SECRET SAUCE

AL - PAPER 3

Zain Merchant

Data Representation

User-defined data types

Zain Merchant

User-defined data type set

- A set user-defined data type is a composite data type which includes a list of unordered elements
- Set theory operations, such as intersection and union, can be applied to these elements
- A set data type includes the type of data/data type it uses as part of its definition
- All the elements are of the same data type.

User-defined data type record

- A user-defined record data type is a composite data type
- It uses other data types in its definition to form a single new data type
- The data types referenced may be primitive data types from a programming language or they may be other user-defined data types.
- Includes related items
- Includes a fixed number of items.

Non-composite data type

- Non-composite data types can both be user-defined or primitive
- Non-composite data types do not refer to other data types in their definition / contain one data type in their definition
- Non-composite data types can be primitive/enumerated/pointer

Composite data type

- Composite data types can be user-defined or primitive
- Composite data types refer to other data types in their definition/contain more than one data type in their definition
- Composite data types can be record/set/class

Zain Merchant

User-defined data type enumerated

- A user-defined non-composite (data type)
- with a list of all possible values
- that is ordered.

User-defined data type pointer

- A user-defined non-composite (data type)
- that stores addresses/memory locations only
- and indicates the type of data stored in the memory location.

Defining enumerated

```
TYPE <identifier> = (value1, value2, value3, …)
```

Example

```
TYPE Season = (Spring, Summer, Autumn, Winter)
```

Using enumerated

```
DECLARE ThisSeason : Season
ThisSeason ← Spring
```

Defining pointer

```
TYPE <identifier> = ^<data type>
```

Example - declarations of pointer type

```
TYPE TIntPointer = ^INTEGER
TYPE TCharPointer = ^CHAR
```

Example - declarations of a pointer variable

```
DECLARE MyPointer : TIntPointer
```

Using pointer

```
DECLARE NextSeason : Season
DECLARE MyPointer : TIntPointer

MyPointer ← ^ThisSeason
NextSeason ← MyPointer^ + 1
// access the value stored at the memory address
```

Defining record

```
TYPE <identifier1>
     DECLARE <identifier2> : <data type>
     DECLARE <identifier3> : <data type>
     ...
ENDTYPE
```

Example

```
TYPE StudentRecord
     DECLARE LastName : STRING
     DECLARE FirstName : STRING
     DECLARE DateOfBirth : DATE
     DECLARE YearGroup : INTEGER
     DECLARE FormGroup : CHAR
ENDTYPE
```

Using record

```
DECLARE Pupil1 : StudentRecord
DECLARE Pupil2 : StudentRecord
DECLARE Form : ARRAY[1:30] OF StudentRecord

Pupil1.LastName ← "Johnson"
Pupil1.Firstname ← "Leroy"
Pupil1.DateOfBirth ← 02/01/2005
Pupil1.YearGroup ← 6
Pupil1.FormGroup ← 'A'
Pupil2 ← Pupil1

FOR Index ← 1 TO 30
     Form[Index].YearGroup ← Form[Index].YearGroup + 1
NEXT Index
```

Defining set

```
TYPE <identifier1> = SET OF <data type>
DEFINE <identifier2> (value1, value2, value, … ) : <identifier1>
```

Example

```
TYPE LetterSet = SET OF CHAR
DEFINE Vowels ('A','E','I','O','U'): LetterSet
```

Data Representation

File organisation and access

Zain Merchant

Hashing algorithm in file access

- A hashing algorithm is used in direct access methods on random and sequential files
- It is a mathematical formula used to perform a calculation applied to the key field of the record being searched / stored
- The result of the calculation gives the address where the record should be found / stored.

Collision from hashing algorithm

- A collision is when the two values / data items in the key field for two records (pass through a hashing algorithm and) result in the same hash value
- so the location identified (by the hashing algorithm) may already be in use // two records cannot occupy the same address.

Methods of overcoming collision from hashing algorithm when writing

- A process of collision resolution is used
- Start at the original hashed storage space
- go through the following spaces in a linear fashion
- and store the data item in the first available slot.

OR

- Search the overflow area
- go through the following spaces in a linear fashion
- and store the data item in the first available slot.

Serial file organisation

- Records are stored one after the other as they are collected // records are stored in chronological order
- New records are appended to the end of the file.

Example use of serial file organisation

- Creating unsorted / temporary transaction files
- Creating data logging files

Sequential file organisation

- **Records** (in the file) are ordered
- based on the key field
- A new version (of the file) has to be created to update the file

Random file organisation

- **Records** are stored in no particular order within the file // There is no sequencing in the placement of the **records**
- There is a relationship between the key of the **record** and its location within the file // a hashing algorithm is used to find the location of the record
- Updates to the file can be carried out directly.

Sequential file access

- Sequential access method searches for records one after the other from the physical start of the file until the record is found/the end of file.

Sequential file access on serial organisation

- For serial files, records are stored in chronological order every record needs to be checked until the record is found, or all records have been checked.

Sequential file access on sequential organisation

- For sequential files, records are stored in order of a key field/index, and it is the key field/index that is compared. every record is checked until the record is found, or the key field of the current record is greater than the key field of the target record.

Direct file access

- Direct access allows a record to be found in a file without other records being read.
- Records are found by using the key field of the target record // the location of the record is found using a hashing algorithm.

Direct file access on sequential organisation

- In sequential files, an index of all key fields is kept
- The index is searched for the address of the file location where the target record is stored.

Direct file access on random organisation

- A hashing algorithm is used on the key field of the record
- to calculate the address of the memory location where the target record is expected to be stored.
- Method to find a record if it is not at the expected location e.g. linear probing, search overflow area etc

Zain Merchant

Methods of overcoming collision from hashing algorithm when reading

- A collision occurs when the record key doesn't match the stored record key this means the determined storage location has already been used for another record.
- If the record is to be stored
- Search the file linearly
- to find the next available storage space (closed hash)
- Search the overflow area linearly
- to find next available storage space (open hash)
- If the record is to be found
- search the overflow area linearly (open hash) until the matching record key is found
- search linearly from where you are (closed hash) until the matching record key is found
- If not found record is not in file

Zain Merchant

Data Representation

Floating-point numbers, representation and manipulation

Zain Merchant

Changing allocation of bits for mantissa and exponent

- When the number of bits in the mantissa is raised, the precision / accuracy of the number represented increases // when the number of bits in the mantissa is lowered, the precision / accuracy of the number represented reduces.
- When the number of bits in the exponent is reduced, the range of numbers that can be represented is reduced // when the number of bits in the exponent is increased, the range of possible numbers that can be represented increases.
- When the range increases the accuracy decreases // When the range decreases the accuracy increases.

Why binary representation is sometimes approximation to the real number

- Real numbers (can) have a fractional part (such as 1/3 and ½) / (such as 0.4 and 0.25)
- The fixed length of the storage means that you can't store very large / very small numbers
- Binary numbers represent numbers based on powers of 2, with limited fractional representations such as 1/2, 1/4, 1/8, 1/16, etc.
- It isn't possible to store all fractions with the level of precision provided by this system
- the fractional part of the number is as close as possible within these constraints.

Underflow in binary floating-point system

- Following an arithmetic/logical operation
- the result is too small to be precisely represented in the available system // When the number of bits is not enough / too small for the computer's allocated word size / to represent the binary number

Why binary numbers are stored in normalised form

- To store the maximum range of numbers in the minimum number of bytes / bits
- Normalisation minimises the number of leading zeros/ones represented
- Maximising the number of significant bits // maximising the (potential) precision / accuracy of the number for the given number of bits
- enables very large / small numbers to be stored with accuracy.
- Avoids the possibility of many numbers having multiple representations.

