruction, the CPU first look for the instruction in the L1 cache, the L2 cache, the L3 cache, the RAM, and the hard disk, respectively. The time it takes to fetch the instruction from external storage differs tremendously with those in memories. With the continuous improvement in CPU speed and the slow advancement in memory performance, this hierarchical structure of memory helps to bridge the gap in speed.

3.6 Other types of memory

ROM

ROM (Read Only Memory) are non-volatile memory. The ROM contains start-up instructions that are permanently stored in them and are the first instructions to be executed by the CPU. These instructions are etched into the circuitry at the factory and is known as Basic Input Output System (BIOS). The instructions in the ROM will also performs the Power-On-self-Test (POST) to ensure that all the configured hardware is correct and working. After the start-up is successfully completed, it will load the operating system kernel into RAM.

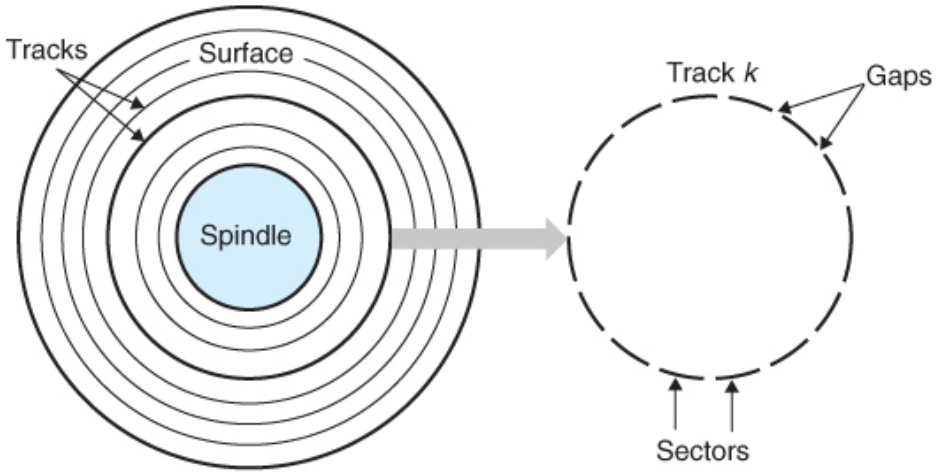
CMOS

CMOS (Complementary Metal Oxide Semiconductor) are volatile memory, but the contents are sustained with a battery. The BIOS uses the CMOS to store all the hardware configurations for the computer system. During the start-up, the BIOS will perform the POST and compared the result with the information stored in the CMOS. If the settings are correct, the BIOS will proceed to load the kernel of the operating system into RAM and ends its job.

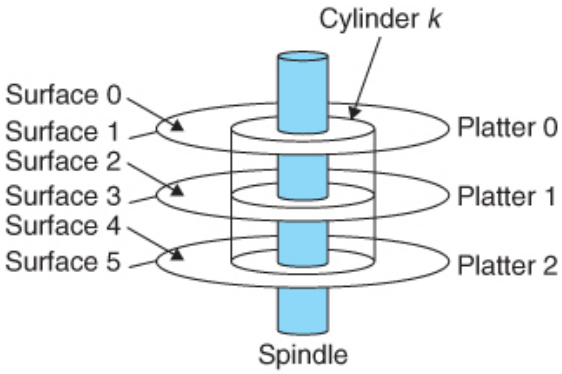
3.7 Data Storage

In contrast to the volatile nature of memories, data storage is non-volatile and typically used for permanent storage of data – typically associated with the hard disk. They have higher storage capacity, cheaper and are much slower than memories.

Hard disk consists of platters that exhibit magnetic properties mounted on a rotating spindle. The spindle is connected to a motor that rotates at a fixed rate, typically, 5400, 7200, 15000 RPM (Rounds per Minute). An actuator arm consisting of Read/Write heads float on the surface of the platters permits reading and writing of data. Please note that the aperture arm will only activate when the rotational speed reaches the desired speed. Each platter surface is organized into concentric rings (tracks) and each track is partitioned into sectors (typically 512 bytes in capacity. Each sector is further separated with a gap that contains control information that identify that sector.



(a) Single-platter view



(b) Multiple-platter view

3.8 Disk Capacity

The capacity of the hard disk is determined by the following factors: the maximum bits recordable into a 1-inch segment of a track (recording density), the maximum tracks per 1-inch segment relative to the centre of the platter (track density) resulting in the maximum recordable capacity for the hard disk (areal density).

The maximum capacity of a hard disk can be derived using the following formula:

$$\text{Capacity} = \frac{\#\text{ bytes}}{\text{sector}} \times \frac{\text{average }\#\text{ sectors}}{\text{track}} \times \frac{\#\text{ tracks}}{\text{surface}} \times \frac{\#\text{ surfaces}}{\text{platter}} \times \frac{\#\text{ platters}}{\text{disk}}$$

Disk Operation

As stated earlier, the read/write (r/w) heads are attached to an actuator arm. The actuator arm needs to move, search, and then read (and transfer) the data. These activities will consume overheads in performance – each activity takes time. The access time for a sector includes

seek time, rotational latency (delay), and transfer time. Seek time is the time it takes for the r/w head to reach a track; rotational latency refers to the time it takes to reach the right sector and the transfer relates to the time required to read or write the data. The rotational speed will also have an impact on the total access time. Assuming the average seek time is 9ms.

We can calculate the maximum rotation speed as follow:

$$T_{max\ rotation} = 1/RPM \times 60\ sec/1\ min$$

Assuming the rotational speed is at 7200 rpm, the average rotational speed is 1/7200 X 60 sec/1 min

$$T_{max\ rotation} = \frac{1\ minute}{7200\ rotations} \times \frac{60\ seconds}{1\ minute} = \frac{60\ seconds}{7200\ rotations} = 8.3ms\ per\ rotations$$

We can calculate the average rotational latency as follow:

$$Average\ rotational\ latency = ½ \times T_{max\ rotation}$$

$$Average\ rotational\ latency = ½ \times 8.3ms = 4ms$$

Assuming the average number of sectors/track is 400, we can calculate the average transfer time as follow:

$$T_{avg\ transfer} = 1/RPM \times 60\ sec/1\ min \times 1/(average\ number\ of\ sectors/track)$$

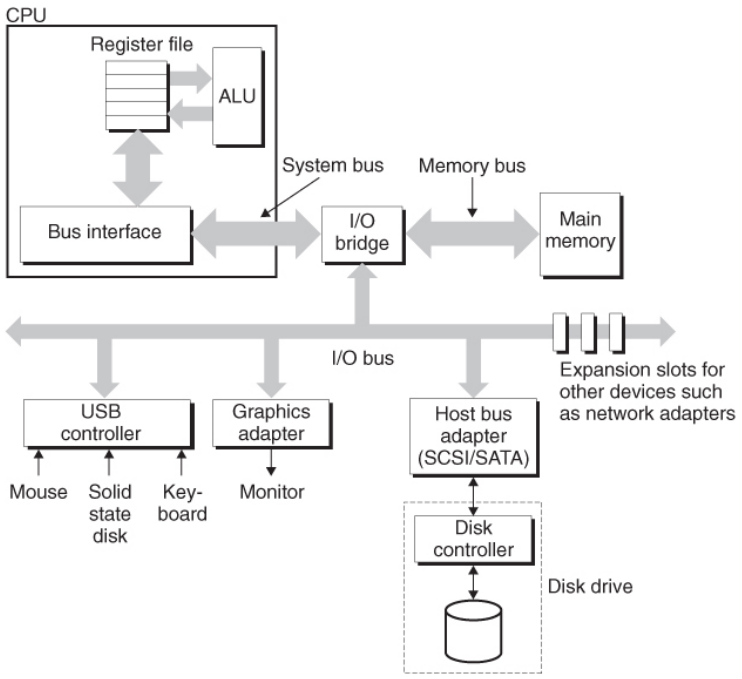
$$T_{avg\ transfer} = 8.3 \times 1/400 = 0.02ms$$

Total access time = average seek time + average rotational latency + average transfer

Total access time = 9ms + 4ms + 0.02ms = 13.02ms

3.9 Connecting Input/Output (I/O) devices

All peripheral devices are connected to the CPU via an I/O bus. A bus is the pathway that connects the peripheral devices to the CPU. Different third party devices can be connected to the I/O bus.



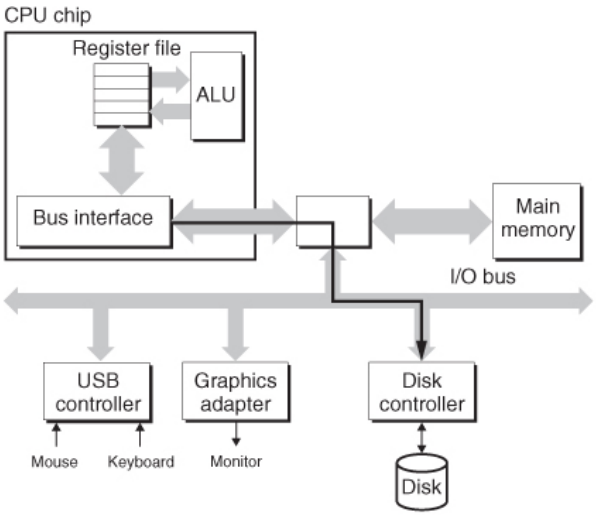
Universal Serial Bus USB: The USB controller is a conduit for devices attached to a USB bus.

A graphics card adapter performs graphics processing on behalf of the CPU.

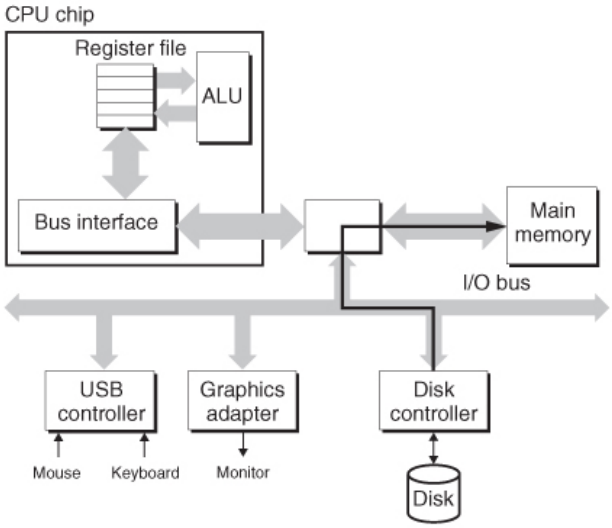
The SCSI/SATA host bus adaptor connects more than on1 disks to the I/O bus enabling it to support multiple hard disk drives.

Accessing Disks

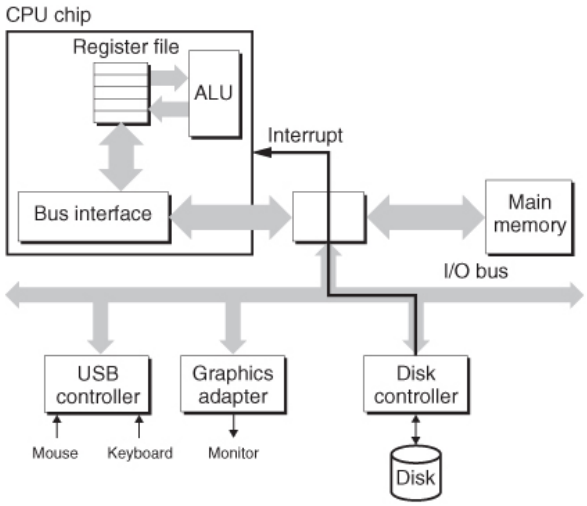
A detailed description of how I/O devices work, we have included a general overview of the steps that takes place when the CPU reads data from a disk.



(a) The CPU initiates a disk read by writing a command, logical block number, and destination memory address to the memory-mapped address associated with the disk.



(b) The disk controller reads the sector and performs a DMA transfer into main memory.



(c) When the DMA transfer is complete, the disk controller notifies the CPU with an interrupt.

References

Bryant, R. E., O'Hallaron, D. R., Manasa, S., & Tahiliani, M. P. (2016). Computer systems: A programmer's perspective.

Topic 04 – Digital Logic

Lesson Learning Outcomes

1. Count in denary, binary, octal and hexadecimal number systems.
2. Convert integers between denary, binary, octal and hexadecimal number systems.
3. Convert fractions between denary and binary number systems.
4. Perform addition and subtraction in binary system.
5. Explain the 2's complement, sign, and magnitude method of representing negative numbers.
6. Develop and use the truth tables for the AND, OR, XOR and NOT logic operators.
7. Write Boolean expression and draw corresponding logic circuit diagram and vice versa.
8. Use the laws of Boolean algebra to simplify Boolean expression.
9. Applications of logic gates in half-adder, full adder, decoder, multiplexer, and latches.

4.1 Introduction to number systems

The base (Radix) of a number system determine the number of symbols used in the system. The base of a number system is indicated by the subscript, for example:

- The denary number system 65_{10}
- The binary number system $0100\ 0001_2$
- The octal number system 101_8
- The hexadecimal number system 41_{16}

The Denary number system

The denary number system is also known as the 'decimal' system, has a radix of 10 and uses ten digits (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9) to represent all the numbers. It is the most common number system used to define a set of values to represent quantity.

The Binary number system

The binary system is used by computer system, has a radix of 2 and uses 2 digits (0 or 1) to represent some value. Each digit is known as a bit – 'binary digit'.

The Octal and Hexadecimal number system

The octal number system has a radix of 8 and uses 8 digits (0, 1, 2, 3, 4, 5, 6, 7) to represent value. The hexadecimal number system has a radix of 16 and uses 10 digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) and 5 characters (A - 10, B - 11, C - 12, D - 13, E - 14, F - 15) to represent value.

The positional number system - In a positional number system, the value of each digit is determined by the position of that digit.

1000	100	10	1	Decimal number position weighted value
10^3	10^2	10^1	10^0	Decimal number position
0	0	6	5	Decimal number pattern
MSN			LSN	Decimal number position significance MSN – Most Significant number, LSN – Least Significance number

Figure 4.1 positional value of each digit in decimal system

128	64	32	16	8	4	2	1	Bit position weighted value
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Bit position
0	1	0	0	0	0	0	1	Bit pattern
MSB							LSB	Bit position significance MSB – Most Significant Bit, LSB – Least Significance Bit

Figure 4.2 Positional value of each digit in binary system

512	64	8	1	Octal number position weighted value
8^3	8^2	8^1	8^0	Octal number position
0	1	0	1	Octal number pattern
MSN			LSN	Octal number position significance MSN – Most Significant number, LSN – Least Significance number

Figure 4.3 Positional value of each digit in octal system

4096	256	16	1	Hexa number position weighted value
16^3	16^2	16^1	16^0	Hexa number position
0	0	4	1	Hexa number pattern
MSN			LSN	Hexa number position significance MSN – Most Significant number, LSN – Least Significance number

Figure 4.3 Positional value of each digit in hexadecimal system

Number conversion

Converting Decimal to Binary: Convert 65_{10} to binary

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Bit position
128	64	32	16	8	4	2	1	Bit position weighted value
0	1	0	0	0	0	0	1	0100 0001 →The number in binary system
0	64	0	0	0	0	0	1	65 →The number in decimal system

Converting Decimal to Octal: Convert 65_{10} to octal

1. Convert the decimal number to binary first
2. Group the binary numbers into 3-bits section starting from the LSB to MSB
3. Add zero(s) to the left of the MSB so that all the groups have 3-bits
4. Write the 3-bit binary number to its octal value

Additional bit added so that all the groups have 3 bits		MSB				LSB			
Binary value	0	0	1	0	0	0	0	1	001 000 001_2
Octal value	1		0			1			101_8

Converting Decimal to Hexadecimal: Convert 65_{10} to hexadecimal

1. Convert the decimal number to binary first
2. Group the binary numbers into 4-bits section starting from the LSB to MSB
3. Write the 4-bit binary number to its hexadecimal value

	MSB						LSB	
Binary value	0	1	0	0	0	0	1	0100 0001_2
Hexadecimal value	4				1			41_{16}

Real number (fractions) conversion

Convert 0.6875_{10} to binary

$0.6875_{10} = 0.10110000_2$ (answer should be in 8-bit format)

				result	Integer part of the result	
0.6875	x	2	=	1.375	1	
0.375	x	2	=	0.75	0	
0.75	x	2	=	1.5	1	
0.5	x	2	=	1.0	1	
0	x	2	=	0	0	

Convert 0.6875_{10} to octal

1. Convert the fraction to binary first
2. Group the binary numbers into 3-bits section starting from the Left to right after the decimal point.
3. Add trailing zero(s) to the right to form groups have 3-bits if required
4. Write the 3-bit binary number to its octal value

Trailing 0 bit added so that all the groups have 3 bits	Decimal point									
Binary fraction value	.	1	0	1	1	0	0	0	0	0	0.10110000_2
Octal fraction value	.	5			4			0			0.540_8

Convert 0.6875_{10} to hexadecimal

1. Convert the fraction to binary first
2. Group the binary numbers into 4-bits section starting from the Left to Right after the decimal point.
3. Add trailing zero(s) to the right to form groups have 4-bits if required
4. Write the 4-bit binary number to its hexadecimal value

Trailing 0 bit added so that all the groups have 4 bits	Decimal point									
Binary fraction value	.	1	0	1	1	0	0	0	0	0.10110000_2
Hexadecimal fraction value	.	B				0				$0.B0_{16}$

Convert 0.1011_2 to decimal

Decimal point	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	Bit position
	0.5	0.25	0.125	0.0625	0.03125	..	..	..	Bit position weighted value
.	1	0	1	1	0	0	0	0	0.10110000_2
	0.5	0	0.125	0.0625	0	0	0	0	0.6875_{10}

Negative numbers

Sign and magnitude

The sign and magnitude method uses the most significant bit (MSB – bit 7) as a sign bit. Bit 7 has no positional value and is used to denote if the number is negative or positive. If the MSB is set to '1', it represents a negative number. If the MSB is set to '0', it represents a positive number. The range of numbers that can be represented using the sign and magnitude method is -127 to +127.

