April 2023

Fundamental IT Engineer Examination (Afternoon)

Questions must be answered in accordance with the following:

Question Nos.	Q1	Q2 – Q5	Q6	Q7, Q8
Question Selection	Compulsory	Select 2 of 4	Compulsory	Select 1 of 2
Examination Time	13:30 – 16:00 (150 minutes)			

Instructions:

1. Use a pencil. If you need to change an answer, erase your previous answer completely and neatly. Wipe away any eraser debris.

2. Mark your examinee information and test answers in accordance with the instructions below. Your answer will not be graded if you do not mark properly. Do not mark or write on the answer sheet outside of the prescribed places.

   (1) **Examinee Number**

   Write your examinee number in the space provided, and mark the appropriate space below each digit.

   (2) **Date of Birth**

   Write your date of birth (in numbers) exactly as it is printed on your examination admission card, and mark the appropriate space below each digit.

   (3) **Question Selection**

   For questions **Q2** through **Q5**, and **Q7** and **Q8**, mark the ⓢ of the questions you select to answer in the "Selection Column" on your answer sheet.

   (4) **Answers**

   Mark your answers as shown in the sample question below.

   [Sample Question]
   Which of the following should be used for marking your answer on the answer sheet?

   Answer group
   a) Ballpoint pen   b) Crayon   c) Fountain pen   d) Pencil

   Since the correct answer is " d) Pencil ", mark the answer as below:

   [Sample Answer]

Sample	ⓐ ⓑ ⓒ ● ⓔ ⓕ ⓖ ⓗ ⓘ ⓙ

   Do not open the exam booklet until instructed to do so.
   Inquiries about the exam questions will not be answered.

Notations used in the pseudo-language

In questions that use pseudo-language, the following notations are used unless otherwise stated:

[Declaration, comment, and process]

Notation	Description
type: *var1*, … , *array1*[], …	Declares variables *var1*, … , and/or arrays *array1*[], … , by data *type* such as INT and CHAR.
FUNCTION: *function*(*type*: *arg1*, …)	Declares a *function* and its arguments *arg1*, … .
/* comment */ or // comment	Describes a comment.
Process	*variable* ← *expression*;	Assigns the value of the *expression* to the *variable*.
	function (*arg1*, …);	Calls the *function* by passing / receiving the arguments *arg1*, … .

	IF (*condition*) {
	    process1
	}
	ELSE {
	    process2
	}
		Indicates the selection process.
		If the *condition* is true, then *process1* is executed.
		If the *condition* is false, then *process2* is executed, when the optional ELSE clause is present.

	WHILE (*condition*) {
	    process
	}
		Indicates the "WHILE" iteration process.
		While the *condition* is true, the *process* is executed repeatedly.

	DO { *process* } WHILE (*condition*);	Indicates the "DO-WHILE" iteration process. The *process* is executed once, and then while the *condition* is true, the *process* is executed repeatedly.

	FOR (*init*; *condition*; *incr*) {
	    process
	}
		Indicates the "FOR" iteration process.
		While the *condition* is true, the *process* is executed repeatedly.
		At the start of the first iteration, the process *init* is executed before testing the *condition*.
		At the end of each iteration, the process *incr* is executed before testing the *condition*.

[Logical constants]
    true, false

[Operators and their precedence]

Type of operation	Unary	Arithmetic		Relational	Logical	
Operators	+, −, not	×, ÷, %	+, −	>, <, ≥, ≤, =, ≠	and	or
Precedence	High ◄———————————————————————————► Low					

Note: With division of integers, an integer quotient is returned as a result.
    The "%" operator indicates a remainder operation.

Q1. Read the following description of TLS version 1.2 connection establishment process, and then answer Subquestion.

HTTPS refers to the combination of HTTP and Secure Socket Layer (SSL) or Transport Layer Security (TLS). TLS is the successor to SSL and is considered the current standard. It is a cryptographic protocol that can be used to enable secure communication between a web browser and a web server by encapsulating HTTP data inside TLS packets. HTTPS allows the client, in this case, a web browser, to communicate confidentially with the web server and allow the server authentication. The identity of the web server is required to be verified beforehand by submitting the web server certificate to a trusted third-party certification authority (CA). The information submitted for verification include the fully qualified domain name (FQDN) of the web server and the public key that is linked to the private key pair installed on the web server itself. Then, the CA signed the web server certificate with its private key and set the validity period of the signed certificate. Subsequently, the signed certificate is returned to be installed on the web server. The standard format of this type of certificate is X.509.

In general, TLS aims to authenticate a server and start encryption between the client and the server. Therefore, various mechanisms can be used to ensure that both client and server have the same key for a particular session. This is called a session key (also called a shared key) which will be calculated and derived from pieces of information exchanged between the client and server. These take place during TLS connection establishment. A simplified process of TLS connection establishment is shown in Figure 1.

[TLS connection establishment process]
(1) ClientHello: The client sends a client hello message containing the information regarding its cryptographic capabilities, the highest TLS version understood by the client and other related information including a client-generated random number to be used for master secret generation to the server.

(2) ServerHello: The server selects connection parameters, including the encryption and compression mechanism understood by both servers, generates another random number, and then sends them back to the client.
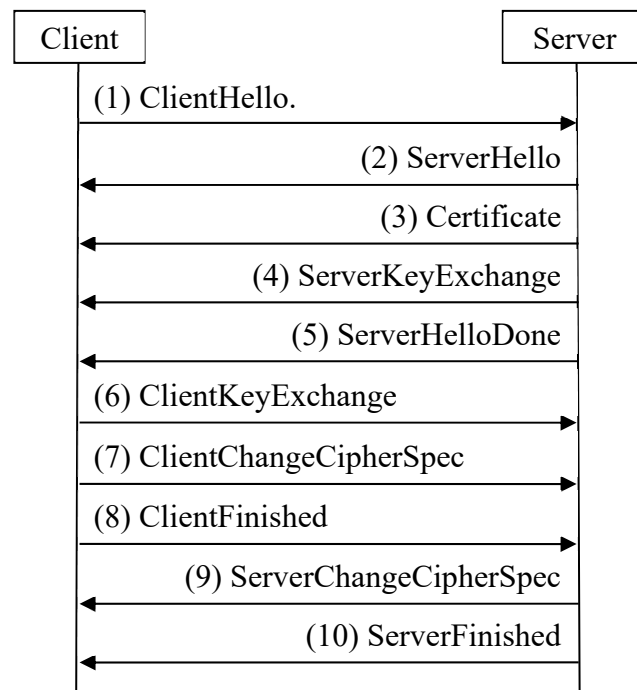
Figure 1 TLS connection establishment process

(3) Certificate: This message usually carries the server's X.509 certificate chain, including the server's public key and other related information to the client. Then, the client can authenticate the server's X.509 certificate with the corresponding CA. Since the server's certificate was digitally signed with the [A], it can then be verified with the CA's public key residing in the CA's certificate. If the authentication fails, the client will warn the user that the certificate is invalid or cannot be verified and will ask whether the user wishes to proceed.

(4) ServerKeyExchange: This message carries additional information required to generate the master secret depending on the selected cipher suite. It is optional and may not be sent if the key exchange used does not require additional information.

(5) ServerHelloDone: This message indicates that the server has completed sending all intended information. Then, it will wait for the response from the client.

(6) ClientKeyExchange: This message carries the client-contributed information required and sufficient to generate the master secret depending on the selected cipher suite. Compare to the server key exchange message, the client key exchange message is mandatory.

(7) ClientChangeCipherSpec: The client sends this message to inform the server that the client has enough information to generate the encryption key, including all required parameters to start encryption. The client switches to encryption after this message.

(8) ClientFinished: The client sends this message to indicate that the connection establishment is completed. It contains a hash of all messages sent and received mixed with the master secret, allowing the server to verify the integrity of the establishment process.

(9) ServerChangeCipherSpec: The server sends this message to inform the client that the server has enough information to generate the encryption key, including all required parameters to start encryption. The server switch to encryption after this message.

(10) ServerFinished: The server sends this message to indicate that the connection establishment on its side is completed. It contains a hash of all messages received and sent mixed with the master secret allowing the client to verify the integrity of the establishment process.

Table 1 shows the summary of the above process.