Affect of decreasing mantissa

Reduction in precision as the number of bits in the mantissa has decreased

Affect of increasing exponent

Increase in range as the number of bits in the exponent has increased

Communication and internet technologies

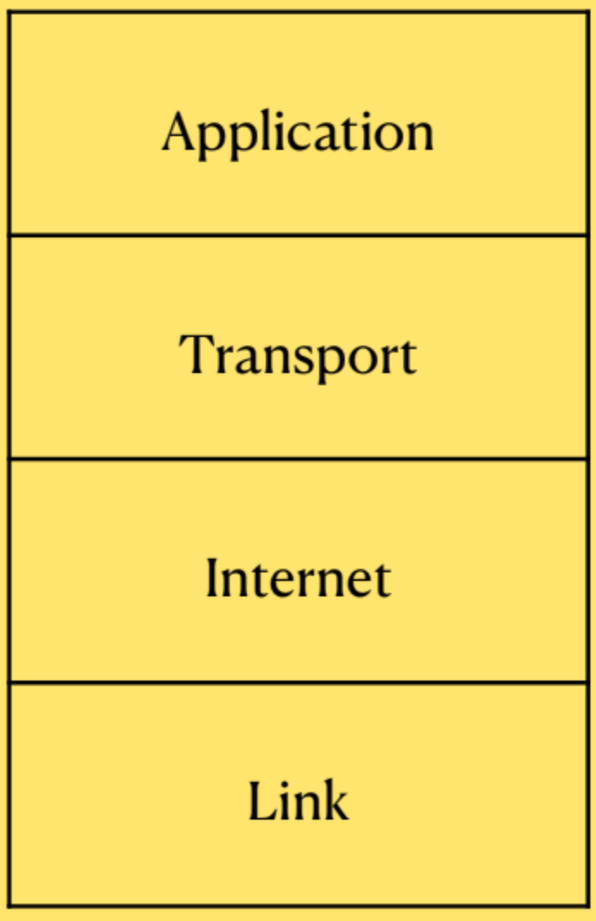
Protocols

Zain Merchant

How layers of TCP/IP protocol stack interact with each other

- Each layer can only accept input from the next higher layer or the next lower layer
- There is an interface between the adjacent layers which is the only interaction between layers
- Data is added to the headers as the frames/packets pass through the layers
- The interactions are carried out by installed software
- MP5 User interaction takes place at the highest/Application layer of the stack through protocols associated with that layer of the stack
- Direct access to hardware takes place at the lowest/Link layer of the stack.

TCP/IP protocol suite



Application layer protocols

- HTTP/HTTPS: For sending and receiving / transferring web pages / hypertext
- FTP: For sending and receiving files over a network / between devices
- POP3: Pull protocol / for receiving / downloading emails
- IMAP: Pull protocol / for receiving / downloading emails
- SMTP: Push protocol / for sending / uploading emails
- BitTorrent: Peer-to-peer file sharing over a network

Purpose and function of application layer

- The application layer provides access to all the programs that exchange data // Interacts directly with user. used by, for example, web browsers, server software.
- Communicates/enables data transfer to/from Transport layer // It allows applications to access the services used in other TCP/IP layers.
- It defines the protocols that any application uses to allow the exchange of data.

Why protocols are essentials for communication

- Protocols provide a standard set of rules that enables successful data transfer between devices.
- Allows communication between devices on different platforms.
- Makes communications independent of software and hardware.

Zain Merchant

Function of transport layer

- The transport layer is responsible for delivery of data from the source host to the destination host
- It is where data is broken up into packets and sent to the internet layer
- Adds the sequence number to the packet header
- It establishes end to end contact
- It ensures data arrives error free // It retransmits packets if lost.

BitTorrent protocol provides peer-to-peer file sharing

- BitTorrent allows the sharing of files between thousands of users who are connected together over the internet.
- It allows more users to share files with each other than would be the case with a peer-to-peer network.
- Users share files directly with each other // the users' computers are acting as peers
- no web server / central device is used // all users are of equal status.

Purpose of link layer

- To ensure correct network protocols are followed
- To enable the upper layers to access the **physical medium** //
- enables connection/ communication with the internet / network layer
- To be responsible for transporting data within the network/local segments
- To format the data into frames for transmission
- Maps IP addresses to MAC/Physical addresses.

Function of internet layer

- The Internet Layer identifies the intended network and host
- It transmits packets to the (Data) Link / Physical Layer
- It routes the packets independently through the **optimum** route
- It addresses packets with their source and destination IP addresses
- It then uses an IP address and port number to form a socket.

Zain Merchant

Why protocol is used in communication between computers

- Protocols set a standard for communication
- Protocols enable communication/compatibility between devices from different manufacturers/platforms
- If two devices were sending messages to each other but using different protocols, they would not be able to communicate properly

IMAP

- used by email clients to **retrieve** email messages // a pull protocol
- from a mail server (over a TCP/IP connection)
- keeps the server and client in sync (by not deleting the original email). // allows a copy of the email to be downloaded from the mail server.

How applications use BitTorrent

- BitTorrent client software made available
- A computer joins a swarm by using this to load a Torrent descriptor file
- A server called a tracker that keeps records of all the computers in the swarm and shares their IP addresses allowing them to connect to each other
- One computer in the swarm must have a complete copy of the torrent to be shared
- Torrent is split into small pieces
- Pieces of the torrent are both downloaded and uploaded
- Once a computer has a piece it can become a seed and upload
- Leeches download much more than they upload

Network model used in BitTorrent

Peer-to-peer

Communication and internet technologies

Circuit and packet switching

Zain Merchant

Circuit switching

- A dedicated circuit / channel is required
- The circuit is established before the transmission begins
- The circuit lasts for the whole of the transmission // The circuit is closed at the end of the transmission
- Data travels in a continuous stream along the same route
- Transmission is usually bidirectional.

Benefits of circuit switching

- No need for data to be reassembled // data / frames arrive in the same order in which they were sent
- Suitable for real time transmission // fast data transfer rate
- The whole of the bandwidth is available

Drawbacks of circuit switching

- No other transmission can use the same circuit when it is in use // Bandwidth can be wasted as it cannot be used by other messages
- Not secure // Can be intercepted as all data travelling along the same route
- If there is a problem with the route the transmission ends // No other route is available without first doing the setup
- The circuit is always there whether or not it's being used
- Can take time to set up before transmission starts.

Zain Merchant

Packet switching

- The data to be transmitted is divided into equal sized packets
- A packet header is attached to each packet containing key information such as source/ destination IP addresses, packet number, etc
- Packets are transmitted independently and may travel though different routes/paths to the destination
- Routes are determined using a routing table//Packets take the optimum route depending on congestion
- The packets usually arrive out of order
- The packets are reassembled in the correct order at the destination // The packets are re-ordered using the sequence number/the header
- If packets are missing/corrupted a re-transmission request is sent / packets are re-sent.

Benefits of packet switching

- Packets are more likely to arrive because they can be re-routed if a problem occurs with one of the routes//Packets are more likely to arrive because if a packet is lost, it can be re-transmitted
- Bandwidth can be shared allowing packets from different messages to share the same path
- Considered secure as the packets generally travel via different routes
- High data transmission rate is possible

Drawbacks of packet switching

- Time delay because packets need to be re-ordered/reassembled at the destination// Time delay caused by missing
- packets needing to be re-sent//Time delay because it has to share the bandwidth of the circuit / channel with other packets
- Requires a complex algorithm to function
- Needs lots of RAM to handle large amounts of data.