	B7 (MSB) Sign Bit	B6	B5	B4	B3	B2	B1	B0 (LSB)
+127	0	1	1	1	1	1	1	1
-127	1	1	1	1	1	1	1	1

Two's complement

The two's complement method is another way to represent negative. The most significant bit (B7) represents the value -128 and the rest of the bits represent +127. The range of numbers that can be represented using the two's complement method is -128 to +127.

	B7 (MSB)	B6	B5	B4	B3	B2	B1	B0 (LSB)
+127	0	1	1	1	1	1	1	1
-128	1	0	0	0	0	0	0	0

Binary Arithmetic

Addition

The Arithmetic logic Unit (ALU) performs addition of binary numbers. If there are more than 2 numbers to be added, it will add the first two and then add the carry over from the previous addition.

Positional value	128	64	32	16	8	4	2	1	
27_{10}	0	0	0	1	1	0	1	1	
155_{10}	1	0	0	1	1	0	1	1	
	0	0	1	1	0	1	1		Carry bit
182_{10}	1	0	1	1	0	1	1	0	

Subtraction

We can perform subtraction using the two's complement method by adding the numbers. Subtract 12_{10} from 26_{10}, i.e., 26_{10} - 12_{10}

The expression 26_{10} - 12_{10} can be re-written as 26_{10} + (- 12_{10})

	Last carry bit ignored	128	64	32	16	8	4	2	1	
26		0	0	0	1	1	0	1	0	
(-12)		1	1	1	1	0	1	0	0	Two's complement
	1	1	1	1						Carry bit
14		0	0	0	0	1	1	1	0	

4.2 Boolean algebra and synthesis of circuits

Boolean algebra can be used to represent logic circuits and logic circuits can be designed to achieve a specific purpose, i.e., decoder, accumulator in the CPU. There are THREE (3) basic operations: 'addition', 'complementation' and 'multiplication'.

The Laws for Addition				The Laws for Multiplication				The Laws for Complement ation	
A	B	A + B		A	B	A · B		A	A'
0	0	0		0	0	0		0	1
0	1	1		0	1	0		1	0
1	0	1		1	0	0			
1	1	1		1	1	1			

Laws of Boolean algebra

Idempotent	Complement	Associative	Commutative
A + A = A A · A = A	A + A' = 1 A · A' = 0	(A+B) + C = A+(B+C) (A · B) · C = A · (B · C)	A+B = B+A A·B = B·A
Distributive	De Morgan	Identity	Involution
A+B·C = (A+B)·(A+C) A·(B+C) = A·B + A·C	(A+B)' = A'·B' (A·B)' = A'+B'	A+0 = A A+1 = 1 A·1 = A A·0 = 0	(A')' = A

The laws as described above can be used to simplify Boolean expression. This is important as the Boolean expression can be used to represent logic circuits and through simplification can results in the reduction of logic components in digital circuits.

Simplification of Boolean algebra

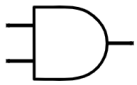
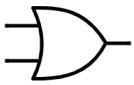
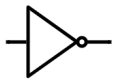
Simplify the following Boolean expression:

$$\boldsymbol{Q = A.B.\overline{C} + A.B.C + A.\overline{B}}$$

$A.B.\overline{C} + A.B.C + A.\overline{B}$	
$A.(B.\overline{C} + B.C + \overline{B})$	Distributive law: 'A' is common
$A.(B.(\overline{C} + C)) + \overline{B})$	Distributive law: 'B' is common
$A.(B.(1) + \overline{B})$	Complement law: $\overline{C} + C$ = 1
$A.(B + \overline{B})$	Complement law: $\overline{B} + B$ = 1
$A.1$	Identity law
A	

Logic gates

Logic gates are electronic devices for implementing Boolean operations to achieve a specific application purpose, i.e., a logic circuit.

AND gate	OR gate	XOR gate	NOT gate
			

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

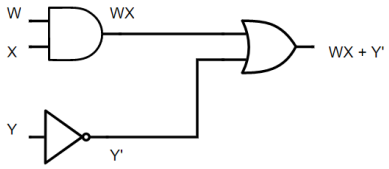
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

A	B
0	1
1	0

Deriving logic circuits from Boolean expression

The Boolean expression for a logic circuit can be marked by following the output of each logic gate as it connects to the input of another logic gate.

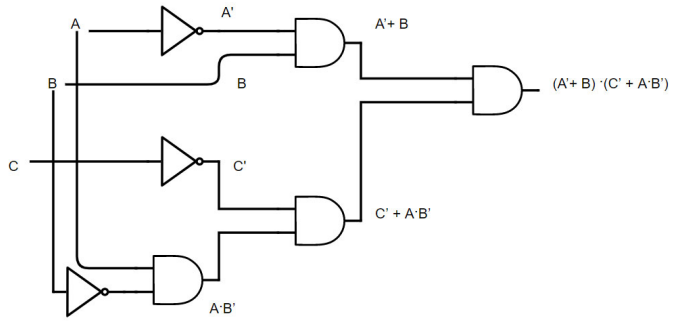
Example 1: W·X + Y'



W	X	Y	WX	Y'	WX+Y'
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Logic/Truth table for W·X + Y'

Example 2: (A' + B) · (C' + A·B')



A	B	C	A'	A'+B	C'	B'	AB'	C'+AB'	(A' + B) · (C' + A·B')
0	0	0	1	1	1	1	0	1	1
0	0	1	1	1	0	1	0	0	0
0	1	0	1	1	1	0	0	1	1
0	1	1	1	1	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0
1	0	1	0	0	0	1	1	1	0
1	1	0	0	1	1	0	0	1	1
1	1	1	0	1	0	0	0	0	0

Logic/Truth table for (A' + B) · (C' + A·B')

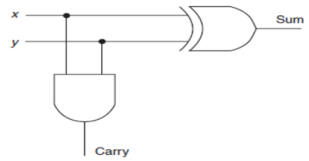
4.4 Applications of logic gates

The Half Adder

The arithmetic circuit of the Arithmetic Logic Unit (in the CPU) perform the addition of two binary digits, giving a sum bit and a carry bit is called half adder. The truth table of the half adder consists of two inputs X and Y and two out puts sum (S) and carry (C) is given below:

Input		Output	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder Logic Diagram

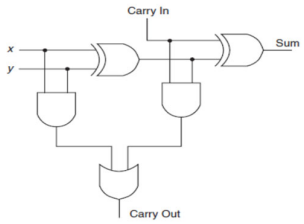


The Full adder

The Full Adder is a combinational circuit that performs the arithmetic sum of three input bits and generates a sum output and an output carry. The full adder is designed for multi-bit addition purposes.

Input			Output	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder Logic Diagram

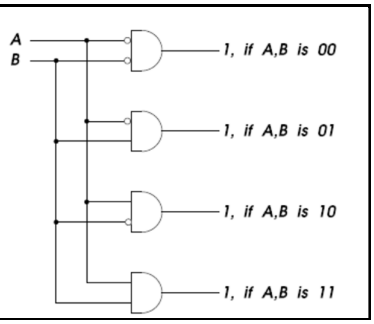


The Decoder

A decoder is a circuit that detects which bit pattern is presented at its input and outputting a '1' to enable the corresponding output. In general, a decoder has n inputs and 2^n outputs. The decoder circuit is used by the Control Unit of the CPU to select appropriate circuits based on the operation code.

A 2 to 4-bit decoder

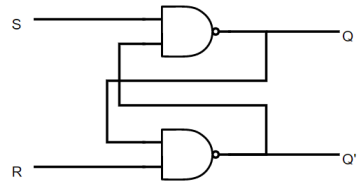
A	B	W0	W1	W2	W3
0	0	1			
0	1		1		
1	0			1	
1	1				1



The RS latch

The RS latch is the basic combinatorial circuit that can be used to build memory element.

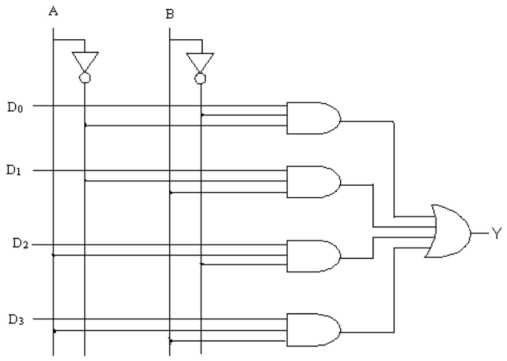
The RS latch				
S	R	Q	Q'	COMMENTS
0	0	1	1	INVALID
0	1	1	0	SET
1	0	0	1	RESET
1	1	NC	NC	No change



The multiplexer

The multiplexer circuit is a combinatorial circuit that has many inputs but only one output. A multiplexor is also called a data selector because the output depends on the input data bit that is selected.

The multiplexer		
A	B	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3



References

Mano, M. M., & Ciletti, M. D. (2006). Digital design. Addison Wesley Longman.

Topic 05 – Introduction to C Programming

Lesson Learning Outcomes

1. Write simple C programs
2. Use simple input and output statements
3. Use fundamental data types
4. Describe memory concepts
5. Describe and perform arithmetic in C
6. Decision Making: Equality and Relational Operators
7. Secure C Programming

Introduction to C programming languages

A program is a set of instructions that tells a computer how to perform tasks for the user. The C programming language provides a set of syntax to develop the program. The C programming language was evolved from another programming B by Dennis Ritchie at Bell Laboratories (1969 - 1973). The C programming language is widely known as the development language of the UNIX operating system.

5.1 A simple C program structure

```
#include <stdio.h>      // This is the C standard library that contains all the input
                        // and output routine files

int main()              // This is the main function
{
  /* My first C program */      // Everything between the /*...*/ will be ignored by the
                                // compiler. It is used mainly for including comments in
                                // the program

  printf("Welcome to FCS \n");  // Displays the  message "Welcome to FCS" on the
                                   screen.
  return 0;                     // The return statement signify the end of the main()
                                // function and returns control to the operating system.
}
```

5.2 Another simple C program: Adding Two Integers

```
#include <stdio.h>

int main()
{
  // ====================================
  // declare and initialise working storage
  // ====================================
     int num1 = 0;     // user input
     int num2 = 0;     // user input
     int sum = 0;      // holds the sum of the two integers

  // ================================
  // Prompts the user to enter two numbers
  // ================================
     printf("\n\nPlease enter the first number: ");
     scanf("%d", &num1);

     printf("\n\nPlease enter the second number: ");
     scanf("%d", &num2);

  // ============================
  // Compute the sum of the two numbers
  // ============================
     sum = num1 + num2;

  // ==============
  // Prints the sum
  // ==============
     printf("\n\nThe sum of %d and %d is %d", num1, num2, sum);

     return 0;
}
```

5.3 Basic I/O

Basic Input operation with scanf()

The scanf (scan formatted) function is used for getting user input from the keyboard. It will convert the input to the type specified in the format string.

For examples:

```
scanf(“%c”, &_nextchar);
scanf(“%f”, &_radius);
scanf(“%d %d”, &_length, &_height);
/* the & symbol refers to an address operator */
```

Basic Output operation with printf()

The printf() (print formatted) function is used to display output – monitor, printer etc. The function format requires a format string that defines:

- The text to print out
- The specifications on how to print out values of variables and the variables whose value is to be printed

  For examples:

```
printf("%d is an ODD number", 113);
printf("43 + 59 is %d\n", 43+59);
printf("_num1+ _num2 = %d\n", _num1 + _num2);
printf("%d+%d=%d\n", _num1, _num2, _num1 + _num2);
```

Some format specification conventions:

- %d – decimal
- %i – integer
- %x – hexadecimal
- %c – ASCII character
- %s – string of ASCII characters
- %f – floating point value
- \n – new line
- \t – tab

C basic language syntax

Comments

Comments are important and are used as in-text documentation in the C program for readability and understandability for maintenance purpose. They are ignored by the compiler. It provides readability and maintainability. Comments are enclosed with an open /* and ends with */ for multiple line comments and // for single line comment.

Identifiers (variables)

Identifier is a label used to identify a variable, a function, or any other user-defined item. An identifier must begin with a letter A to Z or a to z or an underscore _ followed by zero or more letters or digits (0 to 9). Please take note that C does not allow punctuation characters such as @, $, and % within identifiers. The C programming language is case sensitive and Kaplan and kaplan are two different identifiers, occupying two different memory locations in RAM.

Good naming convention for variable names is important for quality & maintenance purpose. Variable names should be meaningful to reflect the intention of use. When you join two words for a variable name, the first character of the second word should be upper-case. You should declare and initialise the variables before use.

Statement terminator

Each individual statement must end with a semicolon (statement terminator). It indicates the end of one logical entity. For example:

```
printf("Welcome to Kaplan! \n");
return 0;
```

Whitespace

Blanks, tabs, newline characters and comments are ignored by the C compilers. They are mainly used for readability purpose. For example:

```
total = qty_on_hand * unit_price;
```

Memory concepts

When we declare the variables int num1, we are telling the C compiler to reserve a memory location in RAM and label them as such. In the scanf() function, we used the & to denote the memory location of that variable. When the user enters a number, that number is stored at that memory location labelled as num1. The & (ampersand) is the address operator.

```
scanf("%d\n", &num1);
```

The C basic language syntax

The following are the C programming language reserved words and cannot be used for naming identifiers, function names etc. They represent special meaning to the C compiler.

Keywords				
auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typedef	
default	for	short	union	
Keywords added in C99 standard				
_Bool	_Complex	_Imaginary	inline	restrict
Keywords added in C11 standard				
_Alignas	_Alignof	_Atomic	_Generic	_Noreturn
_Static_assert	_Thread_local			

5.5 Basic Data types in the C programming language

Data type	printf conversion specification	scanf conversion specification
Floating-point types		
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
Integer types		
unsigned long long int	%llu	%llu
long long int	%lld	%lld
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short	%hu	%hu
short	%hd	%hd
char	%c	%c

5.6 Arithmetic Operators

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	f + 7
Subtraction	-	$p - c$	p - c
Multiplication	*	bm	b * m
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	x / y
Remainder	%	$r \bmod s$	r % s

5.7 Decision Making: Equality and Relational Operators

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Logical Operators

C operation	Arithmetic Operator	Meaning
AND	&&	Returns a 1 if BOTH conditions are true
OR	||	Returns a 1 if either or both conditions are true
NOT	!	The outcome is reversed

Assignment operators

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
+=	c += 7	c = c + 7	10 to c
-=	d -= 4	d = d - 4	1 to d
*=	e *= 5	e = e * 5	20 to e
/=	f /= 3	f = f / 3	2 to f
%=	g %= 9	g = g % 9	3 to g

Operators precedence

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the *innermost* pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
* / %	Multiplication Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated third. If there are several, they're evaluated left to right.
=	Assignment	Evaluated last.

5.8 Secure C programming

Secure programming is about writing software that are not susceptible to attacks from hackers. Hackers can take advantage of software bugs in your codes or the underlying programming language like C. Remember that the C programming language is used to write operating systems. It is a very powerful and efficient programming language, and the programmer is expected to know what he is writing. Any flaws in the design and writing of the program codes will be detrimental.

References

Bryant, R. E., O'Hallaron, D. R., Manasa, S., & Tahiliani, M. P. (2016). Computer systems: A programmer's perspective.

Topic 06 – Structured Program Development in C

Lesson Learning Outcomes

- Describe the basic problem-solving techniques
- Develop algorithms through the process of top-down, stepwise refinement
- Use the if selection statement
- Use the if…else selection statement
- Use the switch.. case statement

6.1 Algorithms

Given a problem statement, we must establish a finite step-by-step process for solving the problem – this is known as an algorithm. Programmers must develop this algorithm for solving the problem before coding starts, it depicts each specific task to be done and the order in which they are done. The algorithm can be expressed in a graphical form, i.e., flow charts or text form, i.e., pseudocode.

6.2 Pseudocode

Pseudocode or Structured Text or Structured English provides us a way to express our algorithm without having to worry about the syntax of the implementation language. Pseudocode is a combination of the English language and programming control structure that helps us to analyse and develop the algorithm, allowing us to focus on solving the problem.

6.3 Formulating Algorithms with Top-Down, Stepwise Refinement

Stepwise refinement is the progressive refinement in small steps of a program specification into a program, it is also known as top-down design. The notion of adopting the "top down stepwise refinement" steps help us to “think through” an algorithm before coding begins. It is a commonly used technique in problem solving. The steps are as follow:

1. Start with the problem statement
2. Decompose the problem into smaller parts
3. Take each "part", and break it further into more detailed steps
4. Repeat Step 1-3 until each part is specific enough to write as pseudocode.

For example: Write a C program to request the user to enter 2 numbers and compute the total of the 2 numbers.

- The problem statement: request the user to enter 2 numbers, compute and print the sum of the 2 numbers.
- Decompose the problem: request the user to enter 2 numbers, find the sum of the 2 numbers, and print the sum.
- Refinement: Declare and initialise working storage, prompt the user to enter 2 numbers, add the 2 numbers, display the sum of the 2 numbers.

Pseudocode:

- Begin
- Declare and initialise working storage
- Prompt the user to enter 2 numbers
- Compute the sum of the 2 numbers
- Display the sum
- End

Coding:

```
// This program prompts the user to enter 2 numbers
// and prints the sum of the 2 numbers

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int num1 = 0;        // variable to hold user input
   int num2 = 0;
   int sum = 0;         // variable to hold the sum

   // Prompts the user to enter 2 numbers
   printf("\nPlease enter the first number : ");
   scanf("%d", &num1);
   printf("\nPlease enter the second number : ");
   scanf("%d", &num2);

   // Compute the sum of the 2 numbers
   sum = num1 + num2;

   // Display the sum
   printf("\nThe sum of the 2 numbers is %d", sum);

   return 0;    // returns control to the calling program
}
```

6.4 Control Structure

In the design of algorithm, there are some steps that we want to repeat for a given condition or for a given condition, there may be different steps to consider and take. Control structure allows us to control the flow of the algorithm to solve the problem. The THREE (3) main control structures are:

- Sequence: Each step of the algorithm is executed sequentially.
- Selection: This is the conditional control structure. This is the point in your algorithm where you must decide to choose between two or more paths of execution. The choice is based on a specific condition. The specific condition determines the path to continue.
- Iteration: Sometimes, we need to execute several steps several times – this is known as iteration of repetition. Depending on the specific conditions, the iterations may not take place, executed for a fixed number of times, or executed indefinitely until some condition has been met.