Table 1 Summary of the TLS connection establishment process

Steps	Description
(not shown)	Exchange capabilities and agree upon the desired connection parameters, which is completed at step ⬚ B ⬚.
(not shown)	Validate the certificate(s) or authenticate with other methods.
(4), (5), (6), (7) and (9)	Agree on a shared master secret that will be used as ⬚ C ⬚ provided that the server will have enough information required to generate the master secret after receiving the message at step ⬚ D ⬚.
(8) and (10)	Start to send messages on the encrypted communication channel and verify the messages used during the process to ensure that they are not tampered with by a third party.

Here, steps (3), (4), and (6) are considered as a single process called the key exchange. The goal of the key exchange is to generate the premaster secret, which will then be used to construct the master secret later. One of the most well-known key exchange algorithms is the RSA key exchange. The mechanism of RSA key exchange is explained below:

[RSA key exchange]
RSA is universally supported as the standard key exchange algorithm. In this case, the client is responsible for generating a premaster secret and encrypting it with ⬚ E ⬚. The result is sent in the ClientKeyExchange message to the server. The server will then decrypt the premaster secret and use it to generate the master secret later.

The strength of HTTPS comes from the fact that the master secret or the session key is newly generated for every new session, although the web server's private and public key pair remains the same. Public key cryptography is used only at the beginning of the session for the key exchange operation. All subsequent connections in such a session are performed using a symmetric key. Therefore, even if the traffic data are captured and the key used for that session is exposed, such a key is not applicable to any other session. However, if [F] is leaked, all related sessions, both in the past and in the future, are vulnerable since it is possible to derive session keys from the captured session traffic.

Other superior key exchange mechanisms include the Diffie-Hellman ephemeral key exchange with RSA (DHE-RSA). This method allows a shared secret to be established over an insecure communication channel. Instead of public key cryptography, it uses a mathematical function that is easy to calculate in one direction but is difficult to reverse. In this method, RSA is used only for signing the exchanged parameters, not for encryption, and the shared secret to be established is chosen randomly for each session; thus, it is difficult to decrypt the contents of that session even if a malicious person has the leaked [F]. Therefore, these key-exchange mechanisms should also be considered as well in practical use.

Subquestion

From the answer groups below, select the correct answer to be inserted in each blank [   ] in the above description.

Answer group for A, C, E and F
- a) a random number
- b) CA's private key
- c) CA's public key
- d) the server's private key
- e) the server's public key
- f) the session key

Answer group for B and D
- a) (2) ServerHello
- b) (3) Certificate
- c) (4) ServerKeyExchange
- d) (5) ServerHelloDone
- e) (6) ClientKeyExchange
- f) (7) ClientChangeCipherSpec
- g) (8) ClientFinished
- h) (10) ServerFinished

Q2. Read the following description of file systems, and then answer Subquestions 1 and 2.
  Note: In this question, one kilobyte (1 kB) is 1024 bytes.

Files and directories stored on a storage device can be arranged and managed differently depending on file systems.

A storage device is normally formatted into blocks of the same size, and all files and directories are placed into free blocks on the storage device. However, if the file is larger than the size of the block, it will be divided into multiple pieces and placed into available blocks. For example, in a file system with 4 kB blocks, 10 kB data of a file is divided into three pieces and placed into three blocks.

[Linked allocation method]
In the linked allocation method, the list of the files is stored in a table called a directory, where each entry contains the file name and other information, including the block number, indicating the first block of the file.

Apart from the data, each block contains the next block number in the chain containing the next part of the file and continues until the last block, where the end of file (EOF) value is stored instead of the block number.

Figure 1 shows an example of a file system using the linked allocation method. It is assumed that the directory is located at block number 0.
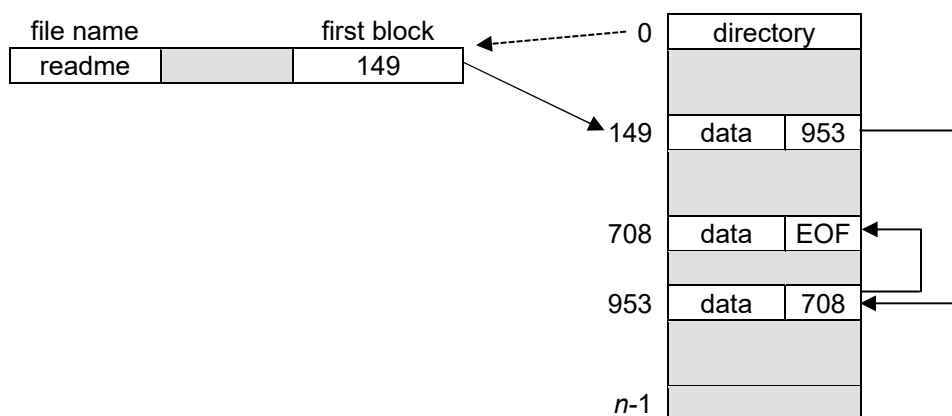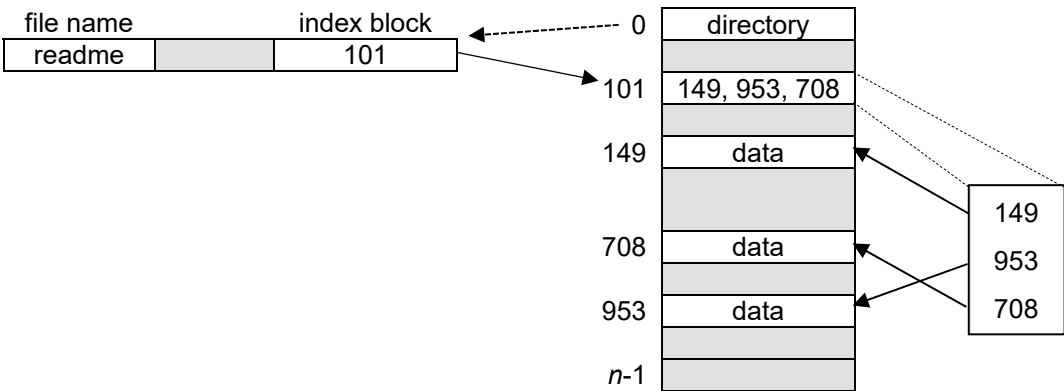

Note: Shaded parts are not shown, and n is the number of blocks on the storage device.

Figure 1 Example of a file system using the linked allocation method

In Figure 1, the upper-left part represents a directory entry for the file named readme, and the right part represents the blocks on the storage device. The arrows represent the links from the block number information to the actual block on the storage device. The file readme is stored in block numbers 149, 953, and 708 in that order.

This allocation method is simple and easy to implement. However, this scheme has a disadvantage because ☐　A　☐.

[Indexed allocation method]

In the indexed allocation method, each file is associated with an index block. The index block maintains a list of block numbers of the data related to the associated file.

Figure 2 shows an example of a file system using the indexed allocation method.

Note: Shaded parts are not shown, and n is the number of blocks on the storage device.

Figure 2 Example of a file system using the indexed allocation method

In Figure 2, the upper-left part represents a directory entry for the file readme. It indicates that the index block of the file readme is stored in block number 101. Block number 101 shows that the data of the file readme is stored in blocks 149, 953, and 708 in that order. Therefore, after reading the index block, it is easy to find the block number of each part of the data. However, this scheme also has a disadvantage because ☐　B　☐.

[Calculation of the required number of blocks]

In the linked allocation method, each block holds an index value (the next block number). In the indexed allocation method, an index block is needed separately instead of having an index value in each block.

Consider the case where a file contains 40 kB data. On the assumption that the size of a block is 4 kB and the size of an index value is 4 bytes, ☐　C1　☐ blocks are required to store the file in the linked allocation method, and ☐　C2　☐ blocks are required in the indexed allocation method. Here, the block for the directory entry is ignored.

Subquestion 1

From the answer group below, select the most appropriate answer to be inserted in each blank [   ] in the above description. Here, the answers to be inserted in C1 and C2 should be selected as the correct combination from the answer group for C.

Answer group for A and B
- a) it cannot store a sequential file when the required number of continuous free blocks is unavailable.
- b) it is ineffective against a direct-access file since it must start at the beginning of the file and continue through each block to find the specific part of the file.
- c) it usually has greater wasted space than the other method especially in the system with a large number of small files.
- d) the content of the file cannot be stored in consecutive blocks due to the limitation of the referencing mechanism, resulting in lower overall performance.

Answer group for C

	C1	C2
a)	10	10
b)	10	11
c)	11	10
d)	11	11

Subquestion 2

From the answer group below, select the correct answer to be inserted in each blank [   ] in the following description.

[Inode allocation method]

The inode allocation method is used in UNIX-based systems.