Function of a router in packet switching

- The router examines the packet's header
- It reads the IP address of the destination (from the packet header)
- A router has access to a routing table
- containing information about, e.g., available hops / netmask / gateway used
- and the status of the routes along the route
- the router decides on the next hop / best route
- and sends the packet on its next hop.

Hardware and virtual machines

Processors, parallel processing and virtual machines

Zain Merchant

Features of RISC

- low number of instruction formats //low number of instruction sets
- uses single-clock cycle instructions
- uses fixed length instructions
- uses many general-purpose registers
- works well with pipelining
- hard-wired control unit
- makes extensive use of RAM
- uses a low number of addressing modes
- the design emphasis is on the software.

Features of CISC

- many instruction formats possible
- large instruction set
- many addressing modes available
- uses variable length/multi-operation instructions
- multi-clock cycle instructions
- complex decoding of instructions
- uses complex circuits
- makes frequent use of cache memory
- uses programmable control unit // uses micro-programmed control unit // uses hardwired control unit
- hardware needs to be able to handle more complex instructions convert into sub-instructions // Design emphasis is on the hardware.

Zain Merchant

Process of interrupt handling

- Once the processor detects an interrupt at the start/end of the fetch-execute cycle
- the current program is temporarily stopped and the status of each register stored on the stack.
- After the interrupt has been serviced/the Interrupt Service Routine (ISR) has been executed ...
- the registers can be restored to its original status before the interrupt was detected // ... the data can be restored from the stack.

How pipelining affects interrupt handling for RISC

- Pipelining adds an additional complexity // there could be a number of instructions still in the pipeline when the interrupt is received
- All the instructions currently in operation are usually discarded except for the last one/ the one at write back
- the interrupt handler routine is applied to the remaining instruction.
- Once the interrupt has been serviced the processor can restart with the next instruction in the sequence.

Virtual machine

- The emulation of a computer system / hardware and/or software using a host computer system.
- Using guest operating system(s) for emulation.

Benefits of virtual machines

- **COMPATIBILITY** e.g. Applications that aren't compatible with the host computer can be run on the virtual machine // It is possible to emulate old software on a new system by running a compatible guest operating system as a virtual machine // Software can be tried on different OS on the same hardware.

- **PROTECTION** e.g. The guest operating system has no effect on anything outside the virtual machine other virtual machines or the host computer//Virtual machines are useful for testing as they will not crash the host computer if something goes wrong // Easier to recover if software causes a system crash as virtual machine software protects the host system.

- **COST** e.g. No need to buy extra computers / hardware as multiple virtual machines can be implemented on the same hardware

Drawbacks of virtual machines

- **PERFORMANCE** e.g. The performance of the guest operating system will not be as good on a virtual machine as it would be on its own compatible machine because of the extra code / using more RAM/memory space // The performance of the VM is dependent on the capabilities of the host computer // Response times cannot be accurately measured using a virtual machine.

- **COMPLEXITY** e.g. Building an in-house virtual machine can be expensive, time consuming and complex to maintain / set-up.

- **HARDWARE/SOFTWARE ISSUES** e.g. Some hardware/software can't be emulated with a virtual machine // Some of the host machine's hardware can't be directly accessed by the virtual machine.

Zain Merchant

Explain the role of host operating system

- The host operating system is the normal operating system for the host computer / machine.
- It has control of all the resources of the host computer / machine. // It can access the physical resources of the host computer / machine.
- It provides a user interface to operate the virtual machine software.
- It also runs the virtual machine software.

Explain the role of guest operating system

- The guest operating system runs within the virtual machine.
- it controls the virtual hardware/software during the emulation. // It accesses the actual hardware through the virtual machine and host operating system.
- It provides a virtual user interface for the emulated hardware/software.
- The guest operating system runs under the control of the host operating system.

MISD

- Multiple Instruction, Single Data (architecture) // Performs different operations on the same data stream.
- Each processor works on the same data stream independently.
- Parallel computers **with** multiple processors.

SISD

- Single Instruction, Single Data (architecture). // Data is taken from a
- single source and a single instruction is performed on the data.
- Contains **one** processor, a control unit and a memory unit.
- that executes instructions sequentially.

MIMD

- Multiple Instruction, Multiple Data (architecture). // At any time, any
- processor can execute different instructions on different sets of data.
- Contains **many** processors
- that operate asynchronously / independently.

SIMD

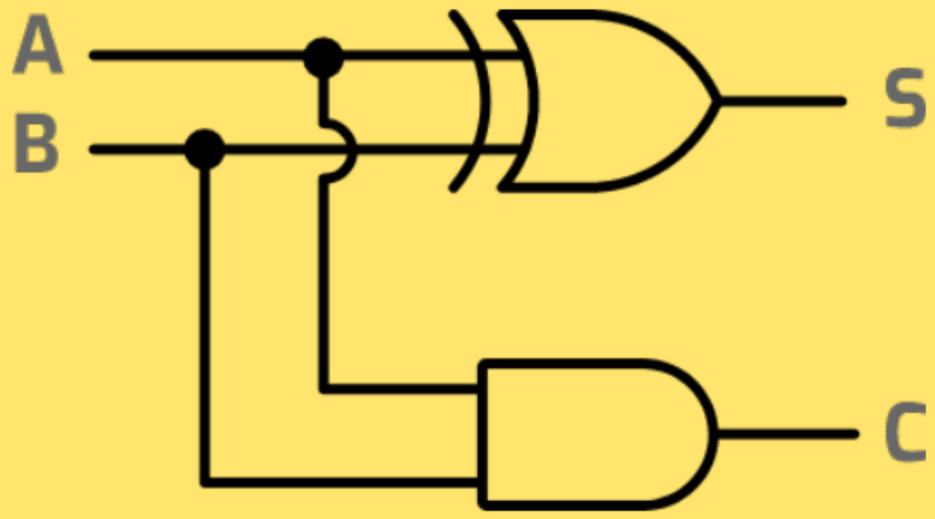
- Single Instruction, Multiple Data (architecture) // Performs the same operation on multiple different data streams simultaneously.
- The instructions can be performed sequentially, taking advantage of pipelining.
- Parallel computers **with** multiple processors.

Boolean algebra and logic circuits

Purpose of a flip-flop

- To store a binary digit / (single) bit.

