Diploma Programme
Programme du diplôme
Programa del Diploma

Markscheme

May 2021

Computer science

Higher level

Paper 1

20 pages

Subject details: Computer science HL paper 1 markscheme

Mark allocation

Section A: Candidates are required to answer **all** questions. Total 25 marks.
Section B: Candidates are required to answer **all** questions. Total 75 marks.
Maximum total = 100 marks.

General

A markscheme often has more specific points worthy of a mark than the total allows. This is intentional. Do not award more than the maximum marks allowed for that part of a question.

When deciding upon alternative answers by candidates to those given in the markscheme, consider the following points:

Each statement worth one point has a separate line and the end is signified by means of a semi-colon (;).

An alternative answer or wording is indicated in the markscheme by a "/"; either wording can be accepted.

Words in (…) in the markscheme are not necessary to gain the mark.

If the candidate's answer has the same meaning or can be clearly interpreted as being the same as that in the markscheme then award the mark.

Mark positively. Give candidates credit for what they have achieved and for what they have got correct, rather than penalizing them for what they have not achieved or what they have got wrong.

Remember that many candidates are writing in a second language; be forgiving of minor linguistic slips. In this subject effective communication is more important than grammatical accuracy.

Occasionally, a part of a question may require a calculation whose answer is required for subsequent parts. If an error is made in the first part then it should be penalized. However, if the incorrect answer is used correctly in subsequent parts then **follow through** marks should be awarded. Indicate this with "**FT**".

General guidance

Issue	Guidance
Answering more than the quantity of responses prescribed in the questions	In the case of an "identify" question, read all answers and mark positively up to the maximum marks. Disregard incorrect answers. In the case of a "describe" question, which asks for a certain number of facts *eg* "describe two kinds", mark the first two correct answers. This could include two descriptions, one description and one identification, or two identifications. In the case of an "explain" question, which asks for a specified number of explanations *eg* "explain two reasons …", mark the first two correct answers. This could include two full explanations, one explanation, one partial explanation *etc*.

Section A

1. *Award **[2 max]***
Client;
Server/email server/DNS server/file server;
Router;
Firewall; **[2]**

2. *Award **[2 max]***

Parallel;
old system and new system are operated at the same time until the current system is proved to be successful;

Pilot;
the new (whole) system is operated in one branch/part of the organization before it is rolled out to the whole organization;

Direct;
the new system replaces the old system in an immediate switchover;

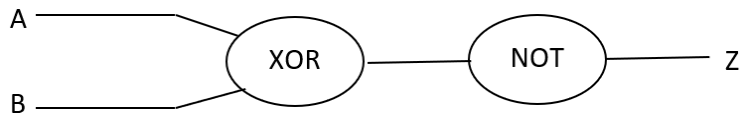
Phased;
the new system in phases / stages, gradually replaces parts of the old system until the current system is completely replaced by the new system;

[2]

3. *Award **[2 max]***

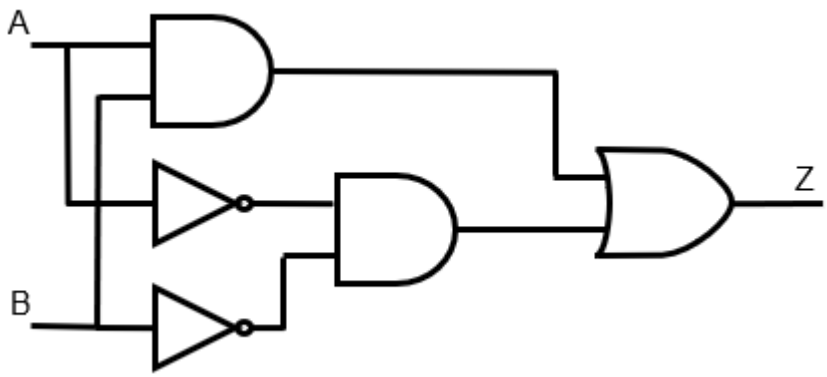
***Note:** There could be many answers that are correct.*