The inode allocation method combines the linked and indexed allocation methods into one. Instead of using a table to look up a file, an entity called inode is used to store the information about a file, similar to the working of the index block in Figure 2. It is an indexed allocation method with the advantage of flexible indexing by chaining up to four index nodes.

The concept of inode used in UNIX-based systems is shown in Figure 3.

There are 15 pointers in an inode. The first 12 pointers directly point to data blocks; thus, any part of the file can be directly accessed if it occupies less than or equal to 12 blocks. For a larger file, the single indirect, double indirect, and triple indirect pointers will be used based on the file size. The single indirect pointer points to a block containing direct pointers, while the double indirect pointer points to a block containing pointers to another single indirect pointer.
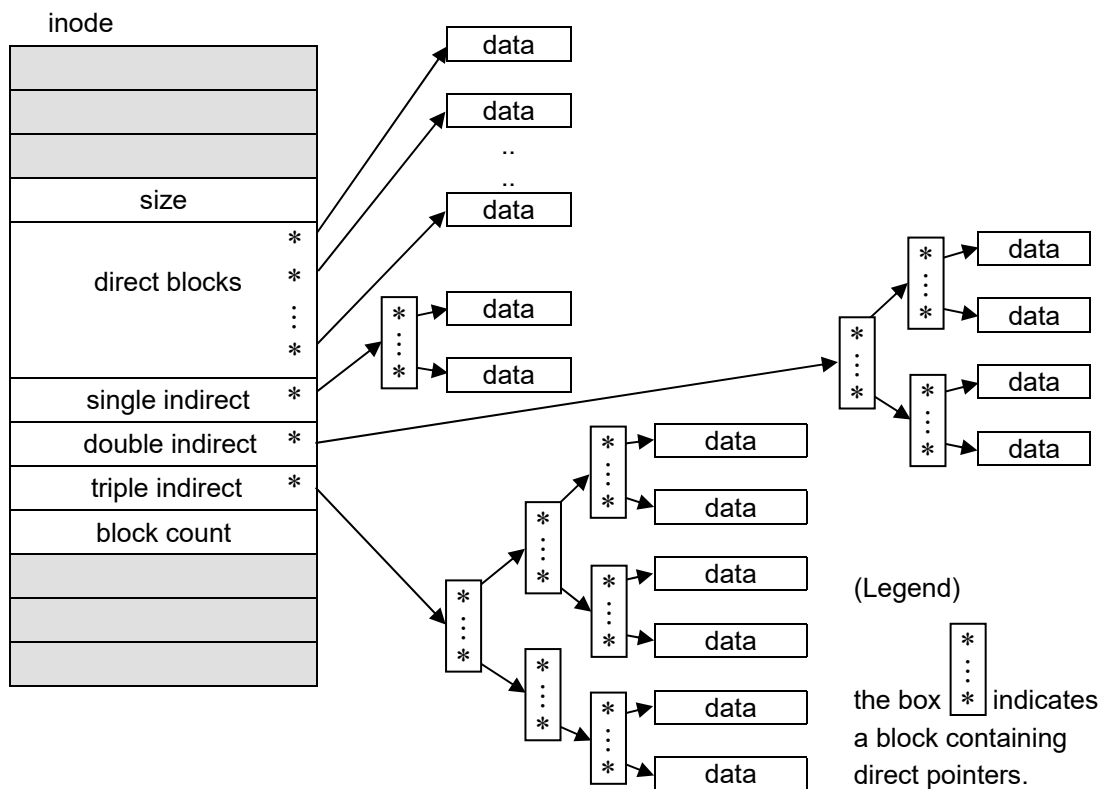
Note: Shaded parts are not shown, and an "*" indicates a pointer to a block.

Figure 3 Concept of inode allocation method in UNIX-based systems

Table 1 shows the largest accessible data size by pointer level. Here, the block size is 4096 bytes (4 kB) and the pointer size is 4 bytes, i.e., a block holds 1024 pointers.

Table 1 Largest accessible data size by pointer level

Pointer level	Number of pointers	Largest accessible data size
direct blocks	12	48 kB
single indirect	1024	4 MB
double indirect	1024 × 1024	4 GB
triple indirect	1024 × 1024 × 1024	4 TB

Consider the case where a file contains 1 MB of data. On the assumption that the size of a block is 4 kB and the size of a pointer is 4 bytes, [D] blocks are required to store the file in the inode allocation method. Here, the block for the inode is ignored. Use 1MB ÷ 4 kB = 256.

It is possible to specify a different block size other than 4 kB while formatting a file system. A smaller block size helps reduce occupied storage size per file for smaller files, while a larger block size helps reduce the number of blocks per file for larger files.
Change of the block size affects the maximum accessible file size by two factors: the number of bytes per data block and the number of pointers per index block. For example, if the size of a block is 1 kB and the size of a pointer is 4 bytes, the maximum accessible file size will be approximately [E] in the inode allocation method.

Answer group for D
   a) 256   b) 257   c) 258   d) 259

Answer group for E
   a) 16 GB   b) 32 GB   c) 64 GB   d) 128 GB
   e) 256 GB   f) 512 GB   g) 1 TB

Q3. Read the following description of a relational database for a DVD shop, and then answer Subquestions 1 through 3.

DVD shop U provides services of delivering, renting, and buying DVD films to its customers. The accounting division of DVD shop U utilizes a database for payment management. The database consists of four tables: Customer, Store, Staff, and Payment.

The table structures and examples of data storage are as follows:

Customer table

CustomerID	FirstName	LastName	Email	City
C001	Bill	Smith	billsmith@example.net	South lake
C002	Baelfire	Grehn	baelfiregrehn@example.net	East wood
C003	Bill	Gretan	billgretan@example.net	West hill
C004	Brendon	Green	brendongreen@example.net	East wood
C005	John	Doe	johndoe@example.net	North coast

Store table

StoreID	StoreName	ManagerStaffID
10	South-west shop	S001
20	North-east shop	S002

Staff table

StaffID	FirstName	LastName	Email	StoreID
S001	Mike	Hillyer	Mike.Hillyer@example.com	10
S002	Jon	Stephens	Jon.Stephens@example.com	20

Payment table

PaymentID	CustomerID	StaffID	Amount	PayDate
P001	C002	S002	2.99	2023-04-05
P002	C003	S001	0.99	2023-04-08
P003	C001	S001	5.99	2023-04-14
P004	C003	S001	4.99	2023-04-15
P005	C005	S002	9.99	2023-04-21

Subquestion 1

From the answer group below, select the correct answer to be inserted in each blank [   ] in the following SQL statement.

The following SQL statement SQL1 outputs how much sales amount, in dollars, each store brought in. The output records are ordered from higher to lower sales amounts.

```
-- SQL1 --
SELECT st.StoreID, st.StoreName,     A     AS Sales
FROM       B
INNER JOIN Staff s  ON s.StaffID = p.StaffID
INNER JOIN Store st ON st.StoreID = s.StoreID
GROUP BY st.StoreID, st.StoreName
ORDER BY      C
```

The `INNER JOIN` keyword in this statement selects all rows from both tables in case there is a match between the columns.

From the sample data shown in the table definitions, SQL1 outputs the following result:

StoreID	StoreName	Sales
20	North-east shop	12.98
10	South-west shop	11.97

Answer group for A through C

a) `Amount`

b) `Payment`

c) `Payment p`

d) `Sales`

e) `Sales ASC`

f) `Sales DESC`

g) `SUM(Amount)`

Subquestion 2

From the answer group below, select the correct answer to be inserted in the blank [   ] in the following SQL statement.

One day, a store staff found that a customer had left his/her PC in the store. On the PC, there was a name sticker. Unfortunately, the characters were blurred and the staff could not clearly read the owner's name. Finally, the staff identified that:

o the first character of the first name was "B".

o the last name consists of five characters; the first three characters were "Gre", and the last character was "n".

The following SQL statement SQL2 extracts customers whose names are closest to the name on the PC.

```
-- SQL2 --
SELECT CustomerID, FirstName, LastName, Email
FROM Customer
WHERE        D
```

Two wildcard characters often used in a search pattern in pattern matching are as follows:

'%' (percent sign) … represents any string of zero or more characters

'_' (underscore) … represents any single character

From the sample data shown in the table definitions, SQL2 outputs the following result:

CustomerID	FirstName	LastName	Email
C002	Baelfire	Grehn	baelfiregrehn@example.net
C004	Brendon	Green	brendongreen@example.net

Answer group for D

a) (FirstName LIKE 'B%') AND (LastName LIKE 'Gre%n')

b) (FirstName LIKE 'B%') AND (LastName LIKE 'Gre_n')

c) (FirstName LIKE 'B_') AND (LastName LIKE 'Gre%n')

d) (FirstName LIKE 'B_') AND (LastName LIKE 'Gre_n')

Subquestion 3

From the answer group below, select the correct answer to be inserted in the blank [　　　] in the following SQL statement.

