4. SOFTWARE

Types of software

SYSTEM SOFTWARE	APPLICATION SOFTWARE
Provides services that computer requires.	Provides services that the user requires.
Examples: - Utility software (eg. defragmentation software, antivirus, firewall, screensaver) - Operating system	Examples: - Word processor - Web browser - Photo/video editing software - Gameware
Runs when system is turned on and stops running when system is turned off.	Runs as per the user's request.
Smaller; requires less storage space.	Requires more storage space.
Complex; performs variety of tasks.	Only performs a single task.

NOTE:

- Application software runs directly on the operating system.
- Component in the computer that would store both types of software when the power is turned off: Secondary storage // HDD // SSD

Operating System

- A program designed to run other programs on a computer.
- Provides an environment in which applications can run and also provides an interface between computer & human operator.
- Backbone of computer: manages both software and hardware resources.
- Examples: Windows, MAC, Linux

NOTE:

- Most computers store OS on a HDD or SSD, since they tend to be large programs.
- Mobile phones and tablets store OS on SSD (as HDD is too large).

Functions

Provides user interface (Human computer interface / HCI)	- Allows user to interact with OS. - Converts user input to form that computer can understand and vice versa. - Should be easy to use
Manages user accounts	- Each user is provided with account for access to system.

	- Accounts accessed using username + password - Users granted different levels of access - Monitor login activity & log users out if they have been inactive for a while
Manages security software/ provides system security	- Creates/deletes users for system - Provides access level rights - Protects from viruses/worms/malware - Regular security updates
Manages file handling/ file management	- Create/save/open/close/move/copy/ rename/deletes files - Sort files - controls file permissions: ability to open/view
Manages hardware/peripherals/drivers	- Handles all devices connected to computer: input devices (keyboard & mouse) + output devices (monitor & printer) - Communicates with devices through drivers Driver - Software that translate instructions from computer so devices can understand - Allows devices to communicate with the computer
Manages multitasking	- Allows tasks to be completed at same time. - Uses time slicing: splits different tasks into small segments. - Tasks can be run one after the other: to seem like multiple tasks are completed at same time.
Handles interrupts	
Memory management	- Keeps track of status of each memory location - Manages movement of data to & from RAM - Checks that processes have enough memory located to them - Makes sure that 2 processes don't try to access same memory location - Manages transfer of pages between virtual memory & RAM - Allows multitasking
Platform for running application software	- Application programs & hardware communicate through system within OS called application programs interface(API) - OS allocates memory space & controls application's data, devices & user access

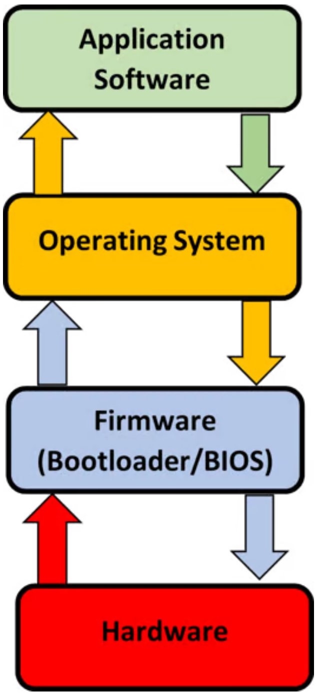
Why PC requires an operating system

- It performs a number of basic tasks, including controlling hardware, file handling.
- It allows the user to communicate with the computer using hardware.
- It provides the user with a user interface.
- PC's are often used to perform many complex tasks at a time.
- An OS is needed to handle this multitasking.
- Therefore, it also provides the ability to handle interrupts.

Describe purpose of operating system

- It performs the basic functions of a computer
- It manages the hardware
- It provides a platform to run software
- It provides a user interface
- It performs tasks such as (any example of function of an operating system)

Hardware, Firmware & the OS - Running of applications



- Applications are run on the operating system
- The operating system is run on the firmware
- The bootloader (firmware) is run on the hardware

Example: If you were to type on a word processing document

- Word processor communicates with OS to request key presses.
- OS communicates with firmware to check whether keyboard (hardware) is available.
- Once connection is established with hardware (keyboard) it communicates back through firmware
- Key instructions passed up to OS & finally displayed on word processing software

Part of a computer of which the bootstrap is an example: firmware

Interrupts

- A signal sent from a device/ software to a microprocessor requesting its attention.
- Microprocessor suspends all operations until interrupt has been serviced.
- It causes OS to take specified action.

