**1**

**(a) Ans:**  [6]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Conditions** | Age under 25 | Y | Y | Y | Y | N | N | N | N |
| Previous accident | Y | Y | N | N | Y | Y | N | N |
| Licence held for 3 or more years | Y | N | Y | N | Y | N | Y | N |
| **Actions** | 10% extra discount |  | X |  |  |  |  |  |  |
| No discount | X |  |  | X | X | X |  |  |
| 5% discount |  |  | X |  |  |  | X | X |
|  |  | **{1}** | **{1}** | **{1}** | **{1}** | **{1}** | | **{1}** | |  |

**(b) Ans:** [3]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Conditions** | Age under 25 | Y | Y | Y | Y | N | N |  |  |
| Previous accident | Y | Y | N | N | Y | Y |  |  |
| Licence held for 3 or more years | Y | N | Y | N | - | - |  |  |
| **Actions** | 10% extra discount |  | X |  |  |  |  |  |  |
| No discount | X |  |  | X | X |  |  |  |
| 5% discount |  |  | X |  |  | X |  |  |
|  |  | **{1}** | | | | **{1}** | **{1}** |  |  |

**(c) Ans:**  [6]

FUNCTION CostPercentageChange(DriverAge AS INTEGER,

HadAccident AS BOOLEAN,

YearsLicenceHeld AS INTEGER) RETURNS INTEGER:

IF DriverAge >= 25 THEN

IF HadAccident THEN

Mark as follows:

Correct function header **{1}**

Correct IF statement **{1}**

Correct IF statement **{1}**

Correct IF statement **{1}**

Correct IF statement **{1}**

Correct IF statement **{1}**

Correct return statement (or equivalent) [max 6]

OR equivalent demonstrating correct logic

RETURN 0

ELSE

RETURN -5

ENDIF

ELSE

IF HadAccident:

IF YearsLicenceHeld < 3 THEN

RETURN 10

ELSE

RETURN 0

ENDIF

ELSE

IF YearsLicenceHeld < 3 THEN

RETURN 0

ELSE

RETURN -5

ENDIF

ENDIF

ENDIF

**2** **(a) Ans:**  [4]

Have a flag variable that is set to True if a swap is made and reset to False at the start of each pass / the outer loop /

**Or** Have a flag variable that is set to True at the start of each pass to indicate that the list is in order and set to False if a swap is made; **{2}**

change the outer loop so that it would stop repeating if no swaps have been made; **{1}**

After the inner loop; subtract 1 from N; // alter inner loop (for) upper limit;

by subtracting Count1 from N; **{1}**

**(b)** **Ans:**  [2]

Knowledge O(n); **{1}**

Understanding

As the size of the list increases the time taken increases at the same rate;

There is a loop that repeats n times; **{1}**

**(c)** **Ans:** [2]

Knowledge O(log n) **{1}**

Understanding

Each comparison halves the size of the list that has to be searched through;

The time taken increases as the size of the list increases but by smaller and smaller amounts;

If the size of the list doubles then the number of comparisons needed only increases by 1; **{1}**

**3** **(a) (i)** **Ans:** [1]

Faster to lookup a record / search (in most circumstances); A. direct access to record

**Or** Faster to insert a record;

**Or** Faster to delete a record;

**Or** Time complexity is O(1). **{1}**

**(ii)** **Ans:**  [1]

More compact file // no "empty" records in file; **{1}**

**Or** Producing a sorted list/processing the records in order is faster;

**Or** Do not have to design a hash function so easier to design/program // no need to deal with collisions;

**(b)** You may use the space provided for working, if required.

**(i) AE21KWB** [1]

Working /Hash value

Hash value = ( position in alphabet of ‘A’ (1) +

position in alphabet of ‘E’ (5) \* 10 +

2 \* 100 +

1 \* 500 ) MOD 1000

= 751 MOD 1000

= 751

**Ans:** 751 **{1}**

1. **KD70DAF** [1]

Working /Hash value

Hash value = ( position in alphabet of ‘K’ (11) +

position in alphabet of ‘D’ (4) \* 10 +

7 \* 100 +

0\* 500 ) MOD 1000

= 751 MOD 1000

= 751

**Ans:** 751 **{1}**

**(c)** Calculating the hash values for the two registration numbers in part 5(b) has produced a collision.

In the context of storing data in files using hashing, explain the effect of this collision and how this might be dealt with. [2]

**Ans:**

**Effect {1}**

The records for both cars would map to / be stored in the same location (in the file) // the record for the second car would overwrite the record for the first;

**How dealt with {1}**

Store one record/car in the next available location in the file // store a pointer (in each file location) that points to a list of records that have all collided at the file location // use a linked list from the location;

**Or** Idea that each storage location could store more than one record e.g. five records per location, if explained - must be clear that each location can store a fixed number of records, just storing the second record in the same location is not enough.