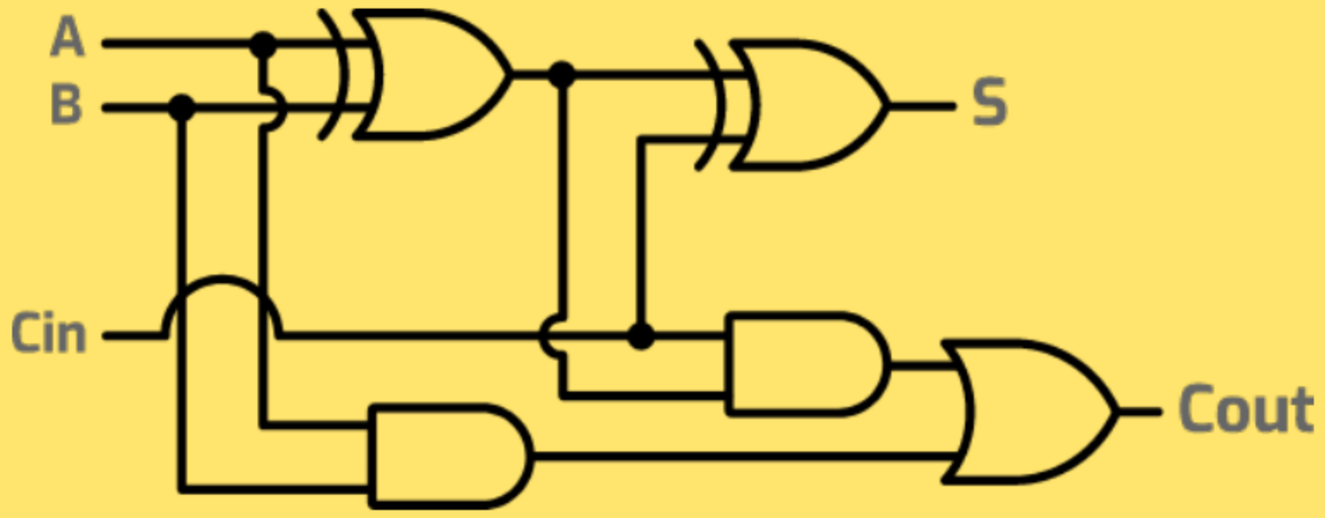
Half adder



Half adder truth table

Inputs		Result	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

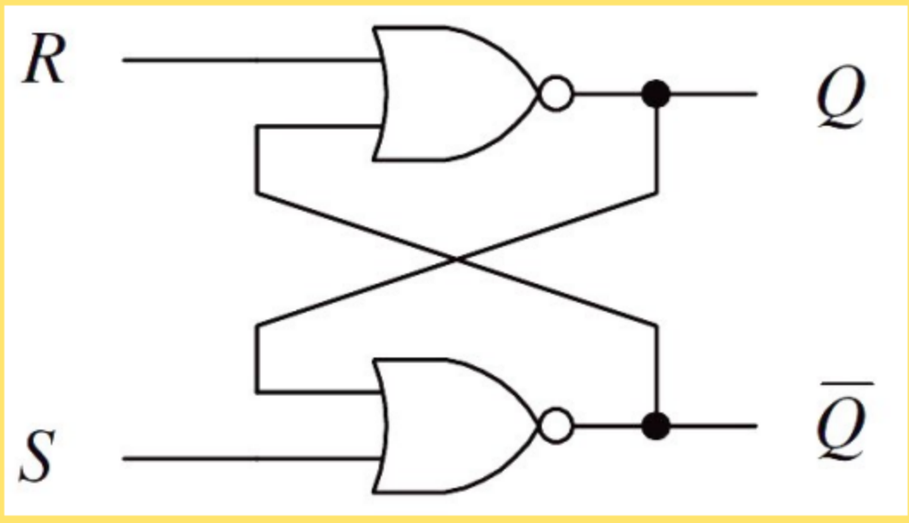
Full adder



Half adder truth table

Inputs			Result	
A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

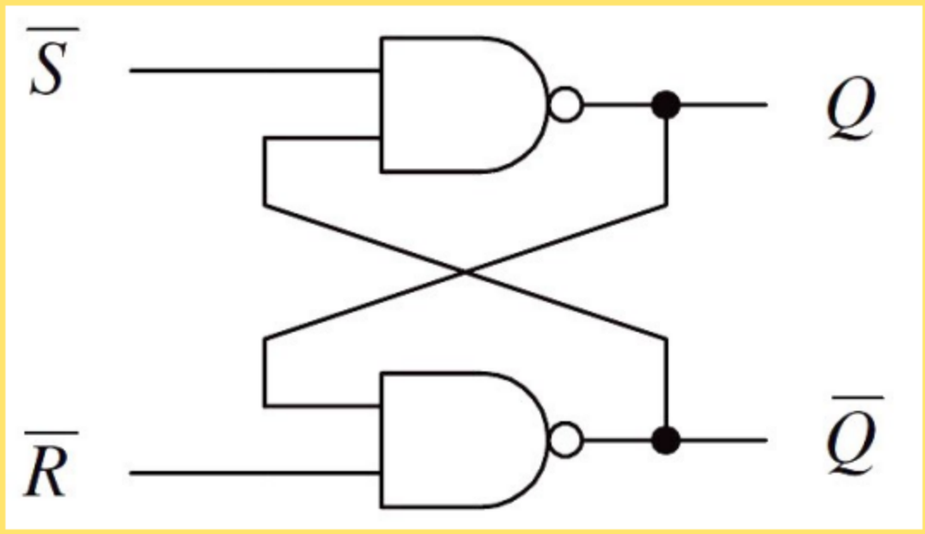
SR flip-flop (NOR gate)



Half adder truth table

Inputs		Outputs	
S	R	Q	
0	0	As previous	
0	1	0	1
1	0	1	0
1	1	Illegal	

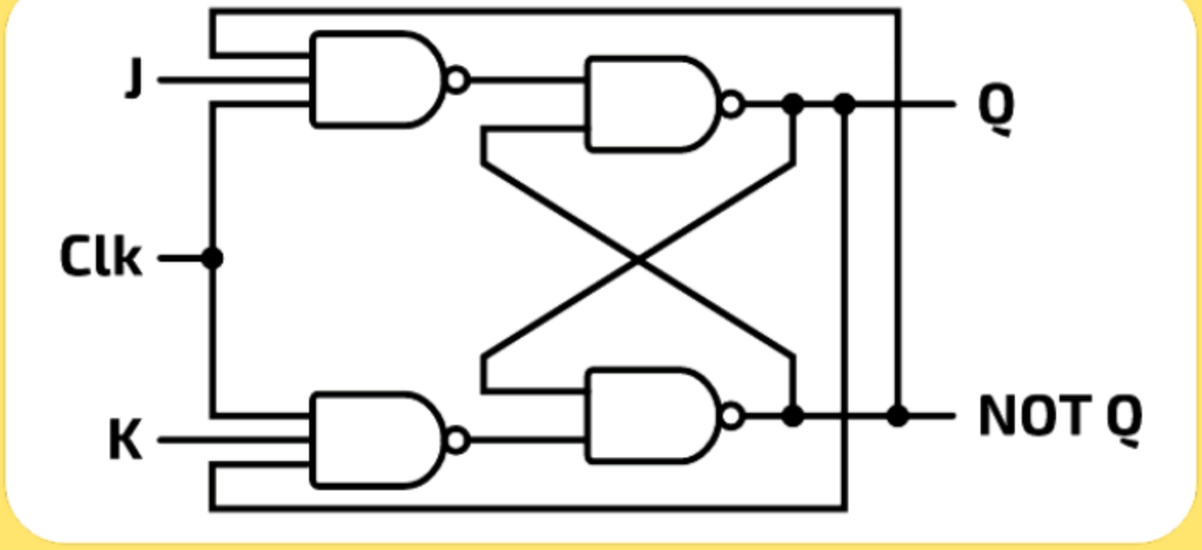
SR flip-flop (NAND gate)



Half adder truth table

Inputs		Outputs	
R	S	Q	Q
1	1	Illegal	
1	0	1	0
0	1	0	1
0	0	As previous	

JK Flip-Flop



Half adder truth table

Inputs			Output	
Clk	J	K	Q	
1	0	0	As previous	
1	0	1	0	1
1	1	0	1	0
1	1	1	Toggle	

De Morgan's law

Equivalences	Further explanation
$\overline{(A \,.\, B)} = (\bar{A}) + (\bar{B})$	NOT(A AND B) is equivalent to (NOT A) OR (NOT B)
$\overline{(A + B)} = (\bar{A}) \,.\, (\bar{B})$	Likewise, NOT(A OR B) is equivalent to (NOT A) AND (NOT B). Notice that when expanding each statement into two brackets, the gate used changes.

System software

Operating systems

Zain Merchant

Segmentation in memory

In segmented memory, the logical / virtual address space is broken into varying sized blocks called segments /

sections.