DVD shop U is planning a discount campaign for repeaters. The following SQL statement SQL3 extracts customers who paid twice or more from April 1 to April 15, 2023.

```
-- SQL3 --
SELECT c.CustomerID, c.LastName, COUNT(*) AS PayCount
    FROM Customer c, Payment p
    WHERE c.CustomerID = p.CustomerID
```

```
        E
```

From the sample data shown in the table definitions, SQL3 outputs the following result:

CustomerID	LastName	PayCount
C003	Gretan	2

Answer group for E

a) ```
AND p.PayDate >= '2023-04-01' AND p.PayDate <= '2023-04-15'
AND COUNT(*) >= 2
```

b) ```
AND p.PayDate BETWEEN '2023-04-01' AND '2023-04-15'
GROUP BY c.CustomerID, c.LastName
HAVING COUNT(*) >= 2
```

c) ```
GROUP BY c.CustomerID, c.LastName, p.PayDate
HAVING p.PayDate >= '2023-04-01' AND p.PayDate <= '2023-04-15'
    AND COUNT(*) >= 2
```

d) ```
HAVING p.PayDate BETWEEN '2023-04-01' AND '2023-04-15'
    AND COUNT(*) >= 2
```

Q4. Read the following description of redesign of a company network, and then answer Subquestions 1 and 2.

Company V is a service integrator company. It currently operates with 100 staff.

A director of Company V plans to extend the services by increasing the number of staff and improving the security controls in each department. A network administrator plans to redesign the existing internal network infrastructure by introducing a variable-length subnet mask network address scheme (different segments of networks) using private IP addresses (172.16.10.0/24). The network segments are connected using a Core Router (CR) and Firewall (FW).

Figure 1 shows the design of the network configuration of company V.
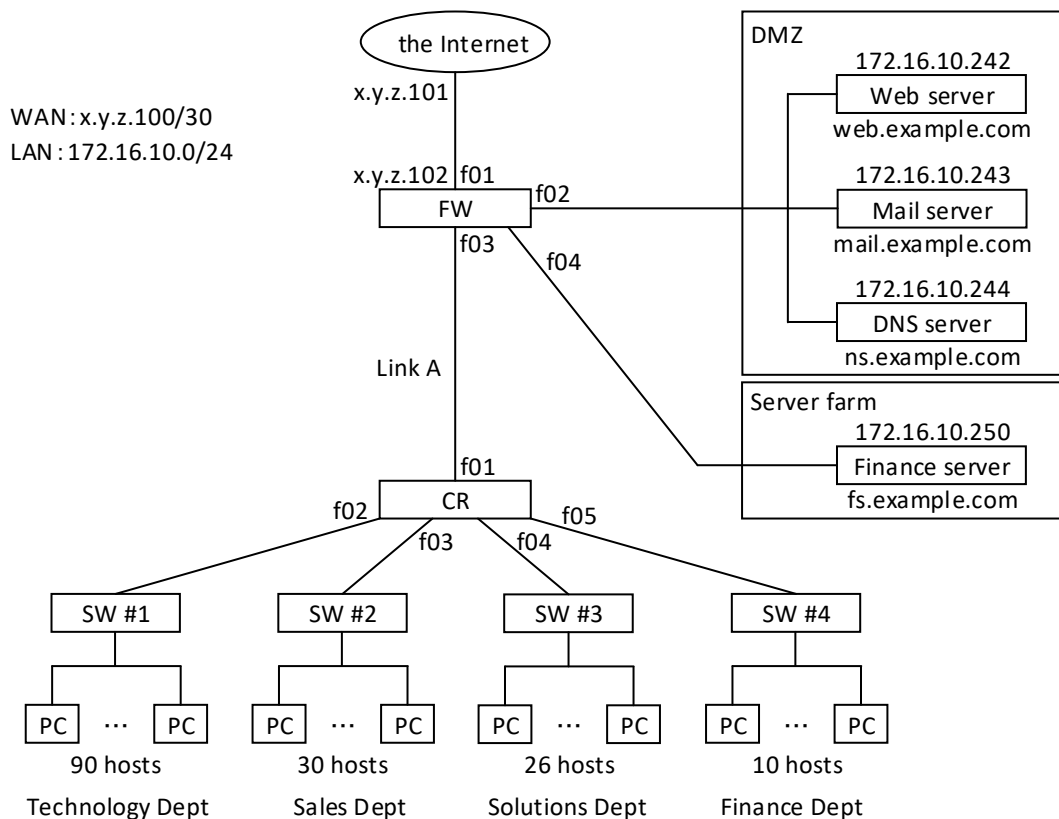
Note: example.com is the domain name of Company V.

Figure 1: Network configuration of Company V

The Network Address Translation function is placed at FW to access the Internet from the internal network. A Mail server, a DNS server, and a Web server are in the DMZ. FW blocks all incoming communication from the Internet to the internal network including the DMZ, except for explicitly allowed communication. IP addresses (x.y.z.100/30) are used for a wide area network (WAN) between the Internet and FW.

Subquestion 1

From the answer groups below, select the correct answer to be inserted in each blank ☐ in the following description and Table 1.

To design a variable length subnet mask network address scheme (different segments of networks), the network administrator uses the following scheme:

o The network segments are designed per the requirements of the number of hosts in each network, starting from the largest LAN to the smallest LAN. Note that the number of hosts includes not only servers and PCs but also the interfaces of network devices connected to the segment.

o After determining the addresses for all the LAN subnets, the first available address is assigned to the interface of FW and the second to the interface of CR for Link A.

o The first available IP address is used as the gateway of each network segment and assigned at the router interfaces. However, for the DMZ and Server farm, the gateway of the network segment is assigned at the interface of FW. The gateway of the DMZ servers is ☐ A ☐.

o Table 1 shows the required different segments of networks as per specific hosts of each network.

Table 1 Variable length subnet mask network address scheme

Number	Network name	Required number of hosts	Network mask
1	Technology department	92	172.16.10.0/25
2	Sales department	32	B
3	Solutions department	28	C
4	Finance department	12	172.16.10.224/28
5	DMZ	4	172.16.10.240/29
6	Server farm	2	172.16.10.248/30
7	Link A	2	172.16.10.252/30

Answer group for A
  a) 172.16.10.1
  b) 172.16.10.240
  c) 172.16.10.241
  d) 172.16.10.248
  e) 172.16.10.249

Answer group for B and C
  a) 172.16.10.0/26
  b) 172.16.10.0/27
  c) 172.16.10.127/26
  d) 172.16.10.128/26
  e) 172.16.10.128/27
  f) 172.16.10.160/27
  g) 172.16.10.191/26
  h) 172.16.10.191/27
  i) 172.16.10.192/26
  j) 172.16.10.192/27

Subquestion 2

From the answer group below, select the correct answer to be inserted in each blank [   ] in Table 2.

After designing the different network segments, the network administrator configured the routing protocol on the CR and FW. As a result, the routing function is working properly and all hosts on the LANs can connect with each other and access to the Internet.

To improve the security of servers located in DMZ and the Finance server, the network administrator implemented several zone-based security policies on FW according to the following rules:

(1) Secure web services (HTTPS) on the external IP address of FW will be mapped to the Web server in DMZ.

(2) Mail transferring services (SMTP) and receiving services (IMAPS) on the external IP address of FW will be mapped to the Mail server in DMZ.

(3) Allow access from DMZ to the Internet via NAT (IP address translation), which is necessary for the correct functionality of the mapped service.

(4) Allow access only from the Finance department to the Finance server with a secure web service to access financial information from the server.

(5) Other hosts from LANs and external users cannot access the Finance server.

(6) Allow other traffic from LANs to DMZ, allowing access to multiple resources at the servers from local users.

Table 2 shows the configured traffic rule table on FW. Each incoming packet is inspected according to the rules on FW as follows:

(1) First, the source, destination, and destination service of the packet are compared with those in rule number 1.

    (i) If it matches, the specified action is executed. If the translation is specified, it will also be applied. Once the action is executed, subsequent rules are not checked.