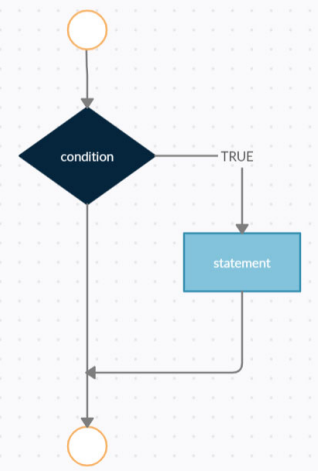
Conditional control structures in C

The C programming language supports three condition control structures: The if, if… else… and switch… case

6.5 The if statement

A boolean condition is evaluated and the statements inside the body of “if” execute if the given condition returns a true value.

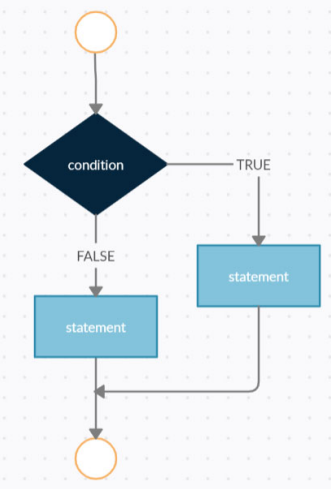
```
if (boolean expression)
  {
        Statement… // execute if true
  }
```



6.6 The if.. else statement

If the Boolean condition returns ‘true’ then the statements inside the ‘true’ block are executed. If the Boolean condition returns ‘false’ then the statements inside the ‘false’ block are executed.

```
if (boolean expression)
   {
        True statement… // execute if true
   }
else
   {
        False statement… // execute if false
   }
```



Compound statement

If you need to include several statements in the body of an if, you need to enclose the set of statements in braces { and }. A set of statements contained within a pair of braces is called a compound statement or a block.

Nested if statement

You can use an if.. else statement within another if.. else statement. When to use this nested structure depends on the problem statement.

```
if (boolean expression)
   {
      statement…
   }
else
   {
      if (boolean expression)
         {
            statement…
         }
      else
         {
            statement…
         }
   }
```

6.7 The switch.. case statement

The switch.. case statement allows us to execute selected block of codes from multiple alternatives. It is a more elegant if.. else statement structure.

```
switch (expression)
{
    case constant1:
       // statements
       break;

    case constant2:
       // statements
       break;
    .
    .
    .
    default:
       // default statements
}
```

When a break statement is reached, the switch structure terminates, and the execution continues after switch statement. It is not essential for each case to have a break statement, however if no break statement is included, the flow of control will continue to the next case until a break is reached or when the default is executed. The default case is optional and must appear at the end of the switch. Typically, the default case is used for performing a task when none of the cases is true. There is not a need to include the break statement in the default case.

Reference List

Bryant, R. E., O'Hallaron, D. R., Manasa, S., & Tahiliani, M. P. (2016). Computer systems: A programmer's perspective.

Topic 07 – C Program Control

Lesson Learning Outcomes

1. Describe the essentials of counter-controlled and sentinel-controlled iterations.
2. Use the for, while and do…while iteration statements to execute statements repeatedly.
3. Use logical operators to form complex conditional expressions in control statements.
4. Describe how to avoid the consequences of confusing the equality and assignment operators.

7.1 Iteration (repetitions) Essentials

Iteration (repetition) structures, or loops, are used when a program needs to execute some instructions several times until some condition is met. There are two main types of repetitions: counter-controlled and sentinel-controlled repetition.

7.2 Counter-controlled repetition

In counter-controlled repetition structure, the precise number of iterations is known in advance. A control variable is used to count the number of iterations and is typically incremented by 1 for each completed iteration. When the control variable indicates the correct number of iterations has been done, the loop will terminate, and execution control is passed to the next statement after the iteration statement.

The structure of the counter-controlled repetition:

1. The name of the control variable (or loop counter)
2. initial value of the control variable
3. increment (or decrement) by which the control variable is modified each iteration through the loop
4. condition that tests for the final value of the control variable

Example:

```
for (counter = 0; counter < 10; counter++)
    {
       statement...
    }
```

In the example, the for loop will execute 10 times. The counter is the control variable, it will increment by 1 (counter++) each time the loop executes. The counter is initially set to the value 0 (counter = 0). The condition (counter < 10) will be evaluated each time the loop executes. If the condition is true, the loop continues but if the condition evaluates to be false, the loop will terminate.

Sentinel-controlled repetitions

In sentinel-controlled repetition structure, we do not know in advance the number of times the loop will be executed. The sentinel value is a special input value that tests a condition within the repetition structure. If the condition evaluates to be true, the loop will exit. A sentinel value is simply a value that is used to terminate the loop. An example of when we would

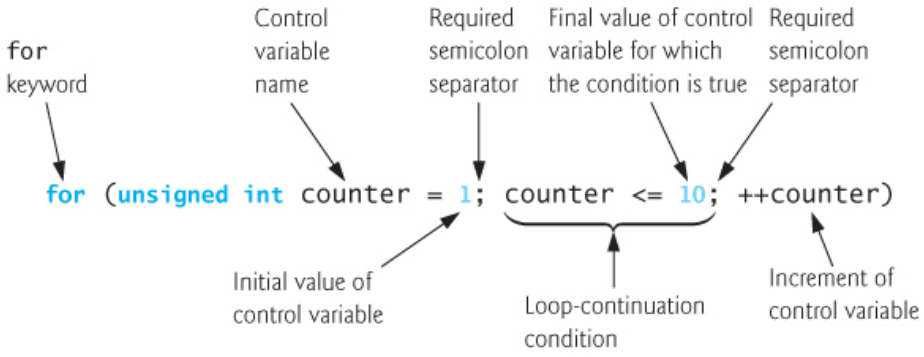
use sentinel-controlled repetition is when we wish to process the sales transaction file and we do not know in advance when we would reach the end of the file. For this example, the sentinel value would be the EOF (End-Of-File indicator).

Example:

```
counter = 2;
while (counter < 100)
   {
      counter = counter * 2
   }
```

7.3 The for() loop structure

The for() loop structure allows you to perform the loop a determined number of times (the condition in the form of a boolean expression). It accomplishes this by using a loop control variable to track the numbers of iterations. The control variable acts like a counter and you may specify the start and end values and the increment/decrement step for the counter. When the condition evaluates to False, the loop terminates.
The syntax for the for() loop structure is as follow:

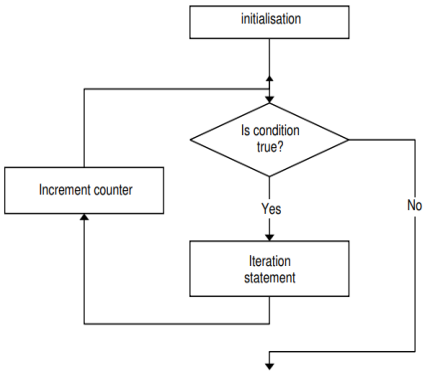



```
int main()
{
   // Declare and initialise working storage
   int counter = 0;

   // Prints the 2 times table
   printf("\nThe 2 times table");
   printf("\n=================\n");

   // The for() structure
   for (counter = 1; counter <= 12; counter++)
     {
        printf("\n%d X 2 = %d", counter, counter*2);
     }

   return 0;    // returns control to the calling program
}
```



Please take note that when the for() loop exit, the value of counter is 13.

7.4 The while() loop structure

The while() loop structure runs a block of statements if the condition specified in the while statement is True. The condition is a boolean expression, evaluating to either a True or False. The loop will continue if the condition remains True. A sentinel value may be used to exit the loop.

The syntax for the while() loop structure is as follow:

```
while (condition)
  {
    statement;
  }
```

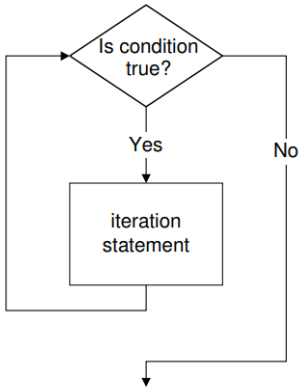
```
// This program demonstrates the while() loop structure

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int counter = 0;


   // The while() structure
   while (counter < 100)
     {
        printf("\nCounter value = %d", counter);
        counter = counter + 3
     }

   return 0;    // returns control to the calling program
}
```



7.5 The do… while loop structure

The do… while loop structure is similar to the while() loop structure except that it will execute the loop at least one time before checking the condition to exit the loop. The do… while loop test a condition at the end of the loop structure. If the condition is True, it will loop again but if the condition is False, it will exit the loop.
The syntax of the do… while loop structure is as follow:

```
do
  {
    statements;
  }  while (condition)
```

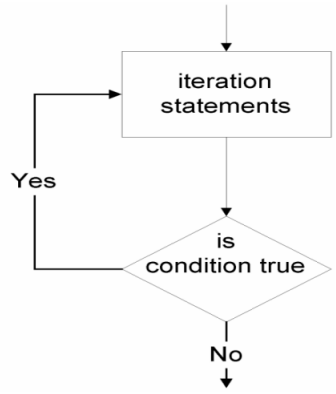
```
// This program demonstrates the do... while() loop structure

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int num = 0;


   // The do... while() structure
   do {
        printf("\nPlease enter a number (99 to quit!) : ");
        scanf("%d", &num);
   } while (num != 99);

   return 0;    // returns control to the calling program
}
```



7.6 Nested loop

You can include a loop structure within another loop structure, you have an outer and inner loop. For each of the outer loop pass, the inner loop completes the entire loop pass. For example:

```
// This program demonstrates a nested loop structure
// The program prints the 12x12 times table

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int inner = 0;       // this is the inner loop counter
   int outer = 0;       // this is the outer loop counter


   // The nested for() loop struture
   for (outer = 2; outer <= 12; outer++)
     {
        for (inner = 1; inner <= 12; inner++)
          {
             printf("\n%d X %d = %d", inner, outer, inner* outer);
          }
     }

   return 0;    // returns control to the calling program
}
```

7.7 Logical operators

Logical operators can be used to perform logical operations in a given Boolean expression. The C programming language has 3 logical operators: logical AND (&&), logical OR (||) and logical NOT (!).
The logical AND (&&) operator returns a True if both conditions evaluates to be True:

A = 10, B = 20, C = 30 then (A < B) && (C > A) evaluates to a True
A = 10, B = 20, C = 30 then (A < B) && (B == C) evaluates to a False

The logical OR (||) operator returns a True if either or both conditions evaluates to be True:

A = 6, B = 10 then ((A > 5) || (B > 50)) will evaluates to a True.
A = 1, B = 2 then ((A > 5) || (B > 50)) will evaluates to a True.

The logical NOT (!) reverse the state of the condition, i.e., if the state is True, the NOT operator will reverse the state to False.

A = 10, B = 10 then (A == B) evaluates to a True
A = 10, B = 10 then !(A == B) evaluates to a False

7.8 Confusing the Relational operator Equality (==) and the Assignment (=) operators.

A common error for novice programmer is the confusion between the relational operator == and the assignment operator =.

The assignment operator = assigned a value to a variable, i.e. counter = 10. In this case, the variable counter is assigned a value 10.

The relational operator == checks for equality, i.e., (num1 == num2). In this case, if num1 and num2 holds the same value, they are evaluated to be True.

Common loop errors

- The one-off error occurs when the '<= is' used instead of the '<' in the Boolean expression.
- Infinite loop occurs when the loop does not terminate because the condition is never met. For example:

```
// This program demonstrates an endless loop structure
// The loop will not terminate because we did not increment
// the counter value. The counter value remains the same
// through each loop and the while() condition will not
// not be false to exit the loop.
//
// To correct this problem, insert counter = counter + 1;
// within the while {}

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int counter = 0;     // this is the loop counter

   // The endless loop
   while (counter <= 10)
     {
        printf("This is the counter value = %d\n", counter);
     }

   return 0;    // returns control to the calling program
}
```

References

Bryant, R. E., O'Hallaron, D. R., Manasa, S., & Tahiliani, M. P. (2016). Computer systems: A programmer's perspective.

Topic 08 – C Functions Part I

Lesson Learning Outcomes

- Construct programs modularly from small pieces called functions
- Use common math functions in the C standard library
- Create new functions
- Use mechanisms that pass information between functions

8.1 Modularizing Programs in C

In the top-down stepwise refinement approach to problem solving, the problem statement is decomposed into smaller parts and each part can be addressed separately, giving rise to a modular structure. In C programming, we can write the smaller part using function.

8.2 Functions in C

Function is a set of instructions (C statements) to carry out a particular task. Functions are the basic building blocks in a C program. Functions can be re-used, thereby improving the quality of the program developed.

There are two types of functions in C – standard inbuilt functions and user-defined functions. C functions are invoked by a function call, which specifies the function name and includes information (as arguments) that the function needs to perform its designated task. The function may return a value after it finishes executing. When a program calls a function, program control is transferred to the called function. The called function will perform the defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program. To call a function you simply need to pass the required parameters along with function name and if function returns a value then you can store returned value.

8.3 Function Definitions - The function header

All standard inbuilt functions are declared in the header files, <filename.h>. We have already used some inbuilt functions so far, i.e., the printf() and scanf() functions in the <stdio.h> library header file. Other inbuilt library header files in C includes conio.h, string.h, stdlib.h, math.h, time.h, ctype.h. Every C program has at least one function – the main() function.

When the function is invoked, the C compiler will check to ensure the following conditions are correct:

- the number of arguments is correct,
- the arguments are of the correct type,
- the argument types are in the correct order, and
- the return type is consistent with the context in which the function is called.

The general syntax for defining a function is as follow:

```
return-type function-name (formal parameters list)
   {
      // function body
      // ...
      // return <value>
   }

Example:

int addTwoNumber(int first, int second)
   {
      return first + second;
   }
```

return-type:
A function may return a value. The return type specifies the data type of the value to be returned by the function.

function-name:
this is the user defined function name; it should follow the naming conventions for variable.

formal parameter list:
When the function is invoked, the data to be processed by the function is passed via the parameter list – they act like placeholder. The type, order and number of parameters is significant; however, the parameters are optional. Also known as actual parameter or argument.

function body:
It contains the C statements to be executed and define what the function will achieve.

Example of a function call:

```
// This program demonstrates a function that computes
// the sum of two numbers

#include <stdio.h>

int addTwoNumber(int first, int second)
   {
      return (first + second);
   }

int main()
{
   // Declare and initialise working storage
   int num1 = 3;
   int num2 = 5;
   int sum = 0; // holds the sum of the two numbers

   // invoke the function
   sum = addTwoNumber(num1, num2);

   printf("\nThe sum of %d and %d is %\n\n", num1, num2, sum);

   return 0;    // returns control to the calling program
}
```

In the example above, the return statement in the function “return (first + second)” will return the value of the expression to the calling function, i.e., the value 8. If the function header does not include the return-type, there is no need to include the ‘return’ statement in the function. Please take note that the main() function has a return-type of int. The “return 0” statement in the main() function is used to indicate whether the program has executed correctly to the operating system.

8.4 The math library functions

The <math.h> library contains all the mathematical related functions in the C programming language. A summary of the functions in the math library is included below for your reference.

Function name	Description	Example
double ceil (double x)	Returns the smallest integer value greater than or equal to x	ceil(7.3) is 8.0 ceil(-7.3) is -7.0
double floor (double x)	Returns the largest integer value less than or equal to x	floor(7.3) is 7.0 floor(-7.3) is -8.0
double fabs(double x)	Returns the absolute value of x	fabs(7.3) is 7.3 fabs(-7.3) is 7.3
double log(double x)	Returns the natural logarithm (base-e logarithm) of x	log10(1.0) is 0.0 log10(10.0) is 1.0 log10(10000) is 4.0
double fmod(double x, double y)	Returns the remainder of x divided by y	fmod(8.2, 3) is 2.20
double sqrt(double x)	Returns the square root of x	sqrt(225.0) is 15.00

double pow(double x, double y)	Returns x raised to the power of y (x^y)	pow(2, 8) is 256.0
double exp(double x)	Returns the value of e raised to the xth power	exp(1.0) is 2.71828 exp(2.0) is 7.389056
double cos(double x)	Returns the cosine of a radian angle x	cos(0.0) is 1.0 cos(1.0) is 1.543080
double tan(double x)	Returns the tangent of x	tan(0.0) is 0.0 tan(1.0) is 0.76159
double sin(double x)	Returns the sine of a radian angle x	sin(0.0) is 0.0
Double cbrt(double x)	Returns the cube root of x	crbt(27.0) is 3.0

Reference List

Bryant, R. E., O'Hallaron, D. R., Manasa, S., & Tahiliani, M. P. (2016). Computer systems: A programmer's perspective.

Topic 09 – C Functions Part II

Lesson Learning Outcomes

- Explain the purpose of function prototypes
- Explain how the function call/return mechanism is supported by the function call stack and stack frames
- Explain the role of header files in C
- Explain the different function call methods
- Explain the different storage classes

9.1 Function prototype

We need to declare the definition for a user-defined function before use. This is done through declaring the function prototype before the main() function. A function prototype is the declaration of a function that specifies the function's name, parameters and return type. The function prototype gives information to the compiler that the function may later be used in the program. Please take note that you do not need to declare the function prototype if the user-defined function is defined before the main() function. The compiler uses the function prototypes to validate function calls to prevent improper call to functions. If a function call does not match the function prototype, an error message will be generated.

9.2 Function call stack and stack frames

The functions that we invoked will need to use memory storage in RAM, these areas are called the stack to enable the functions to work properly. The computer uses the stack to pass function arguments, to store return information, to save registers for later restoration, and for local variables. The portion of the stack allocated for a single function call is called a stack frame. In other words, for each function call, new space (i.e., stack frame) is created on the stack.

9.3 Header Files in C

The Standard C Library provides a huge wealth of built-in functions and functionality that is available to you to include in your programs. All this functionality becomes available when you #include the appropriate header files. The #include <stdio.h> is a pre-processor command. It tells the C compiler to include (standard input output library) stdio.h file before compilation. The #include tells us that there is a filename ahead which must be included at the top of our program.