How interrupt is generated	- Device/software generates interrupt - Interrupt signal sent from device/software to CPU/processor
How interrupt is handled using interrupt service routine	- Interrupt is given priority - Interrupt is queued - Interrupt causes CPU to stop current process - Status of current process & contents of registers is first saved - CPU services the interrupt - Interrupt service routine is used
What happens as a result of interrupts	- Interrupt will be executed instead of original instructions - Once interrupt is serviced, status of interrupted process is restored - Previous process continues - Message is displayed/output to user

When a user is reading a text on the mobile phone, they may also get a telephone call on the mobile phone. An interrupt signal is generated that results in an output to inform the user that a person is calling them. Describe how the interrupt signal is processed to inform the user that a person is calling them.

- The interrupt signal is sent to the CPU/processor
- The CPU stops the task it is currently processing …
- … to service the interrupt
- An interrupt service routine is used (to service the interrupt)
- Once the interrupt is serviced, a message is displayed to notify the user of the call

Role of interrupt in generating message on computer that paper has jammed

- Printer generates interrupt
- Interrupt is given a priority
- Interrupt is queued
- Interrupt stops CPU from processing current task
- CPU will service interrupt // Interrupt handler services interrupt …
- … generating an output message to state there is a paper jam

Why interrupts are needed

- To identify that the processor's attention is required // to stop current process/task
- To allow multitasking
- To allow for efficient processing, by prioritising actions

- To allow for efficient use of hardware
- To allow time-sensitive requests to be dealt with immediately
- To avoid the need to poll devices

What would happen if computer does not use interrupts

- The computer would only start a new task when it had finished processing current task
- Computer will not be able to multitask
- Errors may not be dealt with
- Computer would become impossible to use

Types of interrupt:

- Hardware Interrupt
    - Moving the mouse
    - Clicking a mouse button
    - Plugging in a device
    - Paper jam in printer
    - Printer out of paper
- Software Interrupt
    - Division by zero
    - Two processes accessing the same memory location
    - Null value

Examples of when interrupt is generated

<u>(in printer)</u>

- Paper jam
- Paper tray empty/ runs out of paper
- Runs out of ink
- Buffer requires more data

<u>Software interrupt</u>

- Division by zero
- Two processes trying to access the same memory location

<u>Change of task</u>

- When switching from one application to another
- A peripheral is connected/disconnected
- A phone/video call is received

<u>Hardware interrupt</u>

- A key on a keyboard is pressed
- A mouse button click

NOTE:

- They can be hardware based or software based
- They are handled by the OS.
- They have different levels of priority; don't work out which program to give priority to.
- They allow the computer to multitask or have several windows open at the same time
- They allow multiple functions to co-exist.

Programming languages

Computer program: a list of instructions that enable a computer to perform a specific task.

Syntax: structure of language statements in a computer program.

High-level language	**Low-level language**
Uses English-like statements / close to human language	- Close to the language processed by computers - May use mnemonics - Eg: assembly language/machine code
- Needs to be converted to machine code to be processed by computer - … using a translator	- Machine code doesn't need to be converted - Assembly language require assembler
Portable / machine independent	Machine dependent
Problem / logic focussed	
Advantages 1. Closer to human language - Easier/quick to read/write/understand: can write code in less time - Easier/quicker to debug: can find and correct errors in less time - Less likely to make errors 2. Machine independent/ code is portable - Can be used on many different computers without need for understanding of hardware - Because it is written in source code. - Because it's compiled to object code. 3. Only need to learn a single language - This can be used on many computers. 4. They have built-in functions or libraries - Saves time when writing the program 5. Can use an IDE	Advantages - Directly manipulate hardware - Can use specialised hardware / machine-dependent instructions - Quicker to execute - Smaller file size // less storage space - Program will be more memory efficient: useful when application has high memory consumption - No requirement for program to be portable - No requirement for compiler/interpreter