    (ii) If it does not match, proceed to the next rule.

(2) Repeat step (1) until the end of the rule table reaches.

(3) Deny the packet if no action is executed in the rule table.

Table 2 Traffic rule table on FW (other settings are not shown)

Rule No.	Source	Destination	Destination Service	Action	Translation
1	the Internet	x.y.z.102	HTTPS	Allow	MAP: 172.1.10.242
2	the Internet	x.y.z.102	SMTP/IMAPS	Allow	MAP: 172.1.10.243
3	DMZ	D	ANY	Allow	NAT
4	E	172.16.10.250	HTTPS	Allow	
5	F	172.16.10.250	ANY	Deny	
6	LANs	DMZ	ANY	Allow	
⋮	⋮	⋮	⋮	⋮	

Note DMZ: A perimeter network that protects private networks from untrusted traffic. It is a subnetwork that sits between the Internet and private networks.

    the Internet: Untrusted or public network. FW is needed to protect private networks from untrusted traffic.

    LANs: Trusted or private network that can access any trusted traffic to DMZ and the Internet.

    ANY: All networks.

Answer group for D through F

    a) 172.16.10.0/24   b) 172.16.10.64/26   c) 172.16.10.128/26

    d) 172.16.10.224/28   e) 172.16.10.240/29   f) ANY

    g) DMZ   h) the Internet

Q5. Read the following description of program development and testing, and then answer Subquestion.

State S consists of four cities: East, North, South, and West. The State Statistics Bureau (SSB) maintains a statistic file containing the information shown in Table 1.

Table 1 Contents of the statistic file

City name	Total population	Adult population	Employed	Unemployed
East	124,000	104,000	90,000	10,000
North	252,000	208,000	176,000	24,000
South	132,000	106,000	86,000	14,000
West	260,000	212,000	168,000	32,000

SSB has developed a program that outputs a statistic report from the statistic file. The program is almost complete, and SSB is now preparing test cases.

[Program Description]

(1) The program reads the statistic file and outputs a statistic report.

(2) For each record of the statistic file, the program calculates and outputs the labor force rate and unemployment rate using the following formulas:

Labor force rate (%) = 100 × (Employed + Unemployed) ÷ Adult population

Unemployment rate (%) = 100 × Unemployed ÷ (Employed + Unemployed)

(3) When all the records are processed, the program outputs the total of the total population, adult population, employed, and unemployed, and the statewide labor force rate and unemployment rate. After that, the program outputs the highest and lowest unemployment rates with the city names.

(4) The following is an example of a statistic report:

```
City   TotalPop  AdultPop   Employ  Unemploy  Labor%  Unemp%
-----  --------  --------  --------  --------  ------  ------
East    124,000   104,000    90,000    10,000    96.2    10.0
North   252,000   208,000   176,000    24,000    96.2    12.0
South   132,000   106,000    86,000    14,000    94.3    14.0
West    260,000   212,000   168,000    32,000    94.3    16.0
-----  --------  --------  --------  --------  ------  ------
TOTAL   768,000   630,000   520,000    80,000    95.2    13.3

Highest Unemp%: 16.0, City: West
Lowest  Unemp%: 10.0, City: East
```

[Program]
```
    STRING: City, CityHigh ← "????", CityLow ← "????"
    INT:  TotalPop, AdultPop, Employ, Unemploy
    INT:  TotalTotalPop ← 0, TotalAdultPop ← 0,
          TotalEmploy ← 0,   TotalUnemploy ← 0
    REAL: LaborRate, UnempRate
X → REAL: UnempRateHigh ← 0.0, UnempRateLow ← 100.0

    OpenFile("statistic");    /* Open the statistic file */
    Print("City   TotalPop AdultPop   Employ Unemploy Labor% unemp%");
    Print("----- -------- -------- -------- -------- ------ ------");
    ReadRecord(City, TotalPop, AdultPop, Employ, Unemploy);
    WHILE (not end-of-file) {    /* Loop until the statistic file reaches the end-of-file */
        LaborRate ← 100.0 × (Employ + Unemploy) ÷ AdultPop;
        UnempRate ← 100.0 × Unemploy ÷ (Employ + Unemploy);
Y →     IF (UnempRate > UnempRateHigh) {
            CityHigh ← City;
            UnempRateHigh ← UnempRate;
        }
Z →     IF (UnempRate < UnempRateLow) {
            CityLow ← City;
            UnempRateLow ← UnempRate;
        }
        TotalTotalPop ← TotalTotalPop + TotalPop;
        TotalAdultPop ← TotalAdultPop + AdultPop;
        TotalEmploy   ← TotalEmploy   + Employ;
        TotalUnemploy ← TotalUnemploy + Unemploy;
        Print(City, TotalPop, AdultPop,
                    Employ, Unemploy, LaborRate, UnempRate);
        ReadRecord(City, TotalPop, AdultPop, Employ, Unemploy);
    }
    Print("----- -------- -------- -------- -------- ------ ------");
    LaborRate ← 100.0 × (TotalEmploy + TotalUnemploy) ÷ TotalAdultPop;
    UnempRate ← 100.0 × TotalUnemploy ÷ (TotalEmploy + TotalUnemploy);
    Print("TOTAL", TotalTotalPop, TotalAdultPop,
                    TotalEmploy, TotalUnemploy, LaborRate, UnempRate);
    Print();    /* Print a blank line */
    Print("Highest Unemp%:", UnempRateHigh, ", City:", CityHigh);
    Print("Lowest  Unemp%:", UnempRateLow,  ", City:", CityLow );
    CloseFile("statistic");    /* Close the statistic file */
```

Note: The function ReadRecord(v₁, v₂, …) reads the next record from the statistic file, and stores the values into variables v_1, v_2, … .

The function Print(p₁, p₂, …) prints arguments p_1, p_2, … in one line. Assuming that spaces between arguments are adjusted automatically.

Subquestion

From the answer groups below, select the correct answer to be inserted into each blank [   ] in the following description.

Note that the shaded parts [   ] are not shown.

The program development is now in a test phase, and testing is performed using various test cases. Four test cases and their test results are shown below:

[Test case 1]

This is the case where two cities have the same unemployment rate for both the highest and lowest rates. In this case, the program outputs the following report:

```
City   TotalPop  AdultPop    Employ  Unemploy  Labor%  Unemp%
-----  --------  --------  --------  --------  ------  ------
East    120,000   104,000    90,000    10,000    96.2    10.0
North   120,000   105,000    90,000    10,000    95.2    10.0
South   120,000   106,000    85,000    15,000    94.3    15.0
West    120,000   107,000    85,000    15,000    93.5    15.0
-----  --------  --------  --------  --------  ------  ------
TOTAL   480,000   422,000   350,000    50,000    94.8    12.5

Highest Unemp%: 15.0, City: [   A   ]
Lowest  Unemp%: 10.0, City: [       ]
```

[Test case 2]

This is the case where all the cities have a 0% unemployment rate. According to the program logic, the city names shown on the last two lines are as follows:

```
City   TotalPop  AdultPop    Employ  Unemploy  Labor%  Unemp%
-----  --------  --------  --------  --------  ------  ------
East    120,000   104,000    90,000         0    86.5     0.0
North   120,000   105,000    90,000         0    85.7     0.0
South   120,000   106,000    85,000         0    80.2     0.0
West    120,000   107,000    85,000         0    79.4     0.0
-----  --------  --------  --------  --------  ------  ------
TOTAL   480,000   422,000   350,000         0    82.9     0.0

Highest Unemp%: 0.0, City: [   B   ]
Lowest  Unemp%: 0.0, City: [       ]
```

[Test case 3]

The phenomenon mentioned in test case 2 can be resolved by changing the operator ">" to "≥" on line Y and changing the operator "<" to "≤" on line Z .