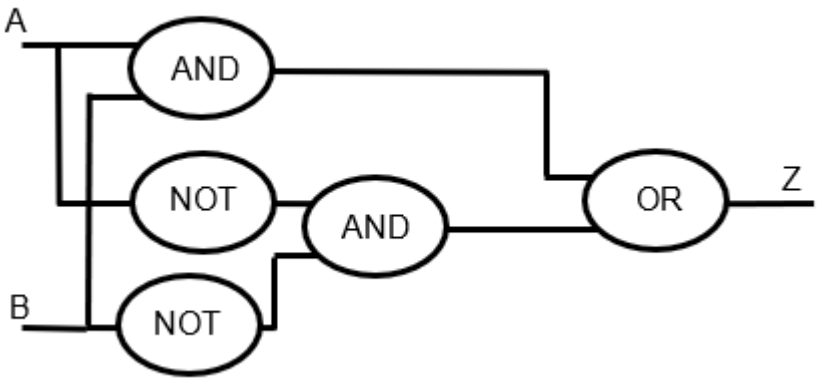
Example 1



Correct inputs and XOR gate;
NOT gate, correct final output and link from XOR gate; **[2]**

Example 2





Award [2] max.

2 marks, 1 mark for each correct input in OR gate
Note:** in this example the two inputs in OR gate are (NOT A AND NOT B)and (A AND B**).
***1** mark for drawing any 3 gates (complete with inputs and outputs)*

4. **(a)** *Award **[2 max]***
Allows bugs/error in operating system to be repaired;
Allows new features to be added to operating system (such as security updates, improving functionality, improving usability, etc.);
Allows compatibility issues to be improved; **[2]**

(b) *Award **[2 max]***
Automatic patches/updates sent (via internet);
User requested updates (via internet);
Patches sent on CD/DVD/memory stick; **[2]**

5. *Award **[2 max]***
11 x 16 + 15;
191;

Allow solution via binary route 1 mark for working, 1 mark for answer.
Allow both marks if correct answer given. **[2]**

6. *Award **[2 max]***
Fibre optics allow faster transmission speeds;
Fibre optic cables are more secure/harder to break into;
Fibre optic cable transmission is more reliable/less likely to suffer interference;
Fibre optics allow transmission over longer distance;
Fibre optics allow greater bandwidth;

[2]

7. *Award **[2 max]***
The value of a variable can change while value of a constant does not change;
During program execution/ during run-time / while stored in the memory;

[2]

8. *Award **[3 max]***
6;
12;
18;

[3]

9. *Award **[1 max]***

Makes maximum use of the available (limited) memory to provide the features needed for the phone;
Does not waste memory space with unwanted/inappropriate features;
Ensures a higher level of security*;*
Makes efficient use of hardware equipment;
Suited to available hardware equipment;

***Note**: Accept examples, GUI will fit right with the screen resolution /can make running the device easier to use / better suited to their audience/ it is faster than universal OS's, etc.*

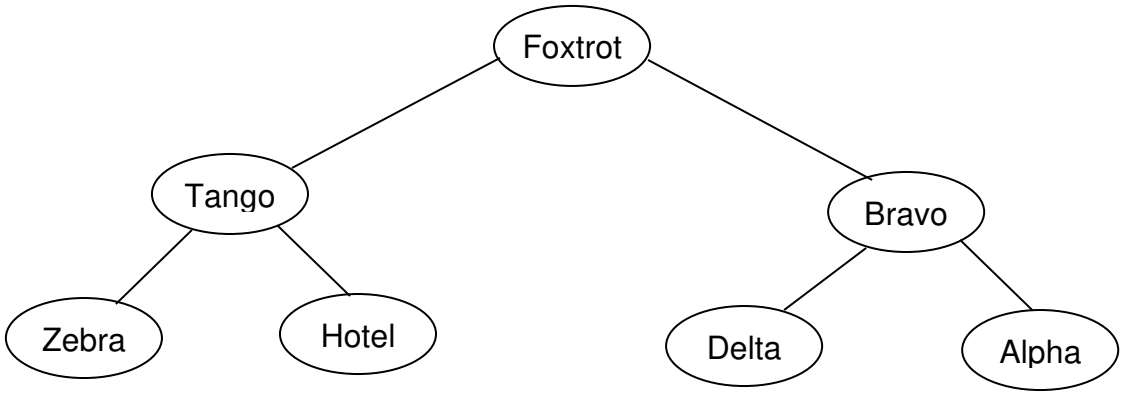
[1]