Each segment has a name and size.

During execution segments from logical / virtual memory are loaded into physical memory.

The address is specified by the user

it contains the segment name and offset value.

Segments are numbered and this number is used as an index in the segment map table.

The offset value determines the size of the segment.

A segment map table maps logical / virtual addresses to physical addresses / contains the segment number and offset.

Disk thrashing

- Disk thrashing is a problem that may occur when virtual memory is being used.
- As the main memory fills up, more and more pages need to be swapped in and out of virtual memory.
- This swapping leads to a very high rate of hard disk access / excessive disk head movements.
- Moving a hard disk read/write head takes a relatively long time / long latency time.
- Eventually, more time is spent swapping pages than processing data thrash point, which can cause the program to freeze or not run.

How kernel acts as an interrupt handler

- the kernel receives a signal when an interrupt is generated
- the kernel checks the priority and reviews the status/priority of the current interrupts
- system enters kernel mode if the type of interrupt is of higher priority than the current process
- the kernel consults the interrupt dispatch table / IDT, and saves the state of the interrupted process / contents of the registers on the kernel stack
- the kernel restores the process state e.g. contents of registers once the interrupt is serviced

Zain Merchant

Multi-tasking

- multi-tasking allows computers to carry out / seem to carry out more than one process at a time

How multi-tasking is implemented

- processor time/common hardware and resources is/are shared between tasks
- scheduling is used to decide on the processes to be carried out to ensure multi-tasking operates correctly / efficiently without clashes
- one task of a higher priority can interrupt another task that is currently running

Benefit of user interface

- The user interface hides the complexities of the computer hardware/operating system from the user
- It provides appropriate access systems for users with differing needs
- Complex commands involving memory locations/buses/computer hardware/ are avoided

Examples of user interface

Clicking on icon rather than writing code

Using a graphical user interface / icons for navigation

Process states

- Ready state
- Running state
- Blocked state

Conditions to change from running to ready state

- When the time slice of the running process expires (round robin).
- and there is a process with a higher priority in the ready queue, the running process is pre-empted
- When an interrupt arrives at the CPU, (the process running on the CPU gets pre-empted).

Why OS may need virtual memory

- Virtual memory is used when RAM is running low
- such as when a computer is running many processes at once.
- Virtual memory may be used for efficient use of RAM / the processor
- such as if data / programs are not immediately needed, they can be moved from RAM to virtual memory

Zain Merchant

Virtual memory

- Disk / secondary storage is used to extend the RAM / memory available
- so the CPU appears to be able to access more memory space than the available RAM
- Only the data in use needs to be in main memory so data can be swapped between RAM and virtual memory as necessary
- Virtual memory is created temporarily.

Paging vs segmentation

- Paging allows the memory to be divided into fixed size blocks and
- Segmentation divides the memory into variable sized blocks.
- The operating system divides the memory into pages, the compiler is responsible for calculating the segment size.
- Access times for paging is faster than for segmentation.

Pipelining during F-E cycle in RISC

- Instructions are divided into subtasks / 5 stages
- Instruction fetch / IF, Instruction decode / ID, operand fetch / OF, opcode/instruction execute IE, result store / write back result / WB
- Each subtask is completed during one clock cycle
- No two instructions can execute their same stage at the same clock cycle
- The second instruction begins in the second clock cycle, while the first instruction has moved on to its second subtask.
- The third instruction begins in the third clock cycle while the first and second instructions move on to their second and third subtasks, respectively, etc.

Zain Merchant

Need for scheduling in process management

- Process scheduling allows more than one program/task to appear to be executed at the same time / enables multi-tasking / multiprogramming.
- To allow high priority jobs to be completed first.
- To keep the CPU busy all the time
- to ensure that all processes execute efficiently
- and to have reduced wait times for all processes / to ensure all processes have fair access to the CPU / prevent starvation of some processes.

Shortest job first scheduling algorithm

- Process are executed in ascending order of the amount of CPU time required // Short processes are executed first **and** followed by longer processes.
- which leads to an increased throughput (because more processes can be executed in a smaller amount of time).

Round robin scheduling algorithm

- Each process is served by the CPU for a fixed time/time slice (so all processes are given the same priority).
- Starvation doesn't occur (because for each round robin cycle, every process is given a fixed time/time slice to execute).

First come first served scheduling algorithm

- No complex logic, each process request is queued as it is received and executed one by one.
- Starvation doesn't occur (because every process will eventually get a chance to run) // less processor overhead.

System software

Translation software

Zain Merchant

How stack can be used to evaluate RPN expressions

- Evaluate the RPN expression from left to right
- Push each element of the RPN expression onto the stack in order until an operator is reached
- Pop the last two elements from the stack and apply the operator
- Push the result of the operation onto the stack
- Repeat the process until the whole expression is evaluated.

Stages in compilation of a program

- Lexical analysis
- Syntax analysis
- Code generation
- Optimisation

Why reverse polish notation (RPN) is used to evaluate expressions

- Reverse Polish Notation provides an unambiguous method of representing an expression
- reading from left to right
- without the need to use brackets
- with no need for rules of precedence / BODMAS

Lexical analysis

converting a sequence of characters into a sequence of tokens

Syntax analysis

- It checks that the code matches the grammar of the language // It checks that the tokens conform with the rules of the programming language
- **Syntax errors** are reported
- A parse tree is produced.

Code generation

converting an intermediate representation of source code into an executable form

Optimisation

minimising a program's execution time and memory requirement

How an interpreter executes a program

- An interpreter examines source code one statement at a time
- Check each statement for errors
- If no error is found the statement is executed
- If an error is found this is reported and the interpreter halts
- Interpretation is repeated for every iteration in repeated sections of code/in loops
- Interpretation has to be repeated every time the program is run

Zain Merchant

Reasons to perform code optimisation

- Produce code that minimises the amount of memory used
- Minimise the execution time

Zain Merchant

Security

Encryption, encryption protocols and digital certificates

Zain Merchant

Digital certificate

- A digital certificate is an electronic/online document.
- used to authenticate/prove the identity of a website/the online identity of an individual/ organisation
- typically issued by a CA
- For example: it contains information identifying a website owner/individual and a public key

Role of digital certificate in creating digital signature

- The digital certificate provides the public key that can be used to validate the private key associated with the organisation/website/digital signature

Asymmetric encryption to send document

- Sheila's computer uses an algorithm to generate a matching pair of keys private and public
- Sheila's computer sends Fred's computer Sheila's public key // Fred's computer acquires Sheila's public key
- Fred's computer encrypts the document/plain text using Sheila's public key to create cipher text
- Fred's computer sends the cipher text to Sheila's computer The cipher text can only be decrypted using Sheila's private key // Sheila's computer uses Sheila's private key to decrypt the cipher text.

Private key

- A private key is the unpublished/secret key/never transmitted anywhere.
- It has a matching public key
- It is used to decrypt data that was encrypted with **its** matching public key.

How digital signature is used to verify a message

- The message together with the digital signature is decrypted using the receiver's private key
- The digital signature received is decrypted with the sender's public key to recover the message digest sent
- The decrypted message received is hashed with the agreed hashing algorithm to reproduce the message digest of the message received
- The two message digests are compared
- if **both digests** are the same the message has **not** been altered // if they are different the message has been altered.

How a digital signature is produced

- The message is hashed with (the agreed hashing algorithm)
- to produce a message digest
- The message digest is then encrypted with the sender's private key to form the digital signature

Zain Merchant

Symmetric vs asymmetric encryption

- Symmetric encryption uses a single key and asymmetric encryption uses a pair of keys.
- The symmetric single key is used by all, whereas only one of the keys for asymmetric encryption is available to everyone / one of the asymmetric encryption keys needs to be kept secret.