The Standard C Library provides a huge wealth of built-in functions and functionality and they are found in the appropriate header files. The header files contain the set of predefined standard library functions that we can include in our c programs. We must include the appropriate header files if we want to use these built-in functions. A header file has a .h extension that contains the C function declarations and macro definition. There are two kinds of header files: the files that we write and the files that come with your compiler. When we include a header file in a program, we copy the content of the header file.

Common header files in C:

Header file name	Description
stdio.h	Contains input and output functions to handle standard I/O operations, i.e., printf(), scanf() etc
assert.h	Contains diagnostics functions to help program debugging.
ctype.h	Contains the character handling functions that deals with character manipulation, i.e., converting lower-case character to upper-case character, checking if the character is a space or a digit etc. Take note that character is of ASCII type.
math.h	Contains the math functions that supports all the mathematical related functions in C language.
setjmp.h	Contains non-local jump functions that provide a way that deviates from standard call and return sequence. This is mainly used to implement exception handling.
signal.h	Contains signal handling functions that manages different conditions that may arise during program execution.
stdlib.h	Contains general utility functions, i.e., conversions of numbers to text and vice versa, memory allocations, random numbers etc.
string.h	Contains string processing functions for string manipulations, i.e., strlen(), strcpy() etc.
time.h	Contains date and time functions for manipulating time and date.
errno.h	Contains features for testing error codes, i.e., useful for reporting error conditions.
locale.h	Contains features to defines the location specific settings, such as date formats and currency symbols.

9.4 Passing arguments to function

When invoking a function, there are two ways that arguments can be passed to the function for processing: call by value and call by reference. The parameters defined in the function header are known as formal parameters and the parameters being passed to the function are known as arguments.

Call by value

This is the default method for passing arguments to a function. A copy of the actual value of an argument is passed into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

Call by reference

In this method, instead of passing a copy of the values, the memory address (reference) of the formal arguments is passed to the formal parameters. Inside the function, the address is used to access the actual argument used in the call. As the function will access the actual value at the memory address location, any changes made will be permanent, i.e., the actual value is changed permanently.

To pass a value by reference, the arguments are passed using the '&' operator, placed in front of the parameter name, i.e., &num1. The formal parameters receiving will used the '*' operator, which represent an address pointer that point to the memory location of the argument, i.e., *num1.

An example demonstrating the call by reference method. The function swap() will swap the contents of two numbers. In this example, &num1 and &num2 are passed to the function swap(). In the function swap() header, the formal parameters are declared as *num1 and *num2 respectively. *num1 can been seen as a pointer as it means that *num1 an address in memory rather than a regular value.

```
# include <stdio.h>

void swap(int *first, int *second)
{
   int temp;

   temp = *second;
   *second = *first;
   *first = temp;
}

int main()
{
    // declare and initialise working storage
    int num1 = 11;
    int num2 = 44;

    // invoke the swap function- passing the address of the arguments
    swap(&num1, &num2);

    printf("\n\nAfter Swapping\nnum1 = %d\nnum2 = %d\n", num1, num2);

   return 0;
}
```

9.5 Storage classes and scope rules

A storage class specifier can be used to modify the declaration of a variable, a function, and parameters. The storage specifier can be used to determine if a variable has internal, external, or no linkage, be stored in memory or in a register, receives the default initial value of 0 or an indeterminate default initial value, can be referenced throughout a program or only within the function, block, or source file where the variable is defined and that the storage duration for the object is maintained throughout program run time or only during the execution of the block where the object is defined

The auto storage class specifier

The auto storage class is the default for variables declared inside a block. A variable num1 that has automatic storage is deleted when the block in which num1 was declared exits. Since the auto storage class is the default, it is usually redundant in a data declaration.

Variables with the auto storage class specifier have automatic storage duration. Each time a block is entered, storage for auto variables defined in that block are made available. When the block is exited, the variables are no longer available for use.

The static storage class specifier

Variables declared with the static storage class specifier have static storage duration, which means that memory for these objects is allocated when the program begins running and is freed when the program terminates. Static storage duration for a variable is different from file or global scope: a variable can have static duration but local scope.

The extern storage class specifier

The extern storage class specifier lets you declare variables that several source files can use. An extern declaration allows the variable usable by the succeeding part of the current source file. All extern variables have static storage duration. Memory is allocated for extern objects before the main function starts executing and is released when the program terminates. The scope of the variable depends on the location of the declaration in the program text. If the declaration appears within a block, the variable has block scope; otherwise, it has file scope.

The register storage class specifier

The register storage class specifier instructs the compiler that the variable should be stored in a CPU register. It is useful for heavily used variables, such as a loop control variable, boosting performance by minimizing access time. However, due to the limited registers that a CPU have, this may not happen, and the variable is treated as having the storage class specifier auto.

Reference List

Bryant, R. E., O'Hallaron, D. R., Manasa, S., & Tahiliani, M. P. (2016). Computer systems: A programmer's perspective.

Topic 10 – Arrays

Lesson Learning Outcomes

- Use the array data structure to represent lists and tables of values
- Define an array, initialise an array and refer to individual elements of an array
- Define symbolic constants
- Pass arrays to functions
- Use arrays to store, sort and search lists and tables of values

10.1 Arrays in C

The C programming language provides a data structure – array that can store a fixed-size sequential collection of elements of the same data type. It is a collection of variables of the same type. Arrays are useful if we wish to store many instances of a variable. We can also access each instance through an array subscript (index). An array can be of data type as int, double, char etc. Another advantage of using an array is – to simplify the creation of algorithms to solve problems that would otherwise be complex and time-consuming.

10.2 Defining Arrays

We can declare an integer array as follow:

```
int testArray[10];
```

We can declare and initialise an array as follow:

```
int testArray[10] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
```

We can use an array subscript to access each element in the array:

```
testArray[0]        // refers to the first element of the array, i.e., the value 2
testArray[1]        // refers to the second element of the array, i.e., the value 4
```

We can assign a specific value to an array element by indicating the subscript:

```
testArray[0] = 50;      // we have assigned a value 50 to testArray[0]
```

10.3 Calculating the size of an array

We can calculate the size of an array by using the sizeof() function. The sizeof()function is a built-in function that is used to calculate the size (in bytes) of a data type RAM. We can use this function to compute the total elements in an array:

```
lengthOfArray = sizeof(testArray) / sizeof(testArray[0]);
```

Since testArray is declared as type int, the size of each element is 4 bytes. The total size of testArray can be computed as sizeof(testArray) = 40 bytes. The length of testArray can be computed through dividing the total size divided by the size of one element, i.e., 40/4 = 10 (elements).

10.4 Static and Dynamic arrays

Arrays can be either static or dynamic. Static arrays are those where the size of the array is defined by the software engineer, i.e., testArray[10]. The size is fix and cannot change during run time.

However, for dynamic array, the program will dynamically calculate and allocates memory for the array during run time. If you omit the size of the array, an array just big enough to hold the initialization is created. For example: int testArray[] = {10,12,14,12,14};, during run time, the testArray[] will be created with the size of 5 elements.

10.5 Array bound checking in C

Bounds checking is any method of detecting whether a variable is within some bounds before it is used. It is usually used to ensure that a number fits into a given type (range checking), or that a variable being used as an array index is within the bounds of the array (index checking). A failed bound check usually results in the generation of some sort of exception signal.
Because performing bounds checking during every usage is time-consuming, it is not always done. Bounds-checking elimination is a compiler optimization technique that eliminates unneeded bounds checking.

10.6 Working with arrays

1 dimensional arrays

It is common to use a for() loop to access all the elements in an array:

```
// This program demonstrates how to print all the array elements
// using a for() loop

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int testArray[10] = {1,3,5,7,9,11,13,15,17,19};
   int counter = 0;     // the loop counter

   // Print the testarray[] contents
   for (counter = 0; counter <= 10; counter++)
     {
        printf("\ntestarray[%d] = %d", counter, testarray[counter]);
     }

   return 0;    // returns control to the calling program
}
```

2 dimensional arrays

A 2-dimensional array is like a two one-dimensional array. A 2-dimensional array can be thought of as a 2-dimensional table that has x number of rows and y numbers of columns.
To declare a 2-dimensional array:

Type arrayName[row][column];

Each element in an array is identified by an element name if the form array[row][column], where the row index and the column index are the subscripts that uniquely identify each element in the array.

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

You can initialise a 2-dimensional array during declaration as follow:

```
int testArray[3][4] = {
                        {1, 2, 3, 4},
                        {5, 6, 7, 8},
                        {9,  10,  11,
12}
                      };
```

	Column 0	Column 1	Column 2	Column 3
Row 0	1	2	3	4
Row 1	5	6	7	8
Row 2	9	10	11	12

Working with two-dimensional array

You can use a nested for() loop to access all the elements of an array. The outer loop refers to the row and the inner loop refers to the column. The outer loop index and the inner loop index will together uniquely identify each element.

```
// This program demonstrates a 2-dimensional array using a nested
// for() loop to print the contents

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int testArray[3][4] = {{1,2,3,4},
                          {5,6,7,8},
                          {9,10,11,12}
                         };

   int outer = 0;       // the outer loop counter - the rows
   int inner = 0;       // the inner loop counter - the columns

   // Print the testarray[][] contents
   for (outer = 0; outer < 3; outer++)
     {
        for (inner = 0; inner < 4; inner++)
          {
             printf("\ntestArray[%d][%d] = %d", outer, inner, testArray[outer][inner]);
          }
     }

   return 0;    // returns control to the calling program
}
```

10.7 Using character arrays to store and manipulate Strings

A string is a sequence of characters of a given length. We can declare and initialise a character array as follow:

char zString[] = “This is a string stored as a character array”;

The character array zString has been initialised using a string literal - “This is a string stored as a character array”.

When we declare a string array that can holds 50 characters, xString[50] but we actually store only 20 characters. The string is a one-dimensional array of characters which is terminated by a null character '\0'. A null-terminated string contains the characters that comprise the string followed by a null. C functions that process string are designed to look for the null terminator and use it to determine the end of string.

Common string functions

strlen(s1) – this function calculates and return the length of a s1, the length does not include the '\0' null terminator.

strcpy(s1, s2) – this function copies string s2 into string s1, including the null terminator '\0'.

strcat(s1, s2) – this function adds s2 to the end of s1.

strcmp(s1, s2) – this function returns a number based on a numerical comparison of the two strings, the function returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

strchr(s1, ch) – this function returns a pointer to the first occurrence of character ch in string s1.

strstr(s1, s2) – this function returns a pointer to the first occurrence of string s2 in string s1.

Reference List

Bryant, R. E., O'Hallaron, D. R., Manasa, S., & Tahiliani, M. P. (2016). Computer systems: A programmer's perspective.

Kaplan City Campus @ Wilkie Edge | 8 Wilkie Road, Level 2, Singapore 228095

Kaplan City Campus @ PoMo | 1 Selegie Rd, Level 7, Singapore 188306

6733 1877 info.sg@kaplan.com

KaplanSingapore

Assessment Criteria Fundamentals of Computer Systems: Individual Assignment

Weightage : 80% (100 marks)
Submission Deadline : Session 14 (FT) / Session 7 (PT)

ASSESSMENT SUMMARY

Part A (60 marks):

Decimal number	Octal number	Hexadecimal number	Binary number
197	(a)	(b)	(c)
35	(a)	(b)	(c)
(g)	(f)	(e)	(d)

Table 1.

1. Please follow the instructions to complete Table 1.
    a. Convert the decimal number to octal.
    b. Convert the octal number to hexadecimal.
    c. Convert the hexadecimal number to binary.
    d. Add the two binary numbers.
    e. Convert the binary sum to hexadecimal.
    f. Convert the hexadecimal number to octal.
    g. Convert the Octal number to decimal.

(14 marks)

2.
    a. Convert the decimal fraction 0.453125 to binary.
    b. Convert the binary fraction 0.10000001 to decimal.
    c. Convert the octal fraction 0.370 to decimal.

(12 marks)

3. Simplify the following Boolean expression:
    a. (x + y) z' + y(x' + z')
    b. x'z (xy + y'z' + yz)
    c. y' (xz' + y'z)'
    d. (y + z')' (x' + y)'

(16 marks)

4. Using the truth table, prove the following expression:
    a. (x + y z)' + y' (x y + z) = x' y' +x' z' + y'z
    b. (yz' + x'z)' = xy' + xz + y'z'

(8 marks)

5. Construct a truth table for R(x,y,z) = x' y + x' z + y' z' and draw the corresponding logic circuits

(10 marks)

Part B (40 marks):

You need to develop a special calculator with the following features:

a. Add 2 number.
b. Divide 2 number.
c. Multiply 2 number.
d. Modulus of 2 numbers.
e. Convert Decimal to Binary
f. Convert Decimal to Octal
g. Convert Decimal to Hexadecimal

Instructions:

1. Produce the pseudocode for each of the features.
2. Write a complete C program that allows the user to choose the feature from a menu and display the result.
3. Each of the feature must be implemented using appropriate C function.
4. Produce test cases for each function that you have developed.

---End of Assessment---

Fundamentals of Computer Science				Total Marks	Marks awarded
Question 1 (a) - Question 1 (g) **Conversion of Numbers**	**0 - 4 marks**	**5 - 9 marks**	**10 - 14 marks**	14	
	No working shown, Only answers shown. Author does not detail the steps..	Steps for solving the problems are detailed but there are some problems or lack of detailed steps to the solution	Solution has detailed steps to reach the solution. Each step is necessary for the next step.		
Question 2(a) - Question 2(b) **Conversion of numbers**	**0-3 marks**	**4 - 6 marks**	**7 - 12 marks**	12	
	No working shown, Only answers shown. Author does not detail the steps..	Steps for solving the problems are detailed but there are some problems or lack of detailed steps to the solution	Solution has detailed steps to reach the solution. Each step is necessary for the next step.		
Question 3(a) - 3(d) **Simplify Boolean Expression**	**0 - 5 marks**	**6 - 8 marks**	**9 - 16 marks**	16	
	No working shown, Only answers shown. Author does not detail the steps..	Steps for solving the problems are detailed but there are some problems or lack of detailed steps to the solution	Solution has detailed steps to reach the solution. Each step is necessary for the next step.		
Question 4(b) - 4 (b) **Prove expression using truth table**	**0 - 3 marks**	**4 - 6 marks**	**7 - 8 marks**	8	
	No working shown, Only answers shown. Author does not detail the steps..	Steps for solving the problems are detailed but there are some problems or lack of detailed steps to the solution	Solution has detailed steps to reach the solution. Each step is necessary for the next step.		
Question 5 **Drawing Logic Circuit**	**0 - 4 marks**	**5 - 7 marks**	**8 - 10 marks**	10	
	No working shown, Only answers shown. Author does not detail the steps..	Steps for solving the problems are detailed but there are some problems or lack of detailed steps to the solution	Solution has detailed steps to reach the solution. Each step is necessary for the next step.		
Part B - (a) to (g) **- Produce the pseudocode for each of the features.** **- Write a complete C program that allows the user to choose the feature from a menu and display the result.** **- Each of the feature must be implemented using appropriate C function.** **- Produce test cases for each function that you have developed.**	**0 - 10 marks**	**11 - 25 marks**	**26 - 40 marks**	40	
	No working shown, Only answers shown. Author does not detail the steps..	Steps for solving the problems are detailed but there are some problems or lack of detailed steps to the solution	Solution has detailed steps to reach the solution. Each step is necessary for the next step.		
Formative comments on submission				100	
WEIGHTAGE				80%	

KAPLAN

Fundamentals of Computer Systems

Topic 01

Computer System Functional Components, Characteristics, Performance, Interactions and Use in IT

Learning Outcomes

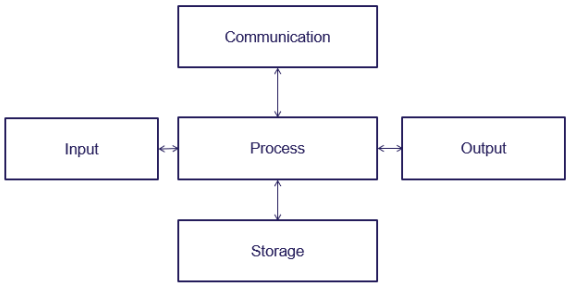
1. Identify the **FIVE (5)** basic operations of a computer
2. Explain the types of hardware associated with each operation
3. Explain how they work together to accomplish a user task
4. Explain the role of the Central Processing Unit (CPU)
5. Describe the components of the CPU
6. Explain how they work together in executing an instruction

2

KAPLAN

FIVE (5) BASIC OPERATIONS OF COMPUTER

- A computer system is an electronic device that accepts input from the user, processes it and display the results.
- All computer systems are comprised of 5 basic operations:



3

KAPLAN

INPUT OPERATION

- The input operation allows the user to communicate with the computer system.
- The computer process information in the form of 1s and 0s.
- The input operation translates the human language into a form that the computer can understand.
- For example, the keyboard , mouse, scanner, graphic tablet, bar code reader, touch screen etc.

4

KAPLAN

OUTPUT OPERATION

- The output operation communicates the result of the processing in a form that the user can understand.
- The processed information is in the form of 1s and 0s, is not ready for human consumption.
- For example, the display monitor, speaker, printer, multimedia projector, graphics card etc.
- Some hardware devices have combined both the input and output operations.
  - Can you suggest an example?

5

KAPLAN

PROCESS OPERATION

- The process operation performs basic arithmetic operations and logical comparison on the user's data.
- This is accomplished by the Central Process Unit (CPU) or microprocessor.
- Can you suggest some examples of popular microprocessor used in desktop, notebook and mobile devices?

6

KAPLAN

STORAGE OPERATION

- The storage operation allows data and instructions to be stored before, during and after processing.
- Data and instructions entered by the user are stored temporarily in storage (memory) before being passed to the CPU for processing and the output operation.
- Can you give some examples of storage devices?

7

KAPLAN

STORAGE OPERATION

- There are TWO (2) types of storage: Primary and Secondary
- Primary memory:
  -Temporary, fast and volatile (RAM, Cache, Registers, etc.)
- Secondary storage:
  -Permanent, slower and non-volatile (ROM, Hard disk, USB stick, etc.)