**Or** Example of what “next available” might be

**Or** Key is rehashed

**Or** Table for file

**Or** Use a hashing function that computes different values for the example registration numbers // use a hashing function that uses more characters from the registration number

**4 (a) Ans:** [3]

**Figure 3**

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Top of Stack 🡪 | Jatinder | 🡻 |
|  |  | 🡻 |
|  | Ben | 🡻 |
|  |  | 🡻 |
|  | Ali | 0 |

*for Top of Stack pointer* **{1}**

*for 3 correct items* **{1}**

*for correct order with null pointer in last node* **{1}**

**(b) (i) Ans:** [4]

|  |  |  |  |
| --- | --- | --- | --- |
| TopOfStackPointer |  | Stack | |
| 0 |  | Name | Pointer |
|  |  |  | 2 |
| FreePointer |  |  | 3 |
| 1 |  |  | 4 |
|  |  |  | 5 |
|  |  |  | 6 |
|  |  |  | 7 |
|  |  |  | 8 |
|  |  |  | 9 |
|  |  |  | 10 |
|  |  |  | 0 |

*TopOfStackPointer* **{1}**

*FreePointer* **{1}**

*Pointers[1] to [9]* **{1}**

*Pointer[10]* **{1}**

**(ii) Ans:** [5]

PROCEDURE Pop()

// Report error if Stack is empty

IF TopOfStackPointer = 0

THEN

Error

ELSE

OUTPUT Stack[TopOfStackPointer].Name

// take a copy of the current top of stack pointer

TempPointer 🡨 TopOfStackPointer

// update the top of stack pointer

TopOfStackPointer 🡨 Stack[TempPointer].Pointer

// link released node to free list

Stack[TempPointer].Pointer 🡨 FreePointer

FreePointer 🡨 TempPointer

ENDIF

ENDPROCEDURE

for each line of code as above (first 4 lines + ENDIF for **{5}**)

**(iii) Ans:**  [5]

**{1}**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Push or Pop | TopOfStackPointer | FreePointer |  |  | Stack | |
|  | 0 | 1 |  |  | Name | Pointer |
| Push | 1 | 2 |  | [1] | Ali | 2 |
| Push | 2 | 3 |  | [2] | Ben | 3 |
| Pop | 1 | 2 |  | [3] | Jatinder | 4 |
| Push | 2 | 3 |  | [4] |  | 5 |
| Push | 3 | 4 |  | [5] |  | 6 |
| Pop | 2 | 3 |  | [6] |  | 7 |
| Push | 3 | 4 |  | [7] |  | 8 |
|  |  |  |  | [8] |  | 9 |
|  |  |  |  | [9] |  | 10 |
|  |  |  |  | [10] |  | 0 |
| **{1}** | **{1}** | **{1}** |  |  | **{1}** | **{1}** |

**5 (a) Ans:**  [2]

A procedure that calls itself // is defined in terms of itself **{1}**

/based cases/ending conditions **{1}**

**(b)**  **Ans:** [4]

Before procedure call is executed current state of the registers/local variables is saved onto

the stack **{2}**

When returning from a procedure call the registers/local variables are re-instated **{2}**

**(c)**  **Ans:** [3]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Call number** | **n** | **(n = 0) OR (n = 1)** | **n DIV 2** | **n MOD 2** |
| 1 | 40 | FALSE | 20 | 0 |
| 2 | 20 | FALSE | 10 | 0 |
| 3 | 10 | FALSE | 5 | 0 |
| 4 | 5 | FALSE | 2 | 1 |
| 5 | 2 | FALSE | 1 | 0 |
| 6 | 1 | TRUE |  |  |
|  |  | **{1}** | **{1}** | **{1}** |

OUTPUT **101000** – [2]

1 mark for each pair of bits. **{2}**

**(d) Ans:** . [1]

Conversion of denary number into binary **{1}**

**6 (a) Ans:**  [2]

Two from: **{2}**

Routers inspect/read the IP address of the data packets sent to it

Sends the data packet to a switch with that IP address

It may use the MAC address of the switch to do that by converting the IP to a MAC using ARP…

…until it finds the corresponding switch

Address Resolution Protocol (ARP) is a procedure for mapping a dynamic IP address to a permanent physical machine address in a local area network (LAN).