10. *Award **[2 max]***

Dynamic data structure does not have predetermined size/ allows memory use to change as needed (no fixed size) / if more space is required to store more data, it can therefore increase;
Can be expanded until all the available RAM is used;
There is no unused/wasted memory;
Memory is allocated to the data structure as the program executes (run-time);
Elements of a dynamic data structure are stored in memory locations that are chained together but not necessarily physically contiguous;
Elements of a dynamic data structure are sequentially accessed;

[2]

11.



*Award **[3 max]**:*

Correct root;
Correct left sub-tree;
Correct right sub-tree;
Note: Award 1 mark for any binary tree with the same number of nodes in the left and right subtree;

[3]

Section B

12. (a) *Award **[4 max]***

(Wireless) router;
A central hub for all the computers to connect to;
Enables wireless network packet forwarding and routing;

Wireless Network Interface Card (NIC);
To allow the computer to 'talk to' the (wireless) router;

Wireless access points;
allow Wi-Fi devices to connect to a wired network;

Wireless repeaters;
To expand the reach of the network;

Mark as 2 and 2. **[4]**

(b) *Award **[2 max]***
The ability to use their own devices at school;
The ability to access the school network from anywhere in the school;
No cables laid, so reduces the risk of tripping over cables;
Numbers of connections are not limited to cable ports, so greater numbers of students can connect at any given time;

[2]

(c) *Award **[4 max]***

Use of encryption;
So that data cannot be understood if it is intercepted;

Use of user authentication/usernames and passwords;
To prevent unauthorized access to the system;

Setting up a file of accepted MAC addresses;
To only allow access to the network by registered mobile devices;

Hide network ID;
So that the wireless network is not publicly seen;

Mark as 2 and 2. **[4]**

(d) *Award **[2 max]***
Client VPN software (to make a secure remote connection);
VPN-aware routers and firewalls (to permit VPN traffic to pass);
VPN appliance/server (to handle incoming VPN traffic);
Encryption protocol IPSec or SSL;
[2]

(e) *Award **[3 max]***

Enhanced **s**ecurity of data;
for example, using encryption;
This prevents unauthorised access;

Remote access to data and resources (from any location);
Normal access of materials on the network;
as though the user was using the network on site; **[3]**

13. (a) *Award **[4 max]***
Collection method `NAMES.getNext()` / `NAMES.getData()` correctly used;
Correct loop;
Correct use of index (in both arrays);
Correct assignment in array for surnames;
Correct assignment in array for first names;

Note:
Award 1 mark in case that string methods are used to separate 'name' and 'surname' in the data item.

Example answer 1:

```
NAMES.resetNext() // reset and
SURNAME[600]      //initialization of arrays
FIRSTNAME[600]   //may not appear in candidates' responses
COUNTER = 0
loop while NAMES.hasNext()
    SURNAME[COUNTER] = NAMES.getNext()
    FIRSTNAME[COUNTER] = NAMES.getNext()
COUNTER = COUNTER + 1
end loop
```

Example answer 2

```
NAMES.resetNext()
loop COUNTER from 0 to 599
    SURNAME[COUNTER] = NAMES.getNext()
    FIRSTNAME[COUNTER] = NAMES.getNext()
end loop
```