8

KAPLAN

COMMUNICATION OPERATION

- The communication operation allows the user to transmit and receive data and/or instructions from other computers or devices on the network.
- Examples of communication devices includes the network interface card (NIC) and modem on your computer.

KAPLAN

PUTTING THEM TOGETHER

OPERATION	TASK
INPUT OPERATION	You created a document.
PROCESS OPERATION	You issued commands to change the fonts, paragraphs, insert pictures, etc.
OUTPUT OPERATION	You are looking at the display screen while editing your document.
STORAGE OPERATION	You saved a copy of the document in your hard disk.
COMMUNICATION OPERATION	You emailed a copy of the document to your friend.

KAPLAN

CENTRAL PROCESSING UNIT (CPU)

- The most important component of the computer system
  - Responsible for fetching, decoding, and executing the CPU instructions.
- These instructions are built into the CPU and are collectively known as the instruction set.
  - Do you think that the instruction set is the same for all types of CPU? Why?
- The main components of the CPU
  - The Control Unit (CU), the Arithmetic Logic Unit (ALU), Registers and Cache memories.

11

KAPLAN

COMPONENTS OF CPU

- The Control Unit (CU):
  - The CU controls all the activities and coordinates the flow of data, instructions, and commands, inside the CPU.
  - It fetches the instruction from RAM (where the instructions are previously loaded from the hard disk).
  - It checks that it is valid for execution, and loads them into the appropriate register before passing the control to the ALU for execution.

12

KAPLAN

COMPONENTS OF CPU

- The Arithmetic login Unit (ALU):
  - The ALU carries out arithmetic and logical comparisons on the data according to the instruction.
    - Arithmetic operations include the basic operations like, addition, subtraction, multiplication, and division.
    - Logical operations include comparing data and deciding the next set of instructions to execute.

13

KAPLAN

COMPONENTS OF CPU

- Registers memory:
  - Before, during and after execution, the data, instructions, and results are stored in the registers within the CPU.
    - The registers are very high-speed volatile memory areas in the CPU.
    - They temporarily store the data, instructions, and result of the processing.
    - There are limited registers in the CPU and thus the contents of the registers are frequently swapped out to RAM to make way for other instructions and data.

14

KAPLAN

COMPONENTS OF CPU

- Cache memory:
  - Cache memories are also very fast memory areas inside the CPU.
  - They are used for speed optimization and they are positioned between the registers and RAM memories.
  - Frequently used data and instructions are stored in the cache instead of RAM which is slower than cache.

15

KAPLAN

CPU – PUTTING ALL COMPONENTS TOGETHER

Now that we have a good understanding of the components that makes up the CPU

How does all the components in the CPU work together in executing an instruction?

It is useful to discuss using the fetch-execute cycle. The fetch-execute consists of **FOUR (4) stages, Fetch, Decode, Execute** and **Store**.

16

KAPLAN

FETCH-EXECUTE CYCLE

- Fetch stage:
  - The control unit fetches the next information to be executed from Cache or RAM memory.
- Decode stage:
  - The control unit checks if the fetched information is data, instruction, or a memory address.
    - If it is data, it will put it into the data registers.
    - If it is an address, it will fetch the information from the memory location repeat the check again.
    - If it is an instruction, it will check again if it is a valid instruction.
      - ≈ If it is an invalid instruction, the CU will stop execution and display an error message.
      - ≈ If it is a valid instruction, the CU will load it into the instruction register.

17

KAPLAN

FETCH-EXECUTE CYCLE (con'td)

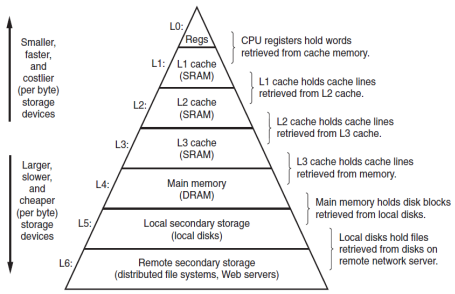
- Execute stage:
  - The CU will inform the ALU to execute the instruction. The ALU will execute the instruction on the data.
- Store stage:
  - The result of the execution will be stored in the appropriate register
- The fetch-execute cycle will repeat until the user terminate the program or shut down the computer.

18

KAPLAN

DATA STORAGE HIERARCHY

- Due to the disparity in speed, the memories and storage are arranged in hierarchy.
- Helps us to visualize how we can compensate for the differences in speed.



19

KAPLAN

END

20

KAPLAN

KAPLAN

Fundamentals of Computer Systems

Topic 02
Operating systems

Learning Outcomes

1. Identify and explain the TWO (2) basic kinds of software
2. Explain the relationships between the user, software, and hardware
3. Explain the role of device drivers
4. Explain the role of software utilities
5. Explain the role operating systems
6. Explain the FIVE (5) basic functions of an operating systems

OPERATING SYSTEM MANAGES HARDWARE

- Software are computer instructions that tells the CPU what to do.
- These instructions are known as a program.
- There are many programs running in the computer system and collectively they are known as software.

KAPLAN

TYPES OF SOFTWARE

- Two main types of software:
  - Application software helps the user to perform a variety of tasks.
    - Can you give an example of application software?
  - Systems software helps the computer to manage the hardware of the computer and interacts with the application software to complete a specific user's task.
    - Can you give an example of system software?

KAPLAN

SYSTEM SOFTWARE

- **Device Driver**
  - When you connect a new printer to your computer, you cannot send print jobs to the printer initially. You need to install the printer's driver program first before you can print.
  - Each peripheral device needs a device driver. They help the computer communicates with that device.
    - Can you give other examples of device drivers?

5

KAPLAN

SYSTEM SOFTWARE

- **Utility software**
  - Helps the operating systems to maintain the computer system to perform optimally.
  - Many utilities come with the operating system and you may buy some of these utilities as well.
    - Can you give some examples of these utilities?

6

KAPLAN

SYSTEM SOFTWARE

- **Operating systems**
  - The operating system is responsible for all the computer operations and manages all the computer resources.
  - Application programs must request the service of the operating systems to complete their tasks.
    - Can you give some examples of popular desktop and module operating systems?

7

KAPLAN

BASIC FUNCTIONS OF OPERATING SYSTEM

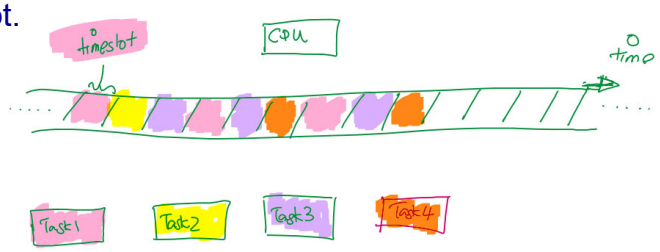
- All operating systems perform **FIVE (5)** basic functions:
  - ✓ Task management
  - ✓ File management
  - ✓ Memory management
  - ✓ User Interface
  - ✓ Utilities

8

KAPLAN

TASK MANAGEMENT

- Task management allows you to run more than one tasks at a time (multi-tasking).
- The CPU time is divided into timeslots and assigned to the tasks.
- The CPU can multiplex all the different tasks and execute each at a time for a given time slot.



9

KAPLAN

THREADS AND MULTI-THREADING

- A process is a program that performs a task at a time
  - A single thread of execution
- Modern CPUs has multiple cores and can multiplex and execute multiple threads simultaneously
  - For example, a web browser may be implemented as a process with several threads – one for retrieving an image, another for fetching information from the Internet, another for processing the image etc.
  - The threads can be from different programs – multiprogramming

10

KAPLAN

TIME SHARING

- An operating system can also allow many users to share its resources
  - Known as multi-users operating system
- Each user is allocated a time slice from the CPU and they can perform any processing tasks within it
- Allows many users to be connected to the CPU at the same time.

11

KAPLAN

MEMORY MANAGEMENT – VIRTUAL MEMORY (VM)

- Memory management manages available RAM memory and storage.
- They keep track of where the programs and data are residing in memory.
- Please take note that an application must first be loaded into memory (RAM) before they can be executed.
- Bigger programs have problem loading into available RAM.

12

KAPLAN

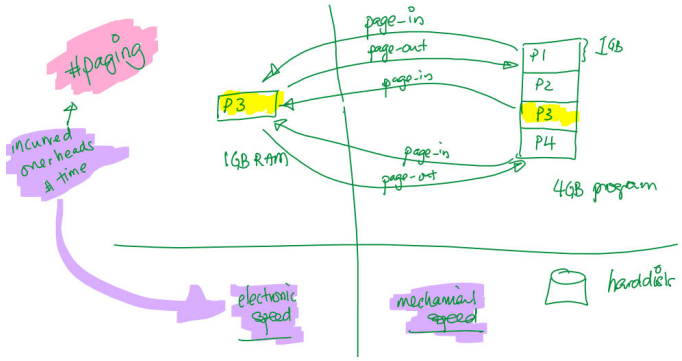
MEMORY MANAGEMENT – VIRTUAL MEMORY (VM)

- VM helps to address this problem by:
  - Reserving spaces on the hard disk to supplement available RAM
  - Allocate portion of the reserved spaces as working storage for RAM.
- Programs are subdivided into smaller sections (called pages) and brought into RAM when needed, while the rest will reside in the working storage area.

13

KAPLAN

MEMORY MANAGEMENT – VIRTUAL MEMORY (VM)



14

KAPLAN

FILE MANAGEMENT

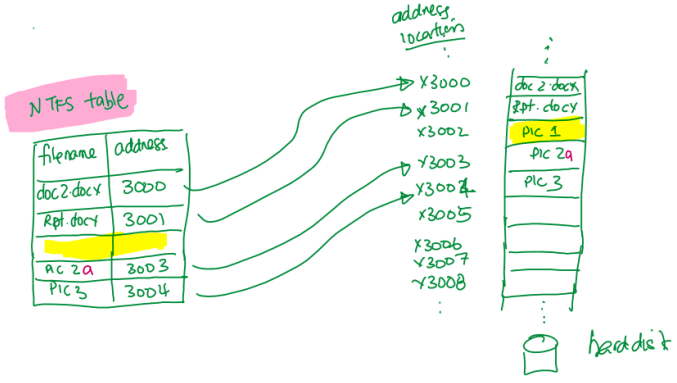
- This function manages the creation, deletion and access controls of files and data in the storage devices.
- It maintains a special system table (FAT or NTFS) that tracks all files in the storage.
- The system table maintains the file name and the physical address where the file is written into when you create that file.
- Each entry forms a map for all the files and data stored in that storage device.

15

KAPLAN

FILE MANAGEMENT

- When a file is deleted, the entry for the deleted file is removed from the system table.



16

KAPLAN

USER INTERFACE

- The user interface allows the user to communicate with the computer.
- It provides a means for the user to articulate his intention to the computer system.
- There are different types of user interface, the most common being:
  - Graphical user interface (GUI)
  - Command line interface
  - Menu driven interface

17

KAPLAN

UTILITIES

- These system software helps the operating system to maintain the computer system to operate in the most optimal state.
- Typically manages the routine and repetitive tasks.
- Can you give some examples of utility software?

18

KAPLAN

END

KAPLAN

Fundamentals of Computer Systems

Topic 03
Memory and storage

Learning Outcomes

1. Understand the basic building block of storage
2. Distinguish between computer's memory and storage
3. Explain volatile and non-volatile storage
4. Identify the different levels of storage in a computer system
5. Discuss the storage characteristics of each level
6. Explain how the different levels of storage works together to support the CPU
7. Other types of memories

STORAGE TECHNOLOGIES – BASIC BUILDING BLOCK OF COMPUTER STORAGE

- The basic building block of storage is a binary digit (bit).
- A bit represents an electronic circuit that is either open or close (two possible states).
- Eight bits are combined to form a byte and can be used to represent the English character set (A,B,C… Z, a, b, c… z, 0, 1, 2… 9, +, -, *, / etc).

3

KAPLAN

DISTINGUISH BETWEEN COMPUTER'S MEMORY AND STORAGE

- Memory refers to short-term electronic storage area, i.e., registers, cache, RAM etc.
- Storage refers to long-term electronic storage area, i.e., mechanical hard disk or solid-state storage device.
- Each memory or storage location has an address for referencing.

4

KAPLAN

VOLATILE MEMORY vs. NON-VOLATILE STORAGE

- Characteristics of Memory (volatile memory)
  - Temporary and loses their contents when the computer is turn off.
  - Smaller capacity,
  - Higher speed
  - More expensive
  - More expensive when compared with storage. Also known as volatile memory.

5

KAPLAN

VOLATILE MEMORY vs. NON-VOLATILE STORAGE

- Characteristics of Storage (non-volatile storage)
  - Permanent and does not loses their contents even when the computer is turn off.
  - Higher capacity
  - Slower speed
  - Cheaper when compared with memory.
  - Also known as non-volatile storage.

6

KAPLAN

DIFFERENT LEVELS OF STORAGE IN COMPUTER SYSTEM AND CHARACTERISTICS

- Speed buffering:
  - The central processing unit works at a very fast (electronic) speed
  - The hard disk is a mechanical device and is considered slow (rounds per minute or RPM).
  - The disparity in speed does not optimise the working capability of the CPU.

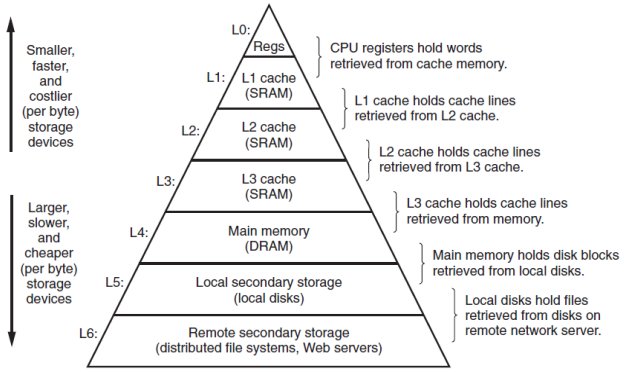
KAPLAN

DIFFERENT LEVELS OF STORAGE IN COMPUTER SYSTEM AND CHARACTERISTICS

- Speed buffering a method to resolve this disparity in speed
- The speed buffering mechanism consists of layers of memory between the hard disk and the CPU.

KAPLAN

DIFFERENT LEVELS OF STORAGE IN COMPUTER SYSTEM AND CHARACTERISTICS



9

KAPLAN

REGISTERS MEMORY

- Registers are volatile by nature and are the fastest memory inside the CPU.
- There are a limited number of registers in the CPU.
- CPU registers holds instructions/data fetched by the control unit
- Please take note that the instructions/data are loaded into the respective registers immediately before, during and after execution.

10

KAPLAN

RAM MEMORY

- The most common type of memory
- Store instructions and data waiting to be executed shortly.
- Volatile by nature and is not part of the CPU.
- Instructions and data first loaded into the RAM memory from hard disk.
  - Practically is a staging area for all the programs being executed
- Please take note that each program is not fully loaded into RAM, only portions of the program are loaded into RAM each time.

11

KAPLAN

RAM MEMORY

- In general, the more RAM you have, the more programs you can load and run.
- During the initial boot-up of the computer system, the supervisory part (kernel) of the operating system is loaded into RAM.
- When the user runs an application, the executable codes of the application are loaded into RAM as well.

12

KAPLAN

RAM MEMORY

- Two basic types of RAM
  - Static RAM (SRAM)
    - Maintains data when power is supplied
    - Faster and more expensive
  - Dynamic RAM (DRAM)
    - Needs to be refreshed frequently even when power is supplied
    - Higher overhead due to refresh
    - Slow (due to refresh) and cheaper

13

KAPLAN

CACHE MEMORY

- There is great disparity in speed between the RAM and the CPU.
- To address these speed latencies, another layer of memory is added – cache memory (between RAM and CPU).
- Cache memory is used to store frequently used instruction.
- A cache controller manages these frequently used instructions
  - Inform the CPU to fetch from the cache instead of RAM.
- Improving the overall performance of the computer system.

14

KAPLAN

CACHE MEMORY

- Most modern CPU have three levels of cache memory Level 1 (L1), Level 2 (L2) and Level 3 (L3).
- L1 and L2 are built into the CPU, level 3 is physically positioned between RAM and L1& L2 cache memories.
- The main purpose is to allow the CPU to access the required instructions/data in the shortest possible time.

15

KAPLAN

DIFFERENT LEVELS OF STORAGE WORK TOGETHER TO SUPPORT CPU

- CPU needs to fetch an instruction.
- The CPU first look for the instruction in the L1 cache, the L2 cache, the L3 cache, the RAM, and the hard disk, respectively.

16

KAPLAN

OTHER TYPES OF MEMAORY - ROM

- ROM (Read Only Memory) are non-volatile memory.
  - Contains computer start-up instructions and are the first instructions to be executed by the CPU.
  - Also known as Basic Input Output System (BIOS).
  - The ROM will also perform the Power-On-self-Test (POST).
    - Check that the hardware configurations are fine
  - The operating system kernel will then be loaded into RAM.

17

KAPLAN

OTHER TYPES OF MEMORY - CMOS

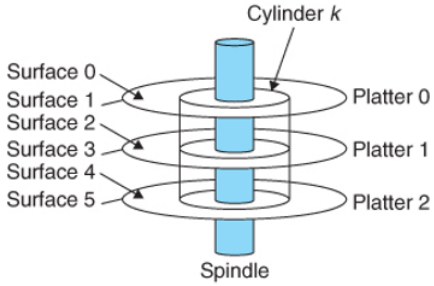
- CMOS (Complementary Metal Oxide Semiconductor) contains the hardware configurations and are volatile memory
  - A small battery maintains the stored information.
- The result of the POST is validated with this information.
- The BIOS will then proceed to load the kernel of the operating system into RAM and passes control to the kernel.