After these changes, another case where all the cities have a 100% unemployment rate is tested. This time, the program outputs the following report:

```
City   TotalPop  AdultPop    Employ  Unemploy  Labor%  Unemp%
-----  --------  --------  --------  --------  ------  ------
East    120,000   104,000         0    90,000    86.5   100.0
North   120,000   105,000         0    90,000    85.7   100.0
South   120,000   106,000         0    85,000    80.2   100.0
West    120,000   107,000         0    85,000    79.4   100.0
-----  --------  --------  --------  --------  ------  ------
TOTAL   480,000   422,000         0   350,000    82.9   100.0

Highest Unemp%:100.0, City: ██████████
Lowest  Unemp%:100.0, City: [   C   ]
```

[Test case 4]

After the changes mentioned in test case 3, test case 1 is tested again. Because the program logic is changed, the program outputs the following report:

```
City   TotalPop  AdultPop    Employ  Unemploy  Labor%  Unemp%
-----  --------  --------  --------  --------  ------  ------
East    120,000   104,000    90,000    10,000    96.2    10.0
North   120,000   105,000    90,000    10,000    95.2    10.0
South   120,000   106,000    85,000    15,000    94.3    15.0
West    120,000   107,000    85,000    15,000    93.5    15.0
-----  --------  --------  --------  --------  ------  ------
TOTAL   480,000   422,000   350,000    50,000    94.8    12.5

Highest Unemp%: 15.0, City: ██████████
Lowest  Unemp%: 10.0, City: [   D   ]
```

Incidentally, the phenomenon mentioned above can be resolved differently by replacing the initial values on line X with, for example, the following values:

```
REAL: UnempRateHigh ← [   E   ], UnempRateLow ← [   F   ]
```

Answer group for A through D

  a) ????   b) East   c) North

  d) South   e) West

Answer group for E and F

  a) -0.1   b) 0.1   c) 99.9   d) 100.1

Q6. Read the following description of programs, and then answer Subquestions 1 and 2.

An operating system needs to manage the dynamic assignment of memory for processes. Figure 1 shows an example of memory allocation status. There are 32 contiguous memory blocks numbered 0 to 31. For example, memory blocks 0 to 3 are allocated to process R, and memory blocks 4 to 6 are free. In the Figures, ▢ indicates free memory blocks.
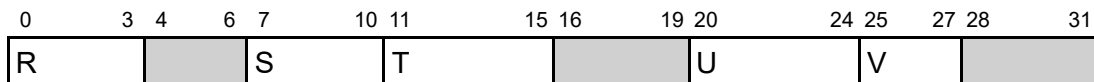

Figure 1 Example of memory allocation status

One way to track the memory allocation status is to maintain a linked list. The linked list in Figure 2 represents the memory allocation status presented in Figure 1. The nodes are kept sorted by starting memory block number.

In Figure 2, sentinels (dotted nodes) are placed at both ends such that any actual node can be handled as an intermediate node, even if it is the first or the last node. The shaded parts are unchangeable by its nature, and "wall" indicates the states of the sentinels are "allocated".
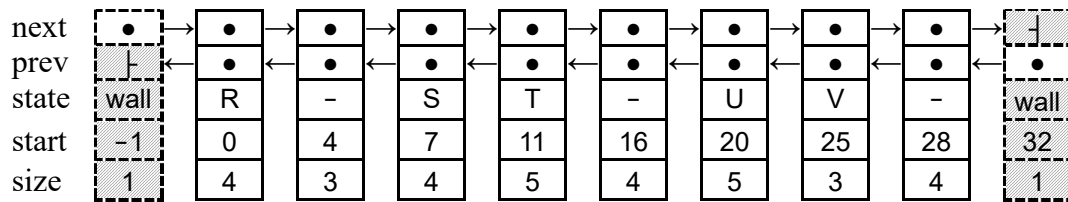

Figure 2 Linked list that represents the memory allocation status in Figure 1

[Program Description]

The program manages the memory shown in Figure 1 with the linked list shown in Figure 2. The program performs the following two kinds of operations:

(i) When a new process starts, allocate the requested size of contiguous memory blocks,

(ii) When an executing process terminates, release the occupied memory blocks.

To handle the nodes, the data type Node is introduced as:

```
Node: node {Node:    next,    // next node
            Node:    prev,    // previous node
            STRING:  state,   // process name (allocated) or "-" (free)
            INT:     start,   // starting memory block number
            INT:     size     // size of memory blocks
           }
```

The program uses four functions: `createNode`, `removeNode`, `allocate`, and `release`. The linked list is maintained by two major functions `allocate` and `release`. The functions `createNode` and `removeNode` are called from `allocate` and `release` respectively.

(1) FUNCTION: `createNode()`

Function `createNode` creates a new node and returns it.
The following statements create a new node and name it `new`.

```
Node: new
new ← createNode();
```

After creating the new node, the program must insert it into the linked list by setting the node pointers of concerned nodes.

(2) FUNCTION: `removeNode(Node: remove)`

Function `removeNode` removes the node `remove` given by the argument.
The following statements remove the node named `garbage`.

```
Node: garbage
removeNode(garbage);
```

Before removing the node, the program must cut off the linkage to/from it by setting the node pointers of concerned nodes.

(3) FUNCTION: `allocate(Node: free, STRING: procName, INT: reqSize)`

Function `allocate` allocates memory blocks of size `reqSize` to the process `procName` from the free memory blocks held by node `free`.
When this function is called, it is ensured that the following conditions are satisfied:
(i) The state of node `free` is "free". Namely, `free.state = "-"`.
(ii) Node `free` holds enough free memory blocks. Namely, `free.size ≥ reqSize`.

In this function, two cases (cases (a) and (b) in Figure 3) should be considered.
Case (a): `free.size = reqSize`.

The entire memory blocks held by the node `free` are allocated to the process. The number of nodes in the linked list remains unchanged.

Case (b): `free.size > reqSize`.

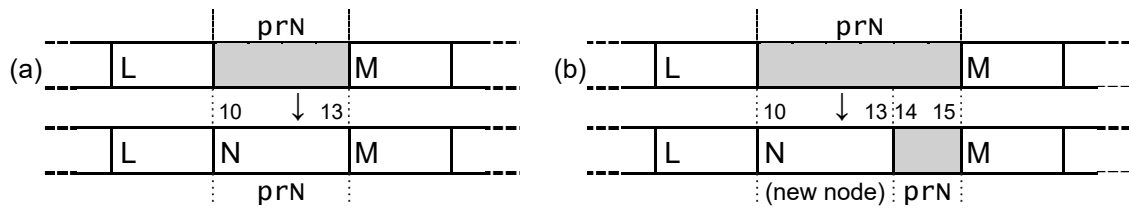The node `free` must be divided into two nodes. Consequently, the number of nodes increases by 1.

Figure 3 Two cases where the function is called as `allocate(prN, "N", 4);`

[Program allocate]

```
FUNCTION: allocate(Node: free, STRING: procName, INT: reqSize) {
    Node: last, new

    IF (free.size = reqSize) {
        [    A    ];
    }
    ELSE {  /* free.size > reqSize */
        new ← createNode();
        last ← free.prev;

        last.next ← new;
        new.next ← free;
        [    B    ];
        [    C    ];

        new.state ← procName;
        new.start ← free.start;
        new.size ← reqSize;
        free.start ← free.start + reqSize;
        free.size ← [    D    ];
    }
}
```

(4) `FUNCTION: release(Node: proc)`

Function `release` releases the memory blocks held by node `proc`. The memory blocks occupied by the process in node `proc` become free memory blocks.

When this function is called, the following condition is satisfied:

The state of node `proc` is "allocated". Namely, `proc.state ≠ "-"`.

In this function, four cases (cases (a) to (d) in Figure 4) should be considered.

Case (a): The states of previous and next nodes are "allocated". After changing node `proc` to "free", the number of nodes in the linked list remains unchanged.

Cases (b) and (c): The state of the previous node is "allocated" and the state of the next node is "free", or vice versa. After changing node `proc` to "free", two consecutive "free" nodes must be combined into a single "free" node. Therefore, the number of nodes reduces by 1.