Process of acquiring digital certificate

- The organisation requests a certificate from a Certificate Authority (CA)
- The organisation may send their public key to CA
- The organisation gathers all the information required by the CA in order to obtain their certificate, which includes information to prove their identity
- The CA verifies the organisation's identity
- The CA generates / issues the certificate including the organisation's public key (and other information).

Reasons for using key cryptography

- To ensure the message is authentic // came from a trusted source
- To ensure that only the intended receiver is able to understand the message
- To ensure the message has not been altered during transmission
- Non-repudiation, neither the sender or receiver can deny the transmission occurred

Benefits of quantum cryptography

- Any eavesdropping can be identified (as the state will be changed)
- Integrity of the key once transferred can be guaranteed (cannot be copied and decrypted at a later date)
- Longer/more secure keys can be exchanged

Drawbacks of quantum cryptography

- Limited range
- requires dedicated fibre (optic) line and specialist hardware
- cost of dedicated fibre (optic) line and specialist hardware is expensive
- polarisation of light may be altered whilst travelling down fibre optic cables

Zain Merchant

Purpose of secure sockets layer (SSL) and transport layer security (TLS)

- The SSL and TLS protocols provide communications security over the internet / network
- they provide encryption
- They enable two parties to identify and authenticate each other
- and communicate with confidentiality and integrity.

How SSL/TLS protocols are used

- An SSL/TLS connection is initiated by an application which becomes the client
- The application which receives the connection becomes the server
- Every new session begins with a handshake (as defined by the (SSL/TLS) protocols)
- The client requests the digital certificate from the server // the server sends the digital certificate to the client
- The client verifies the server's digital certificate and obtains the server's public key
- The encryption algorithms are agreed
- The symmetric session keys are generated / defined

Zain Merchant

Artificial intelligence

AI concepts

Zain Merchant

Deep learning

- Deep learning learns by finding hidden patterns that are undetectable to humans.
- It structures algorithms in layers: input layer, hidden layers and output layer.
- to create an artificial neural network to learn and make intelligent decisions on its own.
- It is trained using large quantities of unlabelled data.
- Deep learning requires/uses a large number of hidden layers.
- the larger the number of layers, the higher the level of success.

Reinforcement learning

- Reinforcement learning is a machine learning technique based on feedback / rewards / punishment.
- in which an agent learns to behave in an environment by performing the actions and seeing the results of the actions.
- for each good action, the agent gets positive feedback / reward and each bad action receives negative feedback / punishment.
- The agent learns automatically using feedback without any labelled data / specific instructions.
- Adjust node weightings to achieve the correct outcome. // Using feedback to improve its performance at accomplishing similar tasks.

Artificial neural network

- An artificial neural network is the component of artificial intelligence that is meant to simulate the functioning of a biological brain.
- Artificial neural networks are a key component of machine learning.
- They can solve problems that would prove impossible or difficult for humans // Artificial neural networks have self-learning capabilities that enable them to produce better results as more data becomes available
- Artificial neural networks can be layered (input, hidden and output layers) // Artificial neural networks have many interconnected layers, some / many of which are hidden
- Weights are assigned between nodes
- Weights are adjusted through training to give a more accurate result
- More complex learning capabilities / more accurate results are available with larger numbers of hidden layers

Reasons for using deep learning

- Deep learning makes good use of unstructured data.
- Deep learning outperforms other methods if the data size is large.
- Deep learning systems enable machines to process data with a nonlinear approach.
- Deep learning is effective at identifying (hidden) patterns / patterns that humans might not be able to see / patterns that are too complex / time consuming for humans to carry out.
- It can provide a more accurate outcome with higher numbers of hidden layers.

Zain Merchant

Supervised learning

- Supervised learning allows data to be collected, or a data output produced, from the previous experience.
- In supervised learning, known input and associated outputs are given // uses sample data with known outputs (in training) // uses labelled input data.
- Able to predict future outcomes based on past data.

Unsupervised learning

- Unsupervised machine learning helps all kinds of unknown patterns in data to be found.
- Unsupervised learning only requires input data to be given.
- Uses any data // not trained on the right output // uses unlabelled input data.

Reinforcement learning

using a large number of tasks with unknown outcomes and the use of feedback to enable a computer program to improve its performance in accomplishing similar tasks

How artificial neural networks enable machine learning

- Artificial neural networks are intended to replicate the way human brains work
- Weights / values are assigned for each connection between nodes
- The data are input at the input layer and are passed into the system
- They are analysed at each subsequent (hidden) layer where characteristics are extracted / outputs are calculated
- this process of training / learning is repeated many times to achieve optimum outputs // reinforcement learning takes place
- Decisions can be made without being specifically programmed
- The deep learning net will have created complex feature detectors
- The output layer provides the results
- Back propagation (of errors) will be used to correct any errors that have been made.

Reasons for having multiple hidden layers in artificial neural network

- Enables deep learning to take place
- Where the problem you are trying to solve has a higher level of complexity it requires more layers to solve
- To enable the neural network to learn and make decisions on its own
- To improve the accuracy of the result.

Zain Mercha

Artificial neural network

A computer system modelled on a brain.

A* algorithm

A computer method used to find the optimal path between two mapped locations.

Graph

A structure used to model relationships between objects.

Machine learning

A computer program that improves its performance at certain tasks with experience.

Use of graphs to aid AI

- Artificial Neural Networks can be represented using graphs
- Graphs provide structures for relationships // graphs provide relationships between nodes
- AI problems can be defined/solved as finding a path in a graph
- Graphs may be analysed/ingested by a range of algorithms
    - e.g. A* / Dijksta's algorithm
    - used in machine learning.
- Example of method e.g. Back propagation of errors / regression methods

Zain Merchant

Algorithms

Search for an element in the array (BinTree)

```
FUNCTION SearchTree(Item : STRING) RETURNS INTEGER
     NowPtr <- RootPtr
          WHILE NowPtr <> -1
               IF BinTree[NowPtr].Data > Item THEN
                    NowPtr <- BinTree[NowPtr].LeftPtr
               ELSE
                    IF BinTree[NowPtr].Data < Item THEN
                         NowPtr <- BinTree[NowPtr].RightPtr
                    ELSE
                         RETURN NowPtr
                    ENDIF
               ENDIF
          ENDWHILE
     RETURN NowPtr
ENDFUNCTION
```

Condition for array to use binary search

The elements are sorted according to the compare function / in ascending / descending order.

Binary search

```
DECLARE Names : ARRAY[1:100000] OF STRING
DECLARE TopOfList : INTEGER
DECLARE EndOfList : INTEGER
DECLARE CurrentItem : INTEGER
DECLARE ToFind : STRING
DECLARE Found : BOOLEAN
DECLARE NotInList : BOOLEAN
TopOfList <— 1
EndOfList <— 100000
OUTPUT "Which name do you wish to find? "
INPUT ToFind
Found <— FALSE
NotInList <— FALSE
WHILE Found = FALSE AND NotInList = FALSE
     CurrentItem <- (TopOfList + EndOfList) DIV 2
     IF ToFind = Names[CurrentItem]// Names[CurrentItem] = ToFind
     THEN
          Found <— TRUE
     ELSE
          IF TopOfList >= EndOfList THEN
               NotInList <— TRUE
          ELSE
               IF ToFind > Names[CurrentItem] THEN
                    TopOfList <— CurrentItem + 1
               ELSE
                    EndOfList <— CurrentItem - 1
               ENDIF
          ENDIF
     ENDIF
ENDWHILE
IF Found = TRUE THEN
     OUTPUT "Item found at position ", CurrentItem, " in array"
ELSE
     OUTPUT "Item not in array"
ENDIF
```

Performance of binary search using Big O notation

- Big O for a binary search is $O(Log_2 n)$.
- Big O notation is used to indicate the time/space complexity of an algorithm.
- The time taken to complete the search increases logarithmically as the number of search items increases linearly
- The time taken to complete the search increases linearly as the number of search items increases exponentially
- As the search field is repeatedly getting smaller, the number of comparisons made before the item is found, or the number of items runs out, is relatively small.