18

KAPLAN

DATA STORAGE

- Data Storage are used for permanent storage, i.e., hard disk.
- The hard disk consists of platters mounted on a rotating spindle.
- The rotating spindle is enabled by a motor (5400 RPM, 7200 RPM etc)
- An actuator arm consisting the read/write heads float on each platter (not shown in diagram)

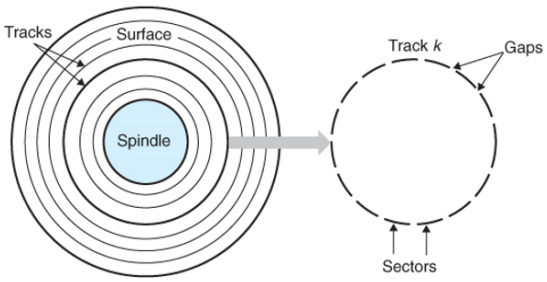


19

KAPLAN

DATA STORAGE

- Each platter surface is organized into concentric rings known as track.
- Each track is partitioned into sectors (512 bytes each).
- Each sector is separated by a gap that contains identification information for the gap.



20

KAPLAN

CALCULATING DISK CAPACITY

- The maximum capacity of a hard disk can be derived using the following formula:

$$\text{Capacity} = \frac{\#\text{ bytes}}{\text{sector}} \times \frac{\text{average }\#\text{ sectors}}{\text{track}} \times \frac{\#\text{ tracks}}{\text{surface}} \times \frac{\#\text{ surfaces}}{\text{platter}} \times \frac{\#\text{ platters}}{\text{disk}}$$

21

KAPLAN

DISK OPERATIONS

- The read/write heads attached to the actuator takes time to move, seek, read and transfer the data → total access time
- Total access time = average seek time + average rotational latency time + average transfer time
  - Calculate the maximum rotational speed : 1/RPM X 60 secs/1 min
  - Calculate the average rotational latency: ½ X maximum rotational speed
  - Calculate the average transfer time: 1/RPM X 60 X 1/(average number of sectors/track)

22

KAPLAN

DISK OPERATIONS

- Given the maximum rotational speed is 7200rpm, average seek time is 9 ms, average number of sectors/track is 400, calculate the total access time

$$T_{max\ rotation} = \frac{1\ minute}{7200\ rotations} \times \frac{60\ seconds}{1\ minute} = \frac{60\ seconds}{7200\ rotations} = 8.3ms\ per\ rotations$$

Average rotational latency = ½ X 8.3ms = 4ms

$T_{avg\ transfer}$ = 1/RPM X 60 sec/1 min X 1/(average number of sectors/track)

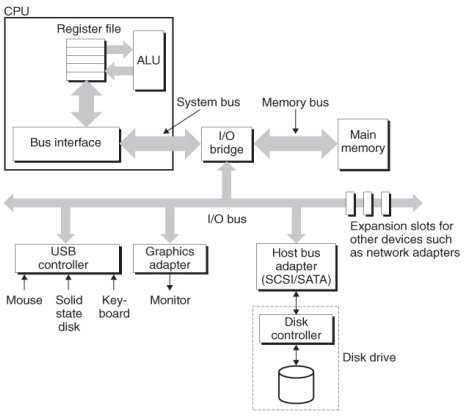
$T_{avg\ transfer}$ = 8.3 X 1/400 = 0.02ms

Total access time = 9ms + 4ms + 0.02ms = 13.02ms

KAPLAN

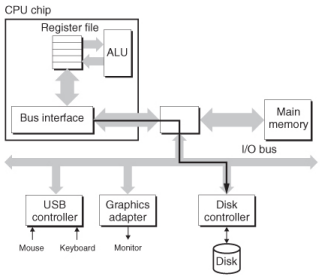
CONNECTING INPUT/OUTPUT (I/O) DEVICES

- Peripheral devices are connected to the CPU via an I/O bus.

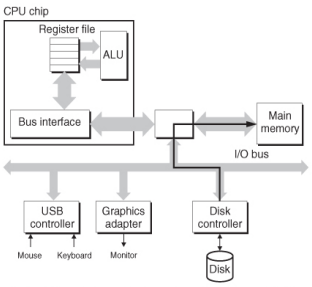


KAPLAN

ACCESSING DISKS



(a) The CPU initiates a disk read by writing a command, logical block number, and destination memory address to the memory-mapped address associated with the disk.

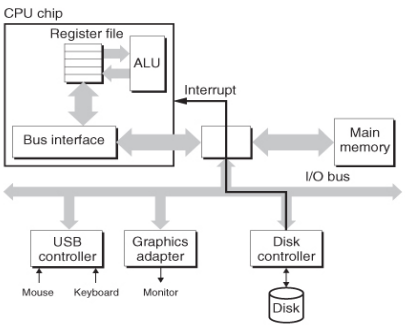


(b) The disk controller reads the sector and performs a DMA transfer into main memory.

25

KAPLAN

ACCESSING DISKS



(c) When the DMA transfer is complete, the disk controller notifies the CPU with an interrupt.

26

KAPLAN

END

KAPLAN

KAPLAN

Fundamentals of Computer Systems

Topic 04
Digital Logic

Learning Outcomes

1. Count in denary, binary, octal and hexadecimal number systems.
2. Convert integers between denary, binary, octal and hexadecimal number systems.
3. Convert fractions between denary and binary number systems.
4. Perform addition and subtraction in binary system.
5. Explain the 2's complement and sign and magnitude method of representing negative numbers.
6. Develop and use the truth tables for the AND, OR, XOR and NOT logic operators.
7. Write Boolean expression and draw corresponding logic circuit diagram and vice versa.
8. Use the laws of Boolean algebra to simplify Boolean expression.
9. Applications of logic gates in half-adder, full adder, decoder, multiplexer, and latches.

INTRODUCTION TO NUMBER SYSTEMS

- The base (Radix) of a number system determine the number of symbols used in the system.
  - The denary number system 65_{10}
  - The binary number system $0100\ 0001_{2}$
  - The octal number system 101_{8}
  - The hexadecimal number system 41_{16}

3

KAPLAN

INTRODUCTION TO NUMBER SYSTEMS

- The Denary number system
  - Also known as the 'decimal' system, has a radix of 10 and uses ten digits:
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
- The Binary number system
  - The binary number system has a radix of 2 and uses 2 digits :
    - 0 or 1 to represent some value.
  - Each digit is known as a bit – 'binary digit'.

4

KAPLAN

INTRODUCTION TO NUMBER SYSTEMS

- The Octal number system
  - The octal number system has a radix of 8 and uses 8 digits to represent value:
    - 0, 1, 2, 3, 4, 5, 6, 7

- The hexadecimal number system
  - The hexadecimal number system has a radix of 16 and uses 10 digits and 5 characters to represent value
    -0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A - 10, B - 11, C - 12, D - 13, E - 14, F - 15

5

KAPLAN

POSITIONAL VALUE METHOD

- In a positional number system, the value of each digit is determined by the position of that digit.

1000	100	10	1	Decimal number position weighted value
10^3	10^2	10^1	10^0	Decimal number position
0	0	6	5	Decimal number pattern
MSN			LSN	Decimal number position significance MSN – Most Significant number, LSN – Least Significance number

Figure 4.1 positional value of each digit in decimal system

6

KAPLAN

POSITIONAL VALUE METHOD

128	64	32	16	8	4	2	1	Bit position weighted value
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Bit position
0	1	0	0	0	0	0	1	Bit pattern
MSB							LSB	Bit position significance MSB – Most Significant Bit, LSB – Least Significance Bit

Figure 4.2 Positional value of each digit in binary system

7

KAPLAN

POSITIONAL VALUE METHOD

512	64	8	1	Octal number position weighted value
8^3	8^2	8^1	8^0	Octal number position
0	1	0	1	Octal number pattern
MSN			LSN	Octal number position significance MSN – Most Significant number, LSN – Least Significance number

Figure 4.3 Positional value of each digit in octal system

8

KAPLAN

POSITIONAL VALUE METHOD

4096	256	16	1	Hexa number position weighted value
16^3	16^2	16^1	16^0	Hexa number position
0	0	4	1	Hexa number pattern
MSN			LSN	Hexa number position significance MSN – Most Significant number, LSN – Least Significance number

Figure 4.3 Positional value of each digit in hexadecimal system

9

KAPLAN

NUMBER CONVERSION – DECIMAL TO BINARY

- Converting Decimal to Binary: Convert 65_{10} to binary

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Bit position
128	64	32	16	8	4	2	1	Bit position weighted value
0	1	0	0	0	0	0	1	0100 0001 →The number in binary system
0	64	0	0	0	0	0	1	65 →The number in decimal system

10

KAPLAN

NUMBER CONVERSION – DECIMAL TO OCTAL CONVERSION

- Converting Decimal to Octal: Convert 65_{10} to octal

1. Convert the decimal number to binary first.
2. Group the binary numbers into 3-bits section starting from the LSB to MSB.
3. Add zero(s) to the left of the MSB so that all the groups have 3-bits.
4. Write the 3-bit binary number to its octal value.

Additional bit added so that all the groups have 3 bits		MSB						LSB	
Binary value	0	0	1	0	0	0	0	1	$001\ 000\ 001_2$
Octal value		1		0		1			101_8

11

KAPLAN

NUMBER CONVERSION – DECIMAL TO OCTAL CONVERSION

- Converting Decimal to Hexadecimal: Convert 65_{10} to hexadecimal

1. Convert the decimal number to binary first.
2. Group the binary numbers into 4-bits section starting from the LSB to MSB.
3. Write the 4-bit binary number to its hexadecimal value

	MSB						LSB	
Binary value	0	1	0	0	0	0	0	1	$0100\ 0001_2$
Hexadecimal value		4			1				41_{16}

12

KAPLAN

REAL NUMBER (FRACTIONS) CONVERSION

- Convert 0.6875_{10} to binary
- $0.6875_{10} = 0.10110000_2$ (answer should be in 8-bit format)

				result	Integer part of the result	
0.6875	x	2	=	1.375	1	
0.375	x	2	=	0.75	0	
0.75	x	2	=	1.5	1	
0.5	x	2	=	1.0	1	
0	x	2	=	0	0	

13

KAPLAN

REAL NUMBER (FRACTIONS) CONVERSION

- Convert 0.6875_{10} to octal

1. Convert the fraction to binary first.
2. Group the binary numbers into 3-bits section starting from the Left to Right after the decimal point.
3. Add trailing zero(s) to the right to form groups have 3-bits if required.
4. Write the 3-bit binary number to its octal value.

Trailing 0 bit added so that all the groups have 3 bits	Decimal point									
Binary fraction value	.	1	0	1	1	0	0	0	0	0.10110000_2
Octal fraction value	.	5			4			0		0.540_8

14

KAPLAN

REAL NUMBER (FRACTIONS) CONVERSION

- Convert 0.6875_{10} to hexadecimal

1. Convert the fraction to binary first.
2. Group the binary numbers into 4-bits section starting from the Left to Right after the decimal point.
3. Add trailing zero(s) to the right to form groups have 4-bits if required.
4. Write the 4-bit binary number to its hexadecimal value.

Trailing 0 bit added so that all the groups have 4 bits	Decimal point								
Binary fraction value	.	1	0	1	1	0	0	0	0	0.10110000_2
Hexadecimal fraction value	.	B				0				$0.B0_{16}$

15

KAPLAN

REAL NUMBER (FRACTIONS) CONVERSION

- Convert 0.1011_2 to decimal

Decimal point	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	Bit position
	0.5	0.25	0.125	0.0625	0.03125	..	..	..	Bit position weighted value
.	1	0	1	1	0	0	0	0	0.10110000_2
	0.5	0	0.125	0.0625	0	0	0	0	0.6875_{10}

16

KAPLAN

NEGATIVE NUMBERS

- Sign and magnitude

	B7 (MSB) Sign Bit	B6	B5	B4	B3	B2	B1	B0 (LSB)
+127	0	1	1	1	1	1	1	1
-127	1	1	1	1	1	1	1	1

17

KAPLAN

NEGATIVE NUMBERS

- Two's complement

	B7 (MSB)	B6	B5	B4	B3	B2	B1	B0 (LSB)
+127	0	1	1	1	1	1	1	1
-128	1	0	0	0	0	0	0	0

18

KAPLAN

BINARY ARITHMETIC

- Addition

Positional value	128	64	32	16	8	4	2	1	
27_{10}	0	0	0	1	1	0	1	1	
155_{10}	1	0	0	1	1	0	1	1	
	0	0	1	1	0	1	1		Carry bit
182_{10}	1	0	1	1	0	1	1	0	

19

KAPLAN

BINARY ARITHMETIC

- Subtraction
- We can perform subtraction using the two's complement method by adding the numbers.
  - Subtract 12_{10} from 26_{10}, i.e., 26_{10} - 12_{10}
  - The expression 26_{10} - 12_{10} can be re-written as 26_{10} + (- 12_{10})

	Last carry bit ignored	128	64	32	16	8	4	2	1	
26		0	0	0	1	1	0	1	0	
(-12)		1	1	1	1	0	1	0	0	Two's complement
	1	1	1	1						Carry bit
14		0	0	0	0	1	1	1	0	

20

KAPLAN

BOOLEAN ALGEBRA

- Boolean algebra can be used to represent logic circuits and logic circuits can be designed to achieve a specific purpose, i.e., decoder, accumulator in the CPU.
- THREE (3) basic operations: 'addition', 'complementation' and 'multiplication'.

The Laws for Addition		
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

The Laws for Multiplication		
A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

The Laws for Complementation	
A	A'
0	1
1	0

21

KAPLAN

BOOLEAN ALGEBRA

- Laws of Boolean algebra

Idempotent	Complement	Associative	Commutative
A + A = A A · A = A	A + A' = 1 A · A' = 0	(A+B) + C = A+(B+C) (A · B) · C = A · (B · C)	A+B = B+A A·B = B·A
Distributive	**De Morgan**	**Identity**	**Involution**
A+B·C = (A+B)·(A+C) A·(B+C) = A·B + A·C	(A+B)' = A'·B' (A·B)' = A'+B'	A+0 = A A+1 = 1 A·1 = A A·0 = 0	(A')' = A

22

KAPLAN

SIMPLIFICATION OF BOOLEAN ALGEBRA

$$Q = A.B.\overline{C} + A.B.C + A.\overline{B}$$

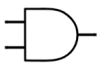
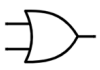
$A.B.\overline{C} + A.B.C + A.\overline{B}$	
$A.(B.\overline{C} + B.C + \overline{B})$	Distributive law: 'A' is common
$A.(B.(\overline{C} + C)) + \overline{B})$	Distributive law: 'B' is common
$A.(B.(1) + \overline{B})$	Complement law: $\overline{C} + C$ = 1
$A.(B + \overline{B})$	Complement law: $\overline{B} + B$ = 1
$A.1$	Identity law
A	

23

KAPLAN

LOGIC GATES

- Logic gates are electronic devices for implementing Boolean operations to achieve a specific application purpose, i.e., a logic circuit.

AND gate	OR gate	XOR gate	NOT gate
			

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

A	B
0	1
1	0

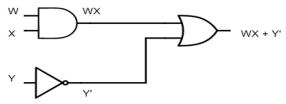
24

KAPLAN

DERIVING LOGIC CIRCUITS FROM BOOLEAN EXPRESSION

- The Boolean expression for a logic circuit can be marked by following the output of each logic gate as it connects to the input of another logic gate.

Example 1: W·X + Y'



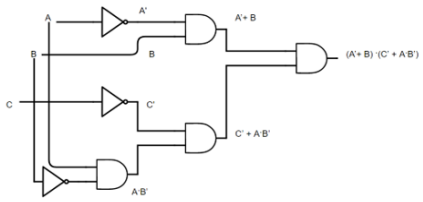
W	X	Y	WX	Y'	WX+Y'
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Logic/Truth table for W·X + Y'

KAPLAN

DERIVING LOGIC CIRCUITS FROM BOOLEAN EXPRESSION

Example 2: (A' + B) · (C' + A·B')



A	B	C	A'	A'+B	C'	B'	AB'	C'+AB'	(A' + B) · (C' + A·B')
0	0	0	1	1	1	1	0	1	1
0	0	1	1	1	0	1	0	0	0
0	1	0	1	1	1	0	0	1	1
0	1	1	1	1	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0
1	0	1	0	0	0	1	1	1	0
1	1	0	0	1	1	0	0	1	1
1	1	1	0	1	0	0	0	0	0

Logic/Truth table for (A' + B) · (C' + A·B')

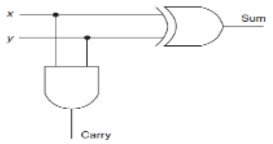
KAPLAN

APPLICATIONS OF LOGIC GATES

- The Half Adder

Input		Output	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder Logic Diagram



27

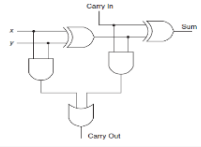
KAPLAN

APPLICATIONS OF LOGIC GATES

- The FULL Adder

Input			Output	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder Logic Diagram



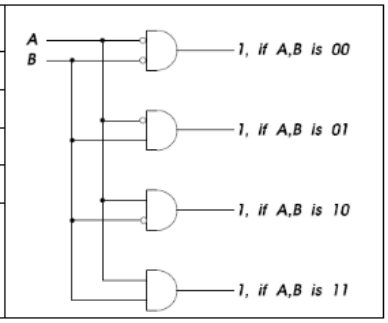
28

KAPLAN

APPLICATIONS OF LOGIC GATES

- The Decoder

A 2 to 4-bit decoder					
A	B	W0	W1	W2	W3
0	0	1			
0	1		1		
1	0			1	
1	1				1



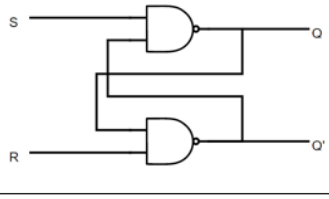
29

KAPLAN

APPLICATIONS OF LOGIC GATES

- The RS latch

The RS latch				
S	R	Q	Q'	COMMENTS
0	0	1	1	INVALID
0	1	1	0	SET
1	0	0	1	RESET
1	1	NC	NC	No change



30

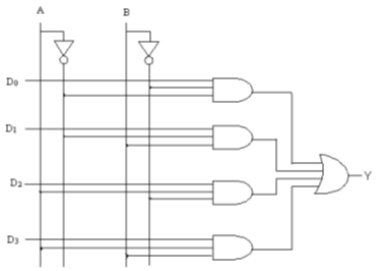
KAPLAN

APPLICATIONS OF LOGIC GATES

- The Multiplexer

The multiplexer

A	B	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3



31

KAPLAN

END

32

KAPLAN

KAPLAN

Fundamentals of Computer Systems

Topic 05
Introduction to C Programming

Learning Outcomes

1. Write simple C programs.
2. Use simple input and output statements.
3. Use fundamental data types.
4. Describe computer memory concepts.
5. Describe the use of arithmetic operators.