**(b) Ans:** ] [3]

Three from:

* Stores IP addresses **{1}**
* IP address is a unique identifier set up by network manager/ISP

Can change but should match the network it’s on **{1}**

* IP address consists of 4 numbers separated by full stops **{1}**

**Or**

* Stores MAC addresses, 6 pairs of hexadecimal digits **{2}**
* MAC address is usually hard coded by manufacturer, never changes **{1}**

**(c)** **(i) Ans:** [8]

**Application layer**-This layer sits at the top of the stack and uses protocols relating to the application being used to transmit data over a network, usually the Internet. **{2}**

**Transport layer**-This layer uses the Transmission Control Protocol (TCP) to establish an end-to-end reliable connection with the recipient computer. The data is then split into packets and labelled with the packet number, the total number of packets and the port number through which the packet should route. **{2}**

**Network layer**-This is sometimes referred to the IP layer or Internet layer. It adds the source and destination IP addresses. **{2}**

**Link layer**-This **is** the physical connection between network nodes and adds the unique Media Access Control (MAC) addresses identifying the Network Interface Cards (NICs) of the source and destination computers. These mean that once the packet finds the correct network using the IP address, it can then locate the correct piece of hardware. **{2}**

**(ii)** **Ans:**  [4]

(Link layer) responsible for network drivers // network cabling // physical connection // changing from one medium to another;

(Link layer) (removes MAC address and) adds MAC address for the next hop;

A hardware address for MAC address **{2}**

(Network layer) looks at destination IP address;

Router decides on next appropriate hop (after seeing destination IP address);

(Network layer) can split/combine/resize packets if required;

error checking / error detection;

encryption for wireless connections;

tunnelling through a firewall; **{2}**

**(d) (i) Ans: Two** from: [2]

Picture and sound not synchronised **{1}**

Interruptions // video not continuous **{1}**

Can be degraded by other competing traffic **{1}**

**(ii) Ans:** [2]

In packet switching, packets can take different routes and may not arrive in order

Will arrive in order (only one route)

As packets can take many different routes / share paths with others can be delayed

Dedicated circuit has full bandwidth

No loss of synch

for any valid point **{1} x 2**

**(e)** **(i)** **Ans:** [2]

Host A will encrypt the message using Host B's **public**; key. **{1}**

The message will be decrypted by Host B using Host B's **private**; key. **{1}**

1. **Ans:** [2]

Detect (unauthorised) changes to message. **{1}**

Authenticate sender's identity // confirm who sent it; **{1}**

**7** **(a)**

**(i)** **Ans:** [1]

The table/each student has a repeated group of attributes. // Each student has a number of subjects.

**{1}**

1. **Ans:** [1]

StudentName, TutorGroup and Tutor would need to be repeated for each record. **{1}**

1. **Ans:** [3]

Table: Student

|  |  |  |
| --- | --- | --- |
| StudentName | TutorGroup | Tutor |
| Tommy | 6 | TAN |
| John | 7 | GOH |
| Sammy | 6 | TAN |

Table: StudentSubjectChoices

|  |  |  |  |
| --- | --- | --- | --- |
| StudentName | Subject | Level | SubjectTeacher |
| Tommy | Physics | H2 | TAN |
| Tommy | Chemistry | H2 | GOH |
| Tommy | General Studies | H1 | WEE |
| John | Geography | H1 | ROG |
| John | French | H1 | HEN |
| Sammy | Computer Science | H2 | VAR |
| Sammy | Chemistry | H2 | GOH |
| Sammy | Maths | H2 | ZEN |
| Sammy | General Studies | H2 | WEE |

Mark as follows:

complete Student table **{1}**

repetition of StudentName in StudentSubjectChoices table **{1}**

complete columns 2, 3, and 4 **{1}**

1. **(i)** **Ans:** [2]

primary key

- an attribute/combination of attributes **{1}**

- chosen to ensure that the records in a table are unique // used to identify a record/tuple **{1}**

1. **Ans:** [1]

StudentName + Subject (This is the only correct answer.) **{1}**

1. **Ans:** [2]

* There is a one-to-many relationship. // Student is the ‘one side’ table –

StudentSubjectChoices is the ‘many side’ table.

* the primary key (attribute StudentName) in Student
* links to StudentName in the StudentSubjectChoices table
* (StudentName in the) StudentSubjectChoices table is the foreign key. //

StudentName is the foreign key that links the two tables. **{2}**

1. **Ans:** [2]

* there are non-key attributes ...
* SubjectTeacher ...
* dependent only on part of the primary key (i.e. Subject) // partial dependency **{2}**

1. **Ans:** [2]

there are dependent non-key attributes // there are non-key dependencies

TutorGroup is dependent on Tutor // Tutor is dependent on TutorGroup **{2}**

1. **Ans:** [3]

**Problem Conditions {1}**

When two users try to update the same record simultaneously;

**How dealt with {2}**

Alternative 1 - Record Locks

Maintain information about which records are currently being accessed;

When a user tries to access a record, consult this information and only permit access if record is not currently being used // only permit read access to a record that is already open;

Alternative 2 - Transaction Queuing

Updates / database changes are (grouped as transactions and) queued;

Database software processes transactions in FIFO order from queue;