Example answer 3 (assumes that items in the collection are objects- two attributes: surname and firstname)

```
I = 0
loop while NAMES.hasNext()
        X= NAMES.getNext()
        SURNAME[I] = X.surname
        FIRSTNAME[I] = X.firstname
        I = I + 1
end loop
```

[4]

(b) *Award **[5 max]***
Correct outer loop;
Correct inner loop;
Checking of surname order;
Swapping surnames if necessary;
Swapping corresponding names;
Correct use of flag;

Example 1:

```
loop I from 0 to 599
   loop C from 0 to 598-I  //accept 598
       if SURNAME[C] > SURNAME[C + 1] then
         TEMP1 = SURNAME[C]
         TEMP2 = FIRSTNAME[C]
         SURNAME[C] = SURNAME[C + 1]
         FIRSTNAME[C] = FIRSTNAME[C + 1]
         SURNAME[C + 1] = TEMP1
         FIRSTNAME[C + 1] = TEMP2
       end if
    end loop
end loop
```

Example 2:

```
FLAG = TRUE
loop while FLAG = TRUE
    FLAG = FALSE
    loop COUNTER from 0 to 598
       if SURNAME[COUNTER] > SURNAME[COUNTER + 1] then
         TEMP1 = SURNAME[COUNTER]
         TEMP2 = FIRSTNAME[COUNTER]
         SURNAME[COUNTER] = SURNAME[COUNTER + 1]
         FIRSTNAME[COUNTER] = FIRSTNAME[COUNTER + 1]
         SURNAME[COUNTER + 1] = TEMP1
         FIRSTNAME[COUNTER + 1] = TEMP2
         FLAG = TRUE
       end if
    end loop
end loop
```

[5]

(c) *Award **[4 max]***

Example 1:

Calculate the index of the middle point in the array SURNAME: (first + last)/2;
Compare surname found with the one stored at middle point;
If greater than the value at the middle point, search the upper half of the array (right side) by calling the binary search method again with a new first index (first = middle + 1);
If smaller than the value at the middle point, search the lower half of the array (left side) by calling the binary search method with a new last (last = middle -1);
if found algorithm terminates;

Example 2:

Find the centre point of the array SURNAME[ ];
Compare surname to be found with the current name in SURNAME[ ];
If correct surname found => STOP;
Else if surname to be found is greater than the current name in SURNAME[ ]
eliminate lower half of array from search and repeat algorithm;
Else if surname to be found is less than the name in SURNAME[ ] eliminate upper half of array from search and repeat algorithm;

Note: Allow a mark for provision for name not found;

[4]

(d) *Award **[2 max]***

1 mark for a benefit (such as reusability, modularity, maintainability, readability)
1 mark for an expansion.

Example answer:
These sub-programs (sorting/searching) could be used in (many) other programs;
Which saves programmer's time/ effort;

[2]

14. (a) (i) *Award **[1 max]***

The algorithm will always output zero as the value of LOWEST, for any set of the input values (because all input values are greater than or equal to 0);
The LOWEST is initialized to 0 and negative inputs are not allowed, zero will be the permanently lowest number;

[1]

(ii) *Award **[2 max]***

Example 1:

Replace LOWEST = 0;
With LOWEST=99999999/ any very high number;

Example 2:

Before the loop/ *Accept "at the beginning"*;
The first number NUMBER should be inputted and LOWEST set to NUMBER / LOWEST=input(NUMBER);
Loop only 999 times (because 1 number already inputted)/ condition in the loop should be changed to COUNTER>998;

Example 3:

If statement should be used within the loop;
after the statement 'input NUMBER';
For example, if COUNTER==0 then LOWEST = NUMBER/ set LOWEST to the first inputted number;

[2]

(b) *Award **[1 max]***
Validation (check);
Data type check;
Range check; **[1]**