INTRODUCTION TO C PROGRAMMING LANGUAGES

- A program is a set of instructions that tells a computer how to perform tasks for the user.
- Includes a set of syntax and rules.
- Was evolved from another programming B by Dennis Ritchie at Bell Laboratories (1969 - 1973).
- Widely known as the development language of the UNIX operating system.

3

KAPLAN

SIMPLE C PROGRAM STRUCTURE

```
C program structure     Comments
=================       ============================================================

#include <stdio.h>      // The #include <stdio.h> is a preprocessor command.
                        // It tells the C compiler to include (standard input output library)
                        // stdio.h file before compilation.

int main()              // The next line int main() is the name of the main function
                        // where program execution begins.
{
  /* My first C program */      // Everything between the /*...*/ will be ignored by the
                                // compiler. It is used mainly for including comments in
                                // the program.
  printf("Welcome to FCS \n");  // The printf(...) is another function available in C which
                                // displays the message "Welcome to C" on the screen.
  return 0;                     // The return statement terminates the main() function and
                                // returns the value 0.
}
```

4

KAPLAN

SIMPLE C PROGRAM: ADDING TWO INTEGERS

```
// ========================================================
// Program name: addTwoNum.c
// Description: This program requests the user to enter 2 integers,
// compute the sum of the two numbers and print the result
// ========================================================

#include <stdio.h>

int main()
{
 // =====================================
 // declare and initialise working storage
 // =====================================
   int num1 = 0 // holds the first integer entered by user
   int num2 = 0 // holds the second integer entered by user
   int sum = 0;    // holds the sum of the two integers

 // =====================================
 // Prompts the user to enter two numbers
 // =====================================
   printf("\n\nPlease enter the first number: ");
   scanf("%d", &num1);

   printf("\n\nPlease enter the second number: ");
   scanf("%d", &num2);

 // ==================================
 // Compute the sum of the two numbers
 // ==================================
   sum = num1 + num2;

 // ==============
 // Prints the sum
 // ==============
   printf("\n\nThe sum of %d and %d is %d", num1, num2, sum);

   return 0;
}
```

5

KAPLAN

BASIC INPUT OPERATION WITH scanf()

- The scanf() (scan formatted) function is used for getting information from the user via a standard input device, typically a keyboard.

- It also does type conversion from ASCII to the type specified in the format string.

```
scanf("%c", &nextchar);

scanf("%f", &radius);

scanf("%d %d", &length, &height);

/* the & symbol refers to an address operator */
```

6

KAPLAN

BASIC INPUT OPERATION WITH printf()

- The printf() (print formatted) function is used to display output
- Uses a format to defines:
  - The text to print out
  - The specifications on how to print out values of variables and the variables whose value is to be printed

```
printf("%d is a prime number", 43);

printf("43 + 59 in decimal is %d\n", 43+59);

printf("a+b=%f\n", a+b);

printf("%d+%d=%d\n", a, b, a+b);
```

7

KAPLAN

FORMAT SPECIFICATION

- Both the printf() and scanf() use the following conventions
  - %d – decimal
  - %i – integer
  - %x – hexadecimal
  - %c – ASCII character
  - %s – string of ASCII characters
  - %f – floating point value
  - \n – new line
  - \t – tab

8

KAPLAN

C BASIC LANGUAGE SYNTAX

- Comments
  - Comments are used as in-text documentation and are ignored by the compiler
  - Useful for readability and maintainability.

```
/* For multiple lines comments*/

// For single line comment
```

9

KAPLAN

C BASIC LANGUAGE SYNTAX

- Identifiers
  - Used to label variables
  - Must begin with a letter A to Z or a to z or an underscore _ followed by zero or more letters or digits (0 to 9)
  - Cannot use punctuation characters such as @, $, and %
  - Case sensitive
- Naming conventions and best practices
  - Names should be meaningful
  - Reflect the intention of use
  - Uses camel-case
  - Declare and initialise at the same time

10

KAPLAN

C BASIC LANGUAGE SYNTAX

- Statement terminator (;)
  - -Each individual statement must end with a semicolon
  - -It indicate the end of one logical entity

- Whitespace
  - -Mainly used for readability
  - -Examples are blanks, tabs, newline, comments

KAPLAN

MEMORY CONCEPTS

- When we declare a variable int num1, the C compiler reserve a memory location in RAM and label that location as num1.

- For example, the scanf() function “&” denote the memory location of that variable. The & (ampersand) is the address operator.

```
scanf(“%d\n”, &num1);
```


KAPLAN

C BASIC LANGUAGE SYNTAX

- The C programming language reserved words and cannot be used for naming identifiers, function names etc. They represent special meaning to the C compiler.

Keywords				
auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typedef	
default	for	short	union	
Keywords added in C99 standard				
_Bool	_Complex	_Imaginary	inline	restrict
Keywords added in C11 standard				
_Alignas	_Alignof	_Atomic	_Generic	_Noreturn	_Static_assert	_Thread_local

13

KAPLAN

BASIC DATA TYPES IN C PROGRAMMING LANGUAGE

Data type	printf conversion specification	scanf conversion specification
Floating-point types		
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
Integer types		
unsigned long long int	%llu	%llu
long long int	%lld	%lld
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short	%hu	%hu
short	%hd	%hd
char	%c	%c

14

KAPLAN

ARITHMETIC IN C

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

15

KAPLAN

DECISION MAKING IN C

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

16

KAPLAN

LOGIC OPERATIONS

C operation	Arithmetic Operator	Meaning
AND	&&	Returns a 1 if BOTH conditions are true
OR	||	Returns a 1 if either or both conditions are true
NOT	!	The outcome is reversed

17

KAPLAN

ASSIGNMENT OPERATIONS

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
+=	`c += 7`	`c = c + 7`	10 to c
-=	`d -= 4`	`d = d - 4`	1 to d
*=	`e *= 5`	`e = e * 5`	20 to e
/=	`f /= 3`	`f = f / 3`	2 to f
%=	`g %= 9`	`g = g % 9`	3 to g

18

KAPLAN

OPERATORS PRECEDENCE

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the *innermost* pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
* / %	Multiplication Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated third. If there are several, they're evaluated left to right.
=	Assignment	Evaluated last.

19

KAPLAN

END

20

KAPLAN

KAPLAN

Fundamentals of Computer Systems

Topic 06
Structured Program Development in C

Learning Outcomes

1. Describe the basic problem-solving techniques
2. Develop algorithms through the process of top-down, stepwise refinement
3. Use the if selection statement
4. Use the if…else selection statement
5. Use the switch.. case statement

STRUCTURED PROGRAM DEVELOPMENT IN C

- Before writing a program to solve a particular problem, it's essential to have a thorough understanding of the problem and a carefully planned approach to solving the problem.

-Algorithm

-The solution to any computing problem involves executing a series of actions in a specific order

-Pseudocode

-Pseudocode is an artificial and informal language that helps you develop algorithms. Pseudocode helps you "think out" a program before you starts writing the codes.

3

KAPLAN

TOP-DOWN, STEPWISE REFINEMENT

- Top-down stepwise refinement is the progressive refinement in small steps of a program specification into a program, it is also known as top-down design.
- Help us to "think through" an algorithm before coding starts
- It is a commonly used technique in problem-solving.

4

KAPLAN

FORMULATING ALGORITHMS WITH TOP-DOWN, STEPWISE REFINEMENT

The steps are as follow:

1. Start with the problem statement.
2. Decompose the problem into smaller parts.
3. Take each "part", and break it further into more detailed steps.
4. Repeat Step 1-3 until each part is specific enough to write as pseudocode.

5

KAPLAN

FORMULATING ALGORITHMS WITH TOP-DOWN, STEPWISE REFINEMENT

- The problem statement:
  - Write a C program to request the user to enter 2 numbers, compute and prints the total of the 2 numbers.
- Decompose the problem:
  - Request the user to enter 2 numbers
  - Add the two numbers
  - Display the total
- Refinement:
  - Declare and initialise working storage
  - Prompt the user to enter 2 numbers
  - Add the two numbers and store the result in a variable total
  - Print total

6

KAPLAN

FORMULATING ALGORITHMS WITH TOP-DOWN, STEPWISE REFINEMENT

- Pseudocode:
  - Begin
    - Declare and initialize working storage
    - Prompt the user to enter 2 numbers
    - Compute the sum of the 2 numbers
    - Display the sum
  - End

KAPLAN

FORMULATING ALGORITHMS WITH TOP-DOWN, STEPWISE REFINEMENT

```
// This program prompts the user to enter 2 numbers
// and prints the sum of the 2 numbers

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int num1 = 0;         // variable to hold user input
   int num2 = 0;
   int sum = 0;          // variable to hold the sum

   // Prompts the user to enter 2 numbers
   printf("\nPlease enter the first number : ");
   scanf("%d", &num1);
   printf("\nPlease enter the second number : ");
   scanf("%d", &num2);

   // Compute the sum of the 2 numbers
   sum = num1 + num2;

   // Display the sum
   printf("\nThe sum of the 2 numbers is %d", sum);

   return 0;    // returns control to the calling program
}
```


KAPLAN

CONTROL STRUCTURES

- Structured programs are clearer, easier to debug and modify and more likely to be bug free.
- All programs can be written in terms of only three control structures:
  - Sequence
  - Selection
  - Repetition

KAPLAN

THE CONDITIONAL CONTROL STRUCTURES IN C

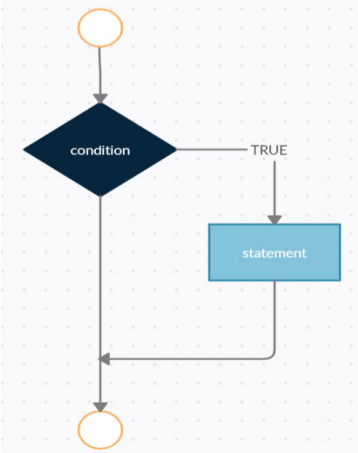
- The C programming language supports three condition control structures:
  - if ()
  - if… else…
  - switch… case

KAPLAN

THE IF STATEMENT

- A Boolean condition is evaluated and the statements inside the body of "if" execute if the given condition returns a true value.

```
if (boolean expression)
  {
       Statement… // execute if true
  }
```



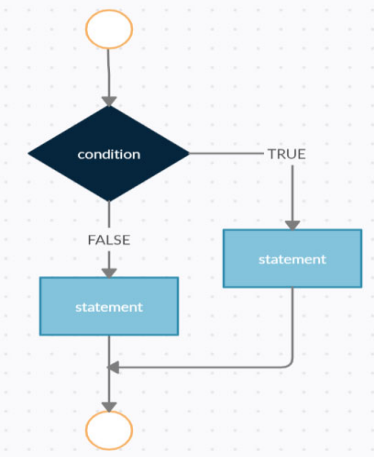
11

KAPLAN

THE IF.. ELSE STATEMENT

- If the Boolean condition returns 'true' then the statements inside the 'true' block are executed.
- If the Boolean condition returns 'false' then the statements inside the 'false' block are executed.

```
if (boolean expression)
   {
        True statement… // execute if true
   }
else
   {
        False statement… // execute if false
   }
```



12

KAPLAN

NESTED IF STATEMENT

- You can use an if-else statement within another if-else statement.
- When to use this nested structure depends on the problem statement.

```
if (boolean expression)
   {
      statement…
   }
else
   {
      if (boolean expression)
         {
            statement…
         }
      else
         {
            statement…
         }
   }
```

13

KAPLAN

COMPOUND STATEMEMNT

- If you need to include several statements in the body of an if, you need to enclose the set of statements in braces { and }.
- A set of statements contained within a pair of braces is called a compound statement or a block.

```
if ( grade >= 60 )
   {
        printf( "Passed.\n" );
    }
else
    {
        printf( "Failed.\n" );
        printf( "You must take this course again.\n" );
    }
```

14

KAPLAN

THE SWITCH.. CASE STATEMENT

- The switch.. case statement allows us to execute selected block of codes from multiple alternatives.
- It is a more elegant if-else statement structure.

```
switch (expression)
{
    case constant1:
       // statements
       break;

    case constant2:
       // statements
       break;
    .
    .
    .
    default:
       // default statements
}
```

15

KAPLAN

THE SWITCH.. CASE STATEMENT

- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.

16

KAPLAN

THE SWITCH.. CASE STATEMENT

- The break statement terminates the switch structure.
- The execution continues after switch statement.
- It is not essential for each case to have a break statement.
- If no break statement is included, the flow of control will continue to the next case until a break is reached or when the default is executed.
- The default case is optional and must appear at the end of the switch, mainly used for handling exceptions.

17

KAPLAN

END

18

KAPLAN

KAPLAN

Fundamentals of Computer Systems

Topic 07

C Program Control

Learning Outcomes

1. Describe the essentials of counter-controlled and sentinel-controlled iterations.
2. Use the for, while and do…while iteration statements to execute statements repeatedly.
3. Use logical operators to form complex conditional expressions in control statements.
4. Describe how to avoid the consequences of confusing the equality and assignment operators.

ITERATION (REPETITIONS) ESSENTIALS

- Iteration (repetition) structures, or loops, are used when a program needs to execute some instructions several times until some condition is met.
- There are two main types of repetitions:

  -counter-controlled

  -sentinel-controlled repetition.

KAPLAN

COUNTER-CONTROLLED REPETITION

- We know in advance the number of times the loop will be performed.
- A control variable is used to track the number of times the loop has performed, the control variable is typically incremented by 1 each time the loop is completed.
- When the control variable reaches the correct number of repetitions, the loop will terminate, and control is passed to the next statement.

KAPLAN

COUNTER-CONTROLLED REPETITION

The structure of the counter-controlled repetition requires:

1. control variable (or loop counter)
2. initial value of the control variable
3. increment (or decrement) by which the control variable is modified each iteration through the loop
4. condition that tests for the final value of the control variable

```
for (counter = 0; counter < 10; counter++)
    {
       statement...
    }
```

5

KAPLAN

SENTINEL-CONTROLLED REPETITIONS

- We do not know in advance the number of times the loop will be executed.
- A sentinel value is used to test the input value. If the condition evaluates to be true, the loop will exit.

```
counter = 2;
while (counter < 100)
   {
      counter = counter * 2
   }
```

- In this case, the sentinel value would be when counter >= 100.

6

KAPLAN

END

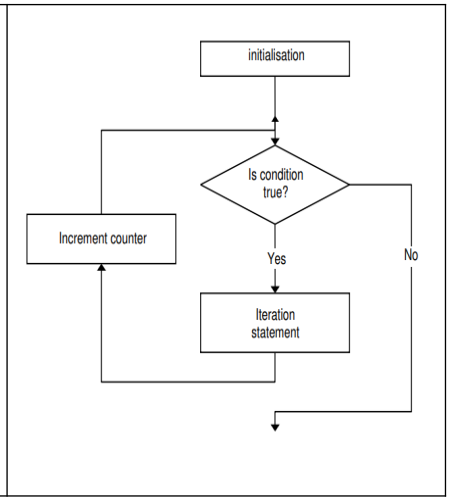
7

KAPLAN

THE FOR() LOOP STRUCTURE

Syntax:

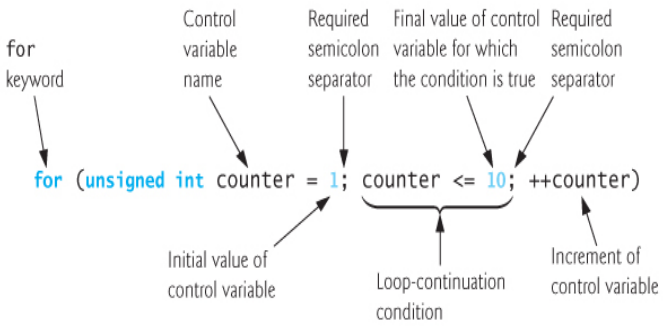
```
for ( init; condition; increment )
    {
        statement(s);
     }
```



8

KAPLAN

THE FOR () LOOP STRUCTURE



9

KAPLAN

THE FOR() LOOP STRUCTURE

```
int main()
{
   // Declare and initialise working storage
   int counter = 0;

   // Prints the 2 times table
   printf("\nThe 2 times table");
   printf("\n=================\n");

   // The for() structure
   for (counter = 1; counter <= 12; counter++)
     {
        printf("\n%d X 2 = %d", counter, counter*2);
     }

   return 0;    // returns control to the calling program
}
```

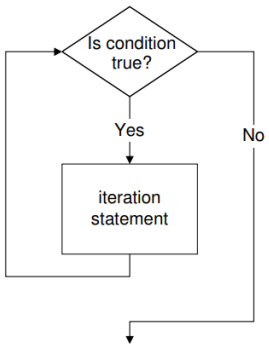
10

KAPLAN

THE WHILE() LOOP STRUCTURE

Syntax:

```
while ( condition )
    {
       statement(s);
    }
```



11

KAPLAN

THE WHILE() LOOP STRUCTURE

```
// This program demonstrates the while() loop structure

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int counter = 0;


   // The while() structure
   while (counter < 100)
     {
        printf("\nCounter value = %d", counter);
        counter = counter + 3
     }

   return 0;    // returns control to the calling program
}
```

12

KAPLAN

THE DO… WHILE LOOP STRUCTURE

- Like the while() loop structure except that it will execute the loop at least one time before checking the condition to exit the loop.
- The do… while loop test a condition at the end of the loop structure.
- If the condition is True, it will loop again but if the condition is False, it will exit the loop.