Big O notation for time complexity of linear search

O(n)

Meaning of O(log n) for binary search

- O(log n) is a time complexity that uses logarithmic time.
- The time taken goes up linearly as the number of items rises exponentially
- O(log n) is the worst case scenario (time complexity for a binary search).

Linear search

```
DECLARE Widgets : ARRAY[1:50000] OF STRING
DECLARE TopOfList : INTEGER
DECLARE EndOfList : INTEGER
DECLARE Count : INTEGER DECLARE ToFind : STRING
DECLARE Found : BOOLEAN
DECLARE NotInList : BOOLEAN
TopOfList <— 1
EndOfList <— 50000
OUTPUT "Enter the name of the item you wish to find "
INPUT ToFind
Found <— FALSE
NotInList <— FALSE
Count <— TopOfList
WHILE Found = FALSE AND NotInList = FALSE // Count <= EndOfList
     IF ToFind = Widgets[Count] // Widgets[Count] = ToFind THEN
          Found <— TRUE
     ENDIF
     Count <— Count + 1
     IF Found = FALSE AND Count > EndOfList THEN
          NotInList <— TRUE
     ENDIF
ENDWHILE
IF Found = TRUE THEN
     OUTPUT "Item found at position ", Count - 1, " in array"
ELSE
     OUTPUT "Item not in array"
ENDIF
```

Zain Merchant

Compare using Big O notation the linear and binary search algorithms

- Linear search sequentially checks each element of the array / list.
- until the matching element is found, or the end of the array / list is reached.
- Binary search finds the mid-point of an array/list and determines which side contains the item to be found
- it discards the half of the array/list not containing the search item // ... it finds the position of a target value within an array / list by repeatedly halving the target search field.
- The binary search requires the elements to be sorted // The linear search does not require the elements to be sorted.
- The binary search will usually do many fewer comparisons of records/iterations against the target than a linear search before it finds its target.
- Linear search starts at the beginning of the array/list and binary search starts in the middle of the array/list.
- Big O for binary search is O (Log_2 n)
- Big O for linear search is O (n)
- Big O notation is used to indicate the time / space complexity of an algorithm

Implementation of stack and pop function

```
CONSTANT Capacity = 25
DECLARE BasePointer : INTEGER
DECLARE TopPointer : INTEGER
DECLARE Stack : ARRAY[1:25] OF REAL

// popping an item from the stack
FUNCTION Pop() RETURNS REAL
     DECLARE Item : REAL
     Item <- 0
          IF TopPointer >= BasePointer THEN
               Item <- Stack[TopPointer]
               TopPointer <- TopPointer - 1
          ELSE
               OUTPUT "The stack is empty - error"
          ENDIF
     RETURN Item
ENDFUNCTION
```

Initialisation of stack and push function

```
// initialisation of stack
PROCEDURE Initialise()
     BasePointer <- 1
     TopPointer <- 0
ENDPROCEDURE

// adding an item to the stack
PROCEDURE Push(NewItem)
     IF TopPointer < Capacity THEN
          TopPointer <- TopPointer + 1
          Stack[TopPointer] <- NewItem
     ENDIF
ENDPROCEDURE
```

Zain Merchant

Compare and contrast queue and stack

- A queue is a first in first out / FIFO data structure and a stack is a first in last out / FILO / LIFO data structure // Data is removed from a queue in the order it is received and removed from a stack in the reverse order to which it is received
- Both ADTs can vary in size / are of indeterminate length
- Data is popped and pushed (onto/from a stack) at the same end but it is enqueued and dequeued (to/from a queue) at different/opposite ends // a queue has two accessible ends and a stack has only one
- A stack has only one moveable pointer whereas a queue has two.

Use of linked list instead of an array to implement stack

- A linked list is a dynamic data structure / not restricted in size
- Has greater freedom to expand or contract by adding or removing nodes as necessary
- Allows more efficient editing using pointers (instead of moving the data).
- An array is a static data structure generally fixed in size
- When the array is full, the stack cannot be extended any further.

Process of binary search

- Find the middle item / index
- Check the value of middle item in the list to be searched
- If equal item searched for is found
- If this is not equal/greater/less than the item searched for
- discard the half of the list that does not contain the search item
- Repeat the above steps until the item searched for is found
- or there is only one item left in the list and it is not the item searched for // lower bound > / = upper bound

Defining empty binary tree

```
TYPE Node
        DECLARE LeftPointer : INTEGER
        DECLARE RightPointer: INTEGER
        DECLARE Data : STRING
ENDTYPE

DECLARE Tree : ARRAY[0 : 9] OF Node
DECLARE FreePointer : INTEGER
DECLARE RootPointer : INTEGER

PROCEDURE CreateTree()
        DECLARE Index : INTEGER
        RootPointer ← -1
        FreePointer ← 0
        FOR Index ← 0 TO 9 // link nodes
            Tree[Index].LeftPointer ← Index + 1
            Tree[Index].RightPointer ← -1
        NEXT
        Tree[9].LeftPointer ← -1
ENDPROCEDURE
```

Recursive procedure for in-order tree traversal

```
PROCEDURE TraverseTree(BYVALUE Pointer : INTEGER)
     IF Pointer <> -1 THEN
            TraverseTree(Tree[Pointer].LeftPointer)
            OUTPUT Tree[Pointer].Data
            TraverseTree(Tree[Pointer].RightPointer)
     ENDIF
ENDPROCEDURE
```

Adding an item to binary tree

```
PROCEDURE AddToTree(ByVal NewDataItem : STRING)
// if no free node report an error
     IF FreePointer = -1 THEN
          ERROR("No free space left")
     ELSE // add new data item to first node in the free list
          NewNodePointer ← FreePointer
          Tree[NewNodePointer].Data ← NewDataItem
          // adjust free pointer
          FreePointer ← Tree[FreePointer].LeftPointer
          // clear left pointer
          Tree[NewNodePointer].LeftPointer ← -1
          // is tree currently empty?

          IF RootPointer = -1 THEN
          // make new node the root node
               RootPointer ← NewNodePointer
          ELSE // find position where new node is to be added
               Index ← RootPointer
               CALL FindInsertionPoint(NewDataItem, Index,
               Direction)

               IF Direction = “Left” THEN
               // add new node on left
                    Tree[Index].LeftPointer ← NewNodePointer
               ELSE // add new node on right
                    Tree[Index].RightPointer ← NewNodePointer
               ENDIF

          ENDIF

     ENDIF
ENDPROCEDURE
```

Recursion

Zain Merchant

How a compiler makes use of a stack when translating recursive code

- The compiler must produce object code to
    - push return addresses / values of local variables onto a stack
    - with each recursive call // ... to set up winding
    - pop return addresses / values of local variables off the stack ...
    - after the base case is reached // ... to implement unwinding.

Essential features of recursion

- Must have a base case/stopping condition
- Must have a general case
- which calls itself (recursively) // Defined in terms of itself
- which changes its state and moves towards the base case Unwinding can occur once the base case is reached.