Case (d): The states of previous and next nodes are "free". After changing node `proc` to "free", three consecutive "free" nodes must be combined into a single "free" node. Therefore, the number of nodes reduces by 2.
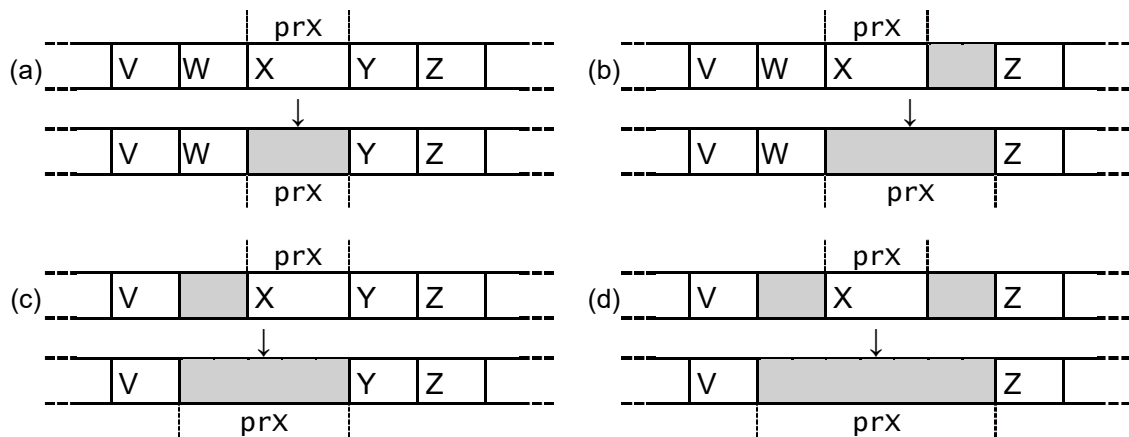

Figure 4　Four cases where the function is called as `release(prX);`

[Program `release`]

```
FUNCTION: release(Node: proc) {
    Node: next1, next2, prev1, prev2

    proc.state ← "-";
    next1 ← proc.next;
    IF (next1.state = "-") {   /* Combine two consecutive "free" nodes */
        next2 ← next1.next;
        proc.next ← next2;
        next2.prev ← proc;
        proc.size ← proc.size + next1.size;
        removeNode(next1);
    }
    prev1 ← proc.prev;
    IF (prev1.state = "-") {   /* Combine two consecutive "free" nodes */
        prev2 ← prev1.prev;
            E      ;
            F      ;
        proc.start ← prev1.start;
        proc.size ←    G     ;
        removeNode(prev1);
    }
}
```

Subquestion 1

From the answer groups below, select the correct answer to be inserted in each blank ☐ in the above programs.

Answer group for A

   a) `/* Do nothing */`   b) `free.start ← free.start + reqSize`

   c) `free.state ← "-"`   d) `free.state ← procName`

Answer group for B and C

   a) `free.prev ← last`   b) `free.prev ← new`

   c) `last.prev ← free`   d) `last.prev ← new`

   e) `new.prev ← free`   f) `new.prev ← last`

Answer group for D

   a) `free.size - reqSize`   b) `free.size - reqSize - 1`

   c) `reqSize`   d) `reqSize - 1`

Answer group for E and F

   a) `prev1.next ← proc`   b) `prev1.prev ← prev2`

   c) `prev2.next ← prev1`   d) `prev2.next ← proc`

   e) `proc.prev ← prev1`   f) `proc.prev ← prev2`

Answer group for G

   a) `prev1.size`   b) `prev1.size + next1.size`

   c) `prev2.size`   d) `prev2.size + next1.size`

   e) `proc.size + prev1.size`   f) `proc.size + prev2.size`

Subquestion 2

From the answer group below, select the correct answer to be inserted in each blank [   ] in the following description.

Figure 5 shows the current memory allocation status, and the linked list in Figure 6 represents the memory allocation status presented in Figure 5. It is assumed that the first three nodes in Figure 6 can be referred to by node names n0, n1, and n2.
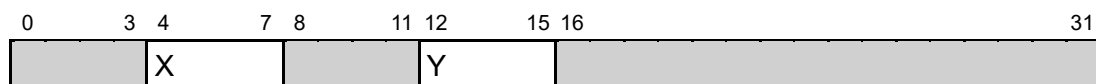
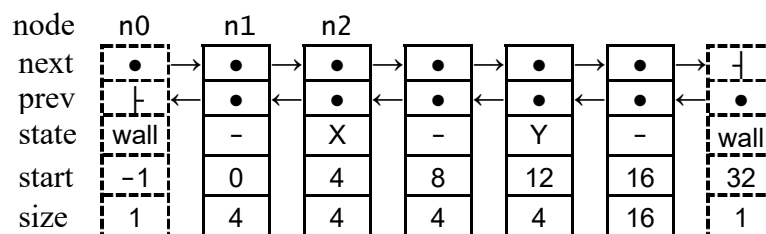

Figure 5 Current memory allocation status

Figure 6 Linked list that represents the memory allocation status in Figure 5

Among the following three cases:

Case 1: Function allocate is called as allocate(n1, "z", 2);
Case 2: Function allocate is called as allocate(n1, "z", 4);
Case 3: Function release is called as release(n2);

the case(s) that update(s) the contents of n0.next is(are) [H].

Answer group for H

a) case 1 only b) case 2 only c) case 3 only
d) case 1 and case 2 e) case 1 and case 3 f) case 2 and case 3

Q7. Read the following description of a C program, and then answer Subquestions 1 and 2.

A palindrome is a word or other sequence of characters that read the same backward as forward, such as *anna, madam,* or *stats.*

There are also numeric palindromes that can read the same forward and backward, such as 353, 787, and 2332, including numbers with the same digits, such as 22, 555, and 8888. In most cases, a natural number becomes a palindrome number when the "reverse and add" process is continuously applied. The process is called the "Lychrel process". For example, 13 becomes a palindrome number when 31, which is the reverse of 13, is added. Similarly, 754 becomes a palindrome number when the Lychrel process is applied twice: 754 + 457 = 1211; 1211 + 1121 = 2332; and 255 becomes a palindrome number when the Lychrel process is applied three times: 255 + 552 = 807; 807 + 708 = 1515; 1515 + 5151 = 6666. However, it is unknown for some numbers how many steps will be taken to become a palindrome number with the Lychrel process. These numbers are 196, 295, and so on.

[Program Description]

The program reads a non-negative number of less than or equal to 10 digits and tries to make a palindrome from the number. Finally, the program outputs a message that tells whether the number is originally a palindrome or becomes a palindrome with the Lychrel process within the specified number of attempts.

The program uses a character string to represent a number. The string has enough size for applying at most 20 times reverse-and-add process to the given number. Each element of the string stores a numerical character corresponding to a digit. The first element stores the highest digit of the number and the last element stores the NUL character (`'\0'`). Figure 1 shows how the number 2023 is represented in a character string.

Index	0	1	2	3	4
Character string	`'2'`	`'0'`	`'2'`	`'3'`	`'\0'`

Figure 1 Representation of the number 2023 in a character string

The program consists of the following six functions:

(1) `int isNumber(char *str)`

   Returns 1 if the string consists of numerical characters only; otherwise, returns 0.

- 30 -

(2) `void reverse(char *rev, const char *str)`

Reverses the contents of the string `str` and stores the result into the string `rev`. For example, calling `reverse(x, y)`, where y is the string "12345", results in x being "54321" with y kept unchanged. Both arguments shall not overlap.

(3) `int isPalindrome(const char *str)`

(The explanation is omitted.)

(4) `void add(char *addend, const char *adder)`

Adds two numbers represented by the strings. For example, adding "12345" and "98765" results in "111110". Both arguments shall be strings of the same length. The result of the addition is stored back to the string `addend`.

(5) `int doLychrelProcess(char *strNum, int maxAttempts)`

Calculates the number of attempts until the number takes to get to a palindrome number using reverse-and-add process. The argument `strNum` holds the number represented by a character string, and `maxAttempts` holds the maximum count of attempts. Returns the actual count of attempts it made, or returns –1 if it cannot determine whether the number becomes a palindrome under the maximum count of attempts.

(6) `int main()`

(The explanation is omitted.)

The program uses the following standard library function.

`size_t strlen(const char *str)`

Returns the length of the string `str`. The length does not contain the terminating NUL character (`'\0'`).

The example outputs of the program are as follows.