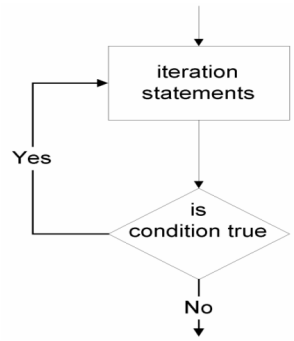
13

KAPLAN

THE DO.. WHILE() LOOP STRUCTURE

Syntax:

```
do
   {
      statement(s);
    } while (condition);
```



14

KAPLAN

LOGICAL OPERATORS - RECAP

The C programming language has 3 logical operators:

- logical AND (&&)
- logical OR (||)
- logical NOT (!)

15

KAPLAN

THE LOGICAL AND (&&)

The logical AND (&&) operator returns a True if both conditions evaluates to be True:

- A = 10, B = 20, C = 30 then (A < B) && (C > A) evaluates to a True
- A = 10, B = 20, C = 30 then (A < B) && (B == C) evaluates to a False

16

KAPLAN

THE LOGICAL OR(||)

The logical OR (||) operator returns a True if either or both conditions evaluates to be True:

- A = 6, B = 10 then ((A > 5) || (B > 50)) will evaluates to a True.
- A = 1, B = 2 then ((A > 5) || (B > 50)) will evaluates to a True.

17

KAPLAN

THE LOGICAL NOT (!)

The logical NOT (!) reverse the state of the condition, i.e., if the state is True, the NOT operator will reverse the state to False.

- A = 10, B = 10 then (A == B) evaluates to a True
- A = 10, B = 10 then !(A == B) evaluates to a False

18

KAPLAN

THE EQUALITY (==) & THE ASSIGNMENT (=) OPERATORS

- The assignment operator = assigned a value to a variable, i.e. counter = 10.
- The relational operator == checks for equality, i.e., (num1 == num2).
  - In this case, if num1 and num2 holds the same value, they are evaluated to be True.

19

KAPLAN

COMMON LOOP ERROR

```
// This program demonstrates an endless loop structure
// The loop will not terminate because we did not increment
// the counter value. The counter value remains the same
// through each loop and the while() condition will not
// not be false to exit the loop.
//
// To correct this problem, insert counter = counter + 1;
// within the while {}

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int counter = 0;     // this is the loop counter

   // The endless loop
   while (counter <= 10)
     {
        printf("This is the counter value = %d\n", counter);
     }

   return 0;    // returns control to the calling program
}
```

20

KAPLAN

END

KAPLAN

KAPLAN

Fundamentals of Computer Systems

Topic 08
C Functions Part I

Learning Outcomes

1. Construct programs modularly from small pieces called functions.
2. Use common math functions in the C standard library.
3. Create new functions.
4. Use mechanisms that pass information between functions.

FUNCTIONS IN C

- Top-down stepwise refinement approach to problem solving helps us to decompose the problem statement into smaller parts.
- These smaller parts form the basic building blocks in a C program and are known as functions.
- Function is a set of instructions to carry out a particular task
  - can be re-used
  - improving the quality of the program developed.

3

KAPLAN

FUNCTIONS IN C

- C functions are invoked by a function call and may return a value after it finishes executing.
- After being invoked, program control is transferred to the function.
- Control is returned to the calling function upon completion.

4

KAPLAN

TYPES OF FUNCTIONS

- Standard inbuilt functions
  - printf() and scanf() functions in the <stdio.h> library header file.
- User-defined functions.
  - These functions are developed by the user according to his requirements, i.e., addTwoNumbers(), findTotal()
- Every C program has at least one function – the main() function.

5

KAPLAN

THE FUNCTION HEADER FILE

- All standard inbuilt functions are declared in the header files, <filename.h>.
- The <stdio.h> header file contains some of the built-in functions we have used before:
  - printf() and scanf()
- Other inbuilt library header files in C includes conio.h, string.h, stdlib.h, math.h, time.h, ctype.h.

6

KAPLAN

FUNCTION SIGNATURE

```
return-type function-name (formal parameters list)
   {
      // function body
      // ...
      // return <value>
   }

Example:

int addTwoNumber(int first, int second)
   {
      return first + second;
   }
```

7

KAPLAN

FUNCTION SIGNATURE

- return-type
  - A function may return a value. The return type specifies the data type of the value to be returned by the function.

- function-name
  - this is the user defined function name; it should follow the naming conventions for variable.

8

KAPLAN

FUNCTION SIGNATURE

- formal parameter list
  - When the function is invoked, the data to be processed by the function is passed via the parameter list – they act like placeholder. The type, order and number of parameters is significant; however, the parameters are optional. Also known as actual parameter or argument.

- function body
  - It contains the C statements to be executed and define what the function will achieve.

9

KAPLAN

USER DEFINED FUNCTION CALL

```
// This program demonstrates a function that computes
// the sum of two numbers

#include <stdio.h>

int addTwoNumber(int first, int second)
   {
      return (first + second);
   }

int main()
{
   // Declare and initialise working storage
   int num1 = 3;
   int num2 = 5;
   int sum = 0; // holds the sum of the two numbers

   // invoke the function
   sum = addTwoNumber(num1, num2);

   printf("\nThe sum of %d and %d is %\n\n", num1, num2, sum);

   return 0;    // returns control to the calling program
}
```

10

KAPLAN

FUNCTION CALL VALIDATION

When the function is invoked, the C compiler will check to ensure the following conditions are correct:

- the number of arguments is correct,
- the arguments are of the correct type,
- the argument types are in the correct order, and
- the return type is consistent with the context in which the function is called.

KAPLAN

COMMONLY USED MATH LIBRARY FUNCTIONS

Function	Description	Example
`sqrt(x)`	square root of x	`sqrt(900.0)` is 30.0 `sqrt(9.0)` is 3.0
`cbrt(x)`	cube root of x (C99 and C11 only)	`cbrt(27.0)` is 3.0 `cbrt(-8.0)` is -2.0
`exp(x)`	exponential function e^x	`exp(1.0)` is 2.718282 `exp(2.0)` is 7.389056
`log(x)`	natural logarithm of x (base e)	`log(2.718282)` is 1.0 `log(7.389056)` is 2.0
`log10(x)`	logarithm of x (base 10)	`log10(1.0)` is 0.0 `log10(10.0)` is 1.0 `log10(100.0)` is 2.0
`fabs(x)`	absolute value of x as a floating-point number	`fabs(13.5)` is 13.5 `fabs(0.0)` is 0.0 `fabs(-13.5)` is 13.5
`ceil(x)`	rounds x to the smallest integer not less than x	`ceil(9.2)` is 10.0 `ceil(-9.8)` is -9.0
`floor(x)`	rounds x to the largest integer not greater than x	`floor(9.2)` is 9.0 `floor(-9.8)` is -10.0
`pow(x, y)`	x raised to power y (x^y)	`pow(2, 7)` is 128.0 `pow(9, .5)` is 3.0

KAPLAN

COMMONLY USED MATH LIBRARY FUNCTIONS – cont'd

`fmod(x, y)`	remainder of x/y as a floating-point number	`fmod(13.657, 2.333)` is `1.992`
`sin(x)`	trigonometric sine of x (x in radians)	`sin(0.0)` is `0.0`
`cos(x)`	trigonometric cosine of x (x in radians)	`cos(0.0)` is `1.0`
`tan(x)`	trigonometric tangent of x (x in radians)	`tan(0.0)` is `0.0`

13

KAPLAN

END

14

KAPLAN

KAPLAN

Fundamentals of Computer Systems

Topic 09
C Functions Part II

Learning Outcomes

1. Explain the purpose of function prototypes.
2. Explain how the function call/return mechanism is supported by the function call stack and stack frames.
3. Explain the role of header files in C.
4. Explain the different function call methods.
5. Explain the different storage classes.

FUNCTION PROTOTYPE

- A function prototype is the declaration of a function that specifies the function's name, parameters and return type before the main() function.
- Need to declare before use.
- Provides information to the compiler that the function may later be used in the program.
- The compiler uses the function prototypes to validate function calls to prevent improper call to functions.

3

KAPLAN

FUNCTION PROTOTYPE

```
#include <stdio.h>      // to include the standard input and output libraries

void f_cal(int num1, int num2, int num3, int num4, int num5);   // declare funtcion prototype

int main()
{
        f_cal(6,5,4,3,2);       // invoke function to compute
        return 0;
}

void f_cal(int num1, int num2, int num3, int num4, int num5)
{
        /*====================================*/
        /* Declare local working variables      */
        /*====================================*/
        int sum = 0;            // to hold the sum of the 5 integers
        double average = 0.0;   // to hold the average of the 5 integers

        sum = num1 + num2 + num3 + num4 + num5; // calculate total sum
        average = sum/5;                        // calculate average

        printf("the sum of the 5 numbers is : %d\n", sum);
        printf("the average value of the 5 numbers is : %f\n",average);
}
```

4

KAPLAN

FUNCTION CALL STACK & STACK FRAMES

- Stack – the memory storage (stack) in RAM used by the function.
  - The stack is used to store function arguments, return value, registers value, and for local variables.
- Stack frame - The portion of the stack allocated for each single function call.

5

KAPLAN

HEADER FILES IN C

Header	Explanation
<assert.h>	Contains information for adding diagnostics that aid program debugging.
<ctype.h>	Contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.
<errno.h>	Defines macros that are useful for reporting error conditions.
<float.h>	Contains the floating-point size limits of the system.
<limits.h>	Contains the integral size limits of the system.
<locale.h>	Contains function prototypes and other information that enables a program to be modified for the current locale on which it's running. The notion of locale enables the computer system to handle different conventions for expressing data such as dates, times, currency amounts and large numbers throughout the world.
<math.h>	Contains function prototypes for math library functions.
<setjmp.h>	Contains function prototypes for functions that allow bypassing of the usual function call and return sequence.
<signal.h>	Contains function prototypes and macros to handle various conditions that may arise during program execution.
<stdarg.h>	Defines macros for dealing with a list of arguments to a function whose number and types are unknown.
<stddef.h>	Contains common type definitions used by C for performing calculations.
<stdio.h>	Contains function prototypes for the standard input/output library functions, and information used by them.
<stdlib.h>	Contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers and other utility functions.
<string.h>	Contains function prototypes for string-processing functions.
<time.h>	Contains function prototypes and types for manipulating the time and date.

6

KAPLAN

PASSING ARGUMENTS TO FUNCTION

- When invoking a function, there are two ways that arguments can be passed to the function for processing:
  - call by value
  - call by reference.
- The parameters defined in the function header are known as formal parameters and the parameters being passed to the function are known as arguments.

KAPLAN

CALL BY VALUE

- This is the default method for passing arguments to a function.
- A copy of the actual value of an argument is passed into the formal parameter of the function.
- Changes made to the parameter inside the function have no effect on the argument.

KAPLAN

CALL BY REFERENCE

- Instead of passing a copy of the values, the memory address (reference) of the formal arguments is passed to the formal parameters.
- The argument's address is used to access the actual value. The actual value at the memory address location is changed permanently if it is modified.
- The arguments use the '&' operator to pass the address and the formal parameters receiving will used the '*' operator, which represent an address pointer that point to the memory location of the argument.

9

KAPLAN

CALL BY REFERENCE

```
# include <stdio.h>

void swap(int *first, int *second)
{
   int temp;

   temp = *second;
   *second = *first;
   *first = temp;
}

int main()
{
    // declare and initialise working storage
    int num1 = 11;
    int num2 = 44;

    // invoke the swap function- passing the address of the arguments
    swap(&num1, &num2);

    printf("\n\nAfter Swapping\nnum1 = %d\nnum2 = %d\n", num1, num2);

   return 0;
}
```

10

KAPLAN

STORAGE CLASSES

A storage class specifier can be used to determine:

- if a variable has internal, external, or no linkage;
- be stored in memory or in a register;
- receives the default initial value of 0 or an indeterminate default initial value;
- can be referenced throughout a program or only within the function, block, or source file where the variable is defined;
- If the storage duration for the object is maintained throughout program run time or only during the execution of the block where the object is defined.

KAPLAN

THE AUTO STORAGE CLASS SPECIFIER

- The auto storage class is the default for variables declared inside a block - it is usually redundant in a data declaration.

- Each time a block is entered, storage for auto variables defined in that block are made available. When the block is exited, the variables are no longer available for use.

KAPLAN

THE STATIC STORAAGE CLASS SPECIFIER

- These variables have static storage duration
  - The memory for these objects is allocated when the program begins running and is freed when the program terminates.
- Static storage can have static duration but local scope.

KAPLAN

THE EXTERN STORAGE CLASS SPECIFIER

- Allows you to declare variables that several source files can use.
- All extern variables have static storage duration.
- Memory is allocated before the main function starts executing and is released when the program terminates.
- The scope depends on the location of the declaration in the program text.
  - If the declaration appears within a block, the variable has block scope; otherwise, it has file scope.

KAPLAN

THE REGISTER STORAGE CLASS SPECIFIER

- The register storage class specifier stores the variable's value in the CPU register.
- Mainly for heavily used variable like loop control variable.
- However, due to the limited registers that a CPU have, this may not happen, and will be treated as the storage class specifier auto.

KAPLAN

END

KAPLAN

KAPLAN

Fundamentals of Computer Systems

Topic 10
C Arrays

Learning Outcomes

1. Use the array data structure to represent lists and tables of values.
2. Define an array, initialise an array and refer to individual elements of an array.
3. Define symbolic constants.
4. Pass arrays to functions.
5. Use arrays to store, sort and search lists and tables of values.

ARRAYS IN C

- Arrays are fixed-size sequential collection of elements of the same data type.
- We use an array subscript (index) to access each element. An array can be of data type as int, double, char etc.
- Used to simplify the creation of algorithms to solve problems that would otherwise be complex and time-consuming.

KAPLAN

DECLARING AND INITIALIZING AN ARRAY

- We can declare and initialise and integer array as follow:

```
int testArray[10];
int testArray[10] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
```

- Accessing an element in the array:

```
testArray[0]   // refers to the first element of the array
```

- Assigning a value to an array element:

```
testArray[0] = 50;
```


KAPLAN

CALCULATING THE SIZE OF AN ARRAY

- We can calculate the size of an array by using the sizeof() function.
- The sizeof()function is a built-in function that is used to calculate the size (in bytes) of a data type RAM.

```
lengthOfArray = sizeof(testArray) / sizeof(testArray[0]);
```

5

KAPLAN

STATIC AND DYNAMIC ARRAYS

- Static arrays are fixed and cannot changed during execution and are defined by the programmer, i.e., int testArray[10].
- Dynamic arrays are dynamic, and the program will calculate and allocates memory for the array during execution.
  - For example: int testArray[] = {10,12,14,12,14};
  - During execution, the testArray[] will be created with the size of 5 elements.

6

KAPLAN

ARRAY BOUND CHECKING IN C

- Bounds checking is used to check if an array index is within the bounds of the array.
- An error is usually flagged for a failed bound check.
- Bounds checking is time-consuming and not always done.
  - Bounds-checking elimination is a compiler optimization technique that eliminates unneeded bounds checking.

7

KAPLAN

WORKING WITH ONE DIMENSIONAL ARRAY

```
// This program demonstrates how to print all the array elements
// using a for() loop

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int testArray[10] = {1,3,5,7,9,11,13,15,17,19};
   int counter = 0;     // the loop counter

   // Print the testarray[] contents
   for (counter = 0; counter <= 10; counter++)
     {
        printf("\ntestarray[%d] = %d", counter, testarray[counter]);
     }

   return 0;    // returns control to the calling program
}
```

8

KAPLAN

TWO (2) DIMENSIONAL ARRAYS

- A 2-dimensional array is like a two one-dimensional array that has x number of rows and y numbers of columns.
- To declare a 2-dimensional array: type arrayName[row][column]

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

9

KAPLAN

INITIALIZING A 2-DIMENSIONAL ARRAY

You can initialise a 2-dimensional array during declaration as follow:

```
int testArray[3][4] = {
                        {1, 2, 3, 4},
                        {5, 6, 7, 8},
                        {9, 10, 11, 12}
                      };
```

	Column 0	Column 1	Column 2	Column 3
Row 0	1	2	3	4
Row 1	5	6	7	8
Row 2	9	10	11	12

10

KAPLAN

WORKING WITH 2-DIMENSIONAL ARRAY

- You can use a nested for() loop to access all the elements of an array.
- The outer loop refers to the row and the inner loop refers to the column.
- The outer loop index and the inner loop index will together uniquely identify each element.

KAPLAN

WORKING WITH 2-DIMENSIONAL ARRAY

```
// This program demonstrates a 2-dimensional array using a nested
// for() loop to print the contents

#include <stdio.h>

int main()
{
   // Declare and initialise working storage
   int testArray[3][4] = {{1,2,3,4},
                          {5,6,7,8},
                          {9,10,11,12}
                         };

   int outer = 0;       // the outer loop counter - the rows
   int inner = 0;       // the inner loop counter - the columns

   // Print the testarray[][] contents
   for (outer = 0; outer < 3; outer++)
     {
        for (inner = 0; inner < 4; inner++)
          {
             printf("\ntestArray[%d][%d] = %d", outer, inner, testArray[outer][inner]);
          }
     }

   return 0;    // returns control to the calling program
}
```


KAPLAN

CHARACTER ARRAY AND STRINGS

- A string is a sequence of characters of a given length. We can declare and initialise a character array as follow:

  char zString[] = “This is a string stored as a character array”;

- The string is a one-dimensional array of characters which is terminated by a null character '\0'.
- C strings functions look for the null terminator and use it to determine the end of string.

13

KAPLAN

SOME COMMON STRING FUNCTIONS

- **strlen(s1)** – this function calculates and return the length of a s1, the length does not include the ‘\0’ null terminator.
- **strcpy(s1, s2)** – this function copies string s2 into string s1, including the null terminator ‘\0’.
- **strcat(s1, s2)** – this function adds s2 to the end of s1.
- **strcmp(s1, s2)** – this function returns a number based on a numerical comparison of the two strings, the function returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
- **strchr(s1, ch)** – this function returns a pointer to the first occurrence of character ch in string s1.
- **strstr(s1, s2)** – this function returns a pointer to the first occurrence of string s2 in string s1.

14

KAPLAN

END

15

KAPLAN