6. No need to manipulate memory addresses directly - specialised knowledge of this not required 7. Can focus on the problem instead of the manipulation of memory/hardware 8. Easier to maintain 9. One line of code can carry out multiple commands	
Disadvantages - Cannot directly manipulate hardware - Take longer to execute: may need to wait for translation before running - Program may be less efficient - Programs larger	Disadvantages - More difficult to read/write/understand. - Takes longer to write & debug - Not machine independent

Assembly language

- Form of low-level language that uses mnemonics
- Assembler is needed to translate assembly language program to machine code

Drawbacks of assembly language

- Programs are not portable
- It is complex to learn
- Difficult to debug

Translators

	Compiler	**Interpreter**
how program is translated	- Translates high-level language to low-level language/ machine code - Checks/translates all code before it is executed - Creates executable file	- Translates high-level language to low-level - Checks/translates one line of code & executes it before moving onto next line
how errors are reported	- Creates an error report after trying to compile - Displays all errors in code - Errors require correction before executing	- Stops when error found - when corrected, program can be run from same position // allows error correction in real time

Similarities between compiler & interpreter

- Both translate high level language to machine code
- Both generate error report // check for errors

Difference between compiler & interpreter

Interpreter	**Compiler**
translates one line at a time / checks one line and then executes immediately	translates whole code in one go / checks all code before executing
stops & reports when error is encountered // corrects errors in real time	creates error report at the end of translation
run code up to the point it finds an error	will not run code at all if an error is found
does not produce an executable file	produces an executable file
required to run the code each time if used	Not required to run the code each time
Used during program development - Easier to debug - … as errors immediately reported when detected	Used for program distribution - Creates an executable file - … so, translator not required every time - … so would not release source code; source code cannot be stolen/edited - … making it machine independent

Benefits of compiling

- Code will run without the need of an translator
- Code is platform independent.
- Source code not available, therefore cannot be modified

Drawbacks of compiling

- Source code not available, therefore cannot be modified
- Comments, etc. are not visible
- Future changes will require the code to be recompiled

Uses of interpreter & compiler

Why both interpreter and compiler are used together

- To translate high-level language to low-level language.
- Interpreter is used while writing the program.
- Interpreter is used to debug code line-by-line.
- Compiler is used when the program is completed.
- Compiler is used to create a separate executable file (so compiler no longer needed).
- If it runs for the first time in a compiler, there are no syntax errors.

At what point of game creation is it appropriate to use an interpreter

- During development / when writing the program // when debugging
- Easier to debug

- Stops when an error is detected
- Reports one error at a time
- Can correct errors in run-time // correct the line and then continue running from that point
- Can test one section without the rest of the code being completed

<u>At what point of game creation is it appropriate to use a compiler</u>

After completion / for distribution:

- It creates an executable file
- That can be distributed without source codE
- So that other people cannot edit/view the code
- So end users do not need translator software // so end users do not need to compile/interpret each time
- So it is machine/platform independent

For final/ repeated testing:

- It creates an executable file
- Do not need to retranslate for each test sequence
- Can test repeatedly with different data faster

Integrated development environment (IDE)

Software that provides useful functions for a programmer writing a computer program

Features/functions

Code editors	- allows users to write & manipulate source code - includes features like auto-completion & auto-correction, bracket matching, syntax checks
Auto-completion	
Auto-correction	
Run-time environment	allow the program to run and see its corresponding output
Built-in translator (compiler/interpreter)	compiles or interprets the code
Error diagnostics	- Identifying errors: highlights areas of code / provides error messages where error occurred e.g. indentation errors - Debugging errors: Provides step by step instructions of what is happening in each line of code, to catch logical errors
Prettyprinting	- changing font, font size, making text bold - displays keywords in different colours
Commenting	- sections of code commented to explain what it is doing