(Example 1)
```
Enter a number: 97
[1] 97 + 79 = 176
[2] 176 + 671 = 847
[3] 847 + 748 = 1595
[4] 1595 + 5951 = 7546
[5] 7546 + 6457 = 14003
[6] 14003 + 30041 = 44044
The number becomes a palindrome within 6 steps.
```

(Example 2)
```
Enter a number: 98
[1] 98 + 89 = 187
[2] 187 + 781 = 968
[3] 968 + 869 = 1837
[4] 1837 + 7381 = 9218
[5] 9218 + 8129 = 17347
[6] 17347 + 74371 = 91718
    ⋮  (omitted)
[15] 664272356 + 653272466 = 1317544822
[16] 1317544822 + 2284457131 = 3602001953
[17] 3602001953 + 3591002063 = 7193004016
[18] 7193004016 + 6104003917 = 13297007933
[19] 13297007933 + 33970079231 = 47267087164
[20] 47267087164 + 46178076274 = 93445163438
The number does not become a palindrome within 20 steps.
```

[Program]
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_ATTEMPTS 20
#define MAX_STRLEN (MAX_ATTEMPTS+11)

int isNumber(char *);
void reverse(char *, const char *);
int isPalindrome(const char *);
void add(char *, const char *);
int doLychrelProcess(char *, int);
                                          /*** α ***/
int isNumber(char *str) {
    int i;
    int len = strlen(str);
    for (i = 0; i < len; i++) {
        if (     A     ) {
            return 0;
        }
    }
    return len != 0;
}
```

```c
void reverse(char *rev, const char *str) {
    int i, j;
    int len = strlen(str);
    for (        B        ) {
        rev[j] = str[i];
    }
    rev[j] = '\0';
}


int isPalindrome(const char *str) {
    int i;
    int len = strlen(str);
    for (i = 0; i < len / 2; i++) {
        if (str[i] !=         C        ) {
            return 0;
        }
    }
    return len != 0;
}


void add(char *addend, const char *adder) {
    int i, digit;
    int carry = 0;
    int len = strlen(addend);
    for (i = len - 1; i >= 0; i--) {
        digit = addend[i] - '0' + adder[i] - '0' + carry;
        addend[i] =        D        + '0';
        carry =        E        ;
    }
    if (carry) {
        for (i = len + 1; i > 0; i--) {
            addend[i] = addend[i - 1];
        }
        addend[0] = carry + '0';
    }
}
                                        /*** β ***/
```

```c
int doLychrelProcess(char *strNum, int maxAttempts) {
    int numAttempts = 0;
    char strRev[MAX_STRLEN];
    while (!isPalindrome(strNum) && ++numAttempts <= maxAttempts) {
        reverse(strRev, strNum);
        printf("[%d] %s + %s = ",
                numAttempts, strNum, strRev);                    /*** γ ***/
        add(strNum, strRev);
        printf("%s\n", strNum);
    }
    if (numAttempts <= maxAttempts) {
        return numAttempts;
    }
    return -1;
}

int main() {
    char strNum[MAX_STRLEN];
    int steps = 0;
    printf("Enter a number: ");
    scanf("%10s", strNum);
    if (!isNumber(strNum)) {
        printf("Please input a non-negative integer.\n");
        return 0;
    }
                                                                 /*** δ ***/
    steps = doLychrelProcess(strNum, MAX_ATTEMPTS);
    if (steps ==    F   ) {
        printf("The number does not become a palindrome within "
                "%d steps.\n", MAX_ATTEMPTS);
    }
    else if (steps == 0) {
        printf("The number is a palindrome number.\n");
    }
    else {
        printf("The number becomes a palindrome within "
                "%d steps.\n", steps);
    }
    return 0;
}
```

Subquestion 1

From the answer groups below, select the correct answer to be inserted in each blank
[   ] in the program.

Answer group for A

  a) `str[i] < '0' && str[i] > '9'`

  b) `str[i] < '0' || str[i] > '9'`

  c) `str[i] > '0' && str[i] < '9'`

  d) `str[i] > '0' || str[i] < '9'`

Answer group for B

  a) `i = 0, j = len; i < len - 1; i++, j--`

  b) `i = 0, j = len; i < len; i++, j--`

  c) `i = len - 1, j = 0; i >= 0; i--, j++`

  d) `i = len, j = 0; i >= 0; i--, j++`

Answer group for C

  a) `str[i / 2]`   b) `str[len - 1 - i]`

  c) `str[len - 1]`   d) `str[len - i]`

  e) `str[len]`

Answer group for D and E

  a) `digit % 10`   b) `digit & 1`

  c) `digit * 10`   d) `digit / 10`

  e) `digit >> 1`

Answer group for F

  a) `-1`   b) `0`

  c) `1`   d) `MAX_ATTEMPTS`

Subquestion 2

From the answer groups below, select the correct answer to be inserted in each blank [   ] in the following description. Here, it is assumed that the correct answers are inserted in the blanks [A   ] to [F   ] in the program.

The program handles the number with leading zeros differently than the same number without leading zeros. For example, the program output for input 0097 differs from the output for 97.

```
Enter a number: 0097
[1] 0097 + 7900 = 7997
The number becomes a palindrome within 1 steps.
```

A new function named `skipLeadingZeros` is added to the program according to the following instructions such that the output for a number is unique even if there are leading zeros or not.

(1) Insert the following line at the line indicated by /*** α ***/.

```
const char *skipLeadingZeros(const char *);
```

(2) Insert the following 9 lines at the line indicated by /*** β ***/.

```
const char *skipLeadingZeros(const char *str) {
    while (*str == '0') {
        str++;
    }
    if (*str == '\0') {
        return     G     ;
    }
    return str;
}
```

This function returns the pointer to the character that is the first occurrence of non-zero character. If `str` consists of zeros only, returns the pointer to the last zero character.

(3) Replace the line indicated by /*** γ ***/ with the following line.

```
numAttempts, strNum, skipLeadingZeros(strRev));
```

(4) Insert the following line at the line indicated by /*** δ ***/.

```
memmove(    H    );
```

Here, memmove is the standard library function defined as follows:

```
void *memmove(void *dest, void *src, size_t len)
```

This function copies the character string of length len from src to dest, where src and dest may overlap each other.

Answer group for G

a) `*str`   b) `str + 1`

c) `str - 1`   d) `NULL`

Answer group for H

a) `skipLeadingZeros(strNum), strNum, strlen(strNum)`

b) `skipLeadingZeros(strNum), strNum, strlen(strNum) + 1`

c) `strNum, skipLeadingZeros(strNum), strlen(skipLeadingZeros(strNum))`

d) `strNum, skipLeadingZeros(strNum), strlen(skipLeadingZeros(strNum)) + 1`

Q8. Read the following description of Java programs, and then answer Subquestions 1 and 2.

[Program Description]

A bag is a finite collection of objects in no particular order. A bag can contain duplicate items.

A data structure bag can perform the following tasks:

i)   Get the number of items currently in the bag

ii) Add a given item in the bag

iii) Count the number of times a certain item occurs in the bag

iv) Check whether the bag is full

v)   Check whether the bag is empty

Consider a bag data structure implemented by the interface `Bag`.

Table 1 summarizes the descriptions of the five methods.

Table 1 Descriptions of the five methods

Method	Description
`int size()`	Get and return the number of items currently in the bag.
`boolean add(T entry)`	Add an entry to the bag. Return `true` if the entry is successfully added; otherwise, return `false`.
`int count(T entry)`	Count and return the number of occurrences of the specified entry among the elements in the bag.
`boolean isFull()`	Check whether the bag is full. Return `true` if the bag is full; otherwise, return `false`.
`boolean isEmpty()`	Check whether the bag is empty. Return `true` if the bag is empty; otherwise, return `false`.

Program 1 represents the interface `Bag`. `T` is a generic type.

Program 2 represents the class `ArrayBag` that implements the interface `Bag`. There are two constructors: (1) initialized with a default capacity of 25 and (2) initialized with the passing capacity. The methods presented in Table 1 are implemented in Program 2.

[Program 1]
```
public interface Bag<T> {
    public int size();
    public boolean add(T entry);
    public int count(T entry);
    public boolean isFull();
    public boolean isEmpty();
}
```

[Program 2]

```java
public class    A    implements Bag<T> {
    private final    B    bag;
    private int numberOfEntries = 0;
    private static final int DEFAULT_CAPACITY = 25;

    public ArrayBag() {
        this(DEFAULT_CAPACITY);
    }

    public ArrayBag(int desiredCapacity) {
        @SuppressWarnings("unchecked")
        T[] tempBag = (T[])new Object[desiredCapacity];
        bag = tempBag;
    }

    public int size() {
        return    C    ;
    }

    public boolean add(T entry) {
        if (entry == null) {
            throw new NullPointerException();
        }
        boolean result = true;
        if (isFull()) {
            result = false;
        } else {
            bag[    D    ] = entry;
        }
        return result;
    }

    public int count(T entry) {
        int counter = 0;
        for (T item : bag) {
            if (entry.equals(item)) {
                counter++;
            }
        }
        return counter;
    }
```

```
        public boolean isFull() {
            return   C   >=   E  ;
        }

        public boolean isEmpty() {
            return   C   ==   F  ;
        }
    }
```

Subquestion 1

From the answer groups below, select the correct answer to be inserted in each blank
[   ] in Program 2.

Answer group for A and B

   a) `ArrayBag`   b) `ArrayBag<T>`   c) `ArrayBag