(c) *Award **[8 max]**:*

Correct initialization of HIGHEST, LOWEST and TOTAL;
Correct use of loop for 1000 inputs (doesn't have to be while loop);
Checking if NUMBER is between 0 to 1000(accept inclusive or exclusive range);
Checking if NUMBER is a whole number;
Running total of numbers (for average calculation);
Correct comparison and changing the value of HIGHEST if needed;
Correct comparison and changing the value of LOWEST if needed;
Appropriate calculation of AVERAGE;
Output of AVERAGE, HIGHEST and LOWEST (outside loop);

Example answer 1

```
HIGHEST = -1    //any value lesser than or equal to 0
LOWEST = 10000  //any value greater than or equal to 1000
COUNTER = 0
TOTAL = 0
loop while COUNTER < 1000 // allow COUNTER <= 999
    input NUMBER
    if NUMBER<0 or NUMBER>1000 or NUMBER div 1 ≠ NUMBER/1
                  //accept NUMBER<1 or NUMBER>999 or NUMBER mod 1 ≠ 0
               output " Your number is invalid, please try again"
    else
           if NUMBER > HIGHEST then
             HIGHEST = NUMBER
           end if
           if NUMBER < LOWEST then
             LOWEST = NUMBER
           end if
           TOTAL = TOTAL + NUMBER
           COUNTER = COUNTER + 1
     endif
end loop
AVERAGE = TOTAL / COUNTER       // allow TOTAL / 1000
output HIGHEST
output LOWEST
output AVERAGE
```

Example answer 2: (the lowest and the highest are set to the value of the first inputted number)

```
input NUMBER
 loop while NUMBER<0 or NUMBER>1000 or NUMBER div 1 ≠ NUMBER/1
      output " Your number is invalid, please try again"
      input NUMBER
end loop
HIGHEST = NUMBER
LOWEST = NUMBER
TOTAL = 0
loop COUNTER FROM 0 to 998
            input NUMBER
            loop while NUMBER<0 or NUMBER>1000 or NUMBER div 1 ≠ NUMBER/1
                output " Your number is invalid, please try again"
                input NUMBER
            end loop
            if NUMBER > HIGHEST then
              HIGHEST = NUMBER
            end if
            if NUMBER < LOWEST then
              LOWEST = NUMBER
            end if
            TOTAL = TOTAL + NUMBER

  end loop
  AVERAGE = TOTAL / 1000
  output HIGHEST, LOWEST, AVERAGE
```

Example answer 3

```
HIGHEST = -1    //any value lesser than or equal to 0
LOWEST = 10000  //any value greater than or equal to 1000
C = 0
TOTAL = 0
loop while C<1000
    input NUMBER
    if NUMBER>=0 and NUMBER<=1000
        if NUMBER mod 1 == 0
             if NUMBER > HIGHEST then
                  HIGHEST = NUMBER
              else
                 if NUMBER < LOWEST then
                       LOWEST = NUMBER
                 endif
             end if
             TOTAL = TOTAL + NUMBER
             C=C+1
         endif
     else
        output("invalid input, enter another number")
    endif
end loop
output HIGHEST, LOWEST, TOTAL/1000
```

[8]

15. (a) *Award **[4 max]***

Virtual memory;
to enable the hard drive to act as primary memory if required;

Paging;
scheme by which a computer stores and retrieves data from secondary storage for use in main memory;

Multitasking;
to enable more than one application to run at the same time;

Scheduling (method by which work is assigned to resources that complete the work);
to determine which process will own CPU for execution while another process is on hold.;

Policies (ways to choose which activities to perform);
according to which decisions are made about which operations should be authorized/ which resources should be allocated;

Interrupt handling;
so that urgent tasks required by parts of the system may be executed;

Polling;
to sampling the status of an external device;

Mark as 2 and 2. **[4]**

(b) *Award **[2 max]***
Award 1 mark for the virtual machine (presented by the OS to the user, hiding all the complexities of the hardware behind layers of OS software) and 1 mark for an expansion.