Reasons to use stack to implement recursion

- A stack is a LIFO data structure
- Each recursive call is pushed onto the stack
- and is then popped as the function ends
- Enables backtracking/unwinding to maintain the required order.

Further programming

Programming paradigms

Reason to declare private attribute in OOP

- To ensure that the attributes are only accessible using the class's own methods/within the class.

Instance

- an occurrence of an object // a specific object based on the class // an instantiation of a class.

Inheritance

the capability of defining a new class of objects that has all the attributes and methods from a parent class.

Polymorphism

allows the same method to take on different behaviours depending on which class is instantiated // methods can be redefined for derived classes.

Encapsulation

putting properties and methods inside a class // ensures sensitive data is hidden from users by hiding values of a structured data object inside a class.

Getter

method that is used to return the value of a property.

Setter

method that is used to update the value of a property.

Structure of a class

- Attributes/properties: with their data types // are variables bound to the class
- Methods: that are subroutines / functions / procedures that act upon the attributes
- Getters/setters: are methods that can fetch / update the contents of attributes
- A constructor: that is used to create instances / objects of this class.

Differences between an object and a class

- A class is only defined once but many objects can be created from that class // A class is a template / blueprint from which objects are created // An object is an instance of a class
- No memory is allocated when a class is defined, but objects are allocated memory space whenever they are created
- A class cannot be manipulated as it is not available in the memory, but objects can be manipulated
- A class is defined but an object is declared / created / instantiated.
- A class can use inheritance. An object cannot.

Methods and properties (Pseudocode)

```
PRIVATE Attempts : INTEGER
Attempts ← 3

PUBLIC PROCEDURE SetAttempts(Number : INTEGER)
     Attempts ← Number
ENDPROCEDURE

PRIVATE FUNCTION GetAttempts() RETURNS INTEGER
     RETURN Attempts
ENDFUNCTION

Player.SetAttempts(5)
OUTPUT Player.GetAttempts()
```

Constructor (Pseudocode)

```
CLASS Pet
     PRIVATE Name : STRING
     PUBLIC PROCEDURE NEW(GivenName : STRING)
          Name ← GivenName
     ENDPROCEDURE
ENDCLASS
```

Inheritance (Pseudocode)

```
CLASS Cat INHERITS Pet
     PRIVATE Breed: INTEGER
     PUBLIC PROCEDURE NEW(GivenName : STRING, GivenBreed : STRING)
          SUPER.NEW(GivenName)
          Breed ← GivenBreed
     ENDPROCEDURE
ENDCLASS
```

Object creation (Pseudocode)

```
<object name> ← NEW <class name>(<param1>, <param2> ...)
```

For example:

```
MyCat ← NEW Cat("Kitty", "Shorthaired")
```

Properties

- the data types // characteristics
- the data items / attributes
- defined in a class

Methods

- the procedures/ functions / programmed instructions in a
- ... that act on the properties / attributes
- class / super class / base class
- ... implementing the behaviours

Low-level programming paradigm

Programs using the instruction set of a processor

Object-oriented programming paradigm

Programs using the concepts of class, inheritance, encapsulation and polymorphism

Zain Merchant

Declarative programming paradigm

- Instructs a program on what needs to be done instead of how to do it
    - using facts and rules
    - using queries to satisfy goals.
- Can be logical or functional
- Logical - states a program as a set of logical relations
- Functional – constructed by applying functions to arguments / uses a mathematical style

Imperative programming paradigm

- Imperative languages use variables
- which are changed using (assignment) statements
- they rely on a method of repetition / iteration.
- The statements provide a sequence of commands for the computer to perform in the order written / given
- each line of code changes something in the program run.

Zain Merchant

Further programming

File processing and exception handling

Find customer account record in random file and output

```
TYPE TAccount
    DECLARE AccountNumber : INTEGER
    DECLARE LastName : STRING
    DECLARE FirstName : STRING
    DECLARE Address : STRING
    DECLARE ContactNumber : STRING
ENDTYPE

DECLARE Customer : TAccount
DECLARE Location : INTEGER
DECLARE AccountFile : STRING
AccountFile <— "AccountRecords.dat"
OPENFILE AccountFile FOR RANDOM
OUTPUT "Please enter an account number"
INPUT Customer.AccountNumber
Location <— Hash(Customer.AccountNumber)
SEEK AccountFile, Location
GETRECORD AccountFile, Customer
OUTPUT Customer
CLOSEFILE AccountFile
```

Exception handling

- (Exception handling is the process of) responding to an unexpected **event** when the program is running so it does not halt unexpectedly

Possible causes of an exception

- Programming errors
- User errors
- Hardware failure
- Runtime errors

Exception

- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions / causes the program to halt execution

Find customer account record in random file with closed hash

```
TYPE TAccount
            DECLARE AccountNumber : INTEGER
            DECLARE Name : STRING
            DECLARE Address : STRING
            DECLARE Telephone : STRING
ENDTYPE

DECLARE Customer : TAccount
DECLARE Location : INTEGER
DECLARE MaxSize : INTEGER
DECLARE: FoundFlag : BOOLEAN
DECLARE SearchCustomer : STRING

MaxSize <— 1000
OPENFILE "AccountRecord.dat" FOR RANDOM
Location <— 1
FoundFlag <— FALSE
OUTPUT "Enter the customer’s name"
INPUT SearchCustomer

WHILE NOT FoundFlag AND Location <= MaxSize
     SEEK "AccountRecord.dat", Location
     GETRECORD "AccountRecord.dat", Customer
     IF SearchCustomer = Customer.Name THEN
          OUTPUT "Customer found: "
          OUTPUT Customer
          FoundFlag <— TRUE
     ENDIF
     Location <— Location + 1
ENDWHILE

IF NOT FoundFlag THEN
     OUTPUT "Customer does not exist."
ENDIF
```

Handling random files (Pseudocode)

Random files are opened using the RANDOM file mode as follows:

```
OPENFILE <file identifier> FOR RANDOM
```

As with text files, the file identifier will normally be the name of the file.

The SEEK command moves the file pointer to a given location:

```
SEEK <file identifier>, <address>
```

The address should be an expression that evaluates to an integer which indicates the location of a record to be read or written. This is usually the number of records from the beginning of the file. It is good practice to explain how the addresses are computed.

The command GETRECORD should be used to read the record at the file pointer:

```
GETRECORD <file identifier>, <variable>
```

The command `PUTRECORD` is used to write a record into the file at the file pointer:

```
PUTRECORD <file identifier>, <variable>
```

When this command is executed, the data in the variable is inserted into the record at the file pointer. Any data that was previously at this location will be replaced.

Handling random files (Pseudocode example)

The records from positions 10 to 20 of a file `StudentFile.Dat` are moved to the next position and a new record is inserted into position 10. The example uses the user-defined type `Student` defined on page 6

```
DECLARE Pupil : Student
DECLARE NewPupil : Student
DECLARE Position : INTEGER

NewPupil.LastName ← "Johnson"
NewPupil.Firstname ← "Leroy"
NewPupil.DateOfBirth ← 02/01/2005
NewPupil.YearGroup ← 6
NewPupil.FormGroup ← 'A'

OPENFILE "StudentFile.Dat" FOR RANDOM
FOR Position ← 20 TO 10 STEP -1
     SEEK "StudentFile.Dat", Position
     GETRECORD "StudentFile.Dat", Pupil
     SEEK "StudentFile.Dat", Position + 1
     PUTRECORD "StudentFile.Dat", Pupil
NEXT Position

SEEK "StudentFile.Dat", 10
PUTRECORD "StudentFile.Dat", NewPupil
CLOSEFILE "StudentFile.dat"
```