The virtualization of real devices;
such as the use of drive letters for partitions to represent a virtual hard drive;

Use of icons;
to represent peripherals/to access different drives; **[2]**

(c) (i) *Award **[2 max]***
The primary memory/secondary memory in a laptop will be less / smaller than would be found on a file server;
because it is a smaller device that is used by an individual, so doesn't need to store / run so many programmes simultaneously; **[2]**

(c) (ii) *Award **[2 max]***
The processor of a file server would be faster/have more cores than that of a laptop;
as it has many more resources to manage/as it needs to keep track of many workstations/printers/peripherals; **[2]**

(d) *Award **[2 max]***
Light sensor;
infra-red/motion sensor; **[2]**

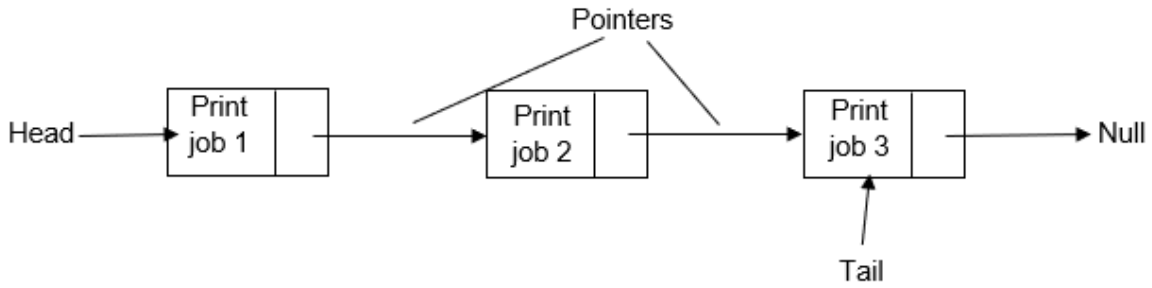
(e) *Award **[4 max]***
The sensors send data to the processor continuously;
The data received from the sensors is converted to digital (using an analogue to digital convertor);
The readings are compared to pre-set values in the controller/system;
If the light level is found to be below the pre-set value **and** a presence is detected (in the room);
A signal is sent to the actuator to turn on the lights; **[4]**

(f) *Award **[2 max]***
A timer (set to 5 minutes);
If the light level is found to be below the pre-set value a timer is activated but the lights remain on;
When the timer reaches 5 minutes, the lights switch off (if no further presence is detected); **[2]**

16. (a) *Award **[2 max]***
Print jobs are to be completed in the order received;
Queue is appropriate because print jobs are added/enqueued at the back/rear and dequeued/removed from the front / because queue is a FIFO data structure;
[2]

(b) *Award **[3 max]***

Correct use of pointers and null pointer in the last node;
External pointer to the beginning of the list (head);
External pointer to the end of the list (tail);
Each node consists of at least 2 parts/fields: data (print jobs) and pointer;



[3]

(c) *Award **[3]***
Print jobs would not be managed in an orderly manner;
Because stack is LIFO data structure / elements are added/pushed and removed/popped at only one end;
And each time another one is added it takes precedence over those that are already in the data structure; **[3]**

(d) *Award **[3 max]***

NUM column correct;
PRODUCT column correct;
OUTPUT column correct;

***Note**: The trace table may be differently presented.*

NUM	PRODUCT	OUTPUT
6		
5		
4		
3		
2		
1		
	1	
2	2	
3	6	
4	24	
5	120	
6	720	
		720

[3]

(e) *Award **[4 max]***

A recursive function calls itself during its execution;
First call (all local variables, data, return addresses, etc.) is pushed onto stack;
Second/subsequent recursive calls are pushed on to the stack (added above previous call(s));
When the terminating condition is met/ execution of recursive function ends;
The function calls pop from the stack;
In the reverse order to which they were pushed/LIFO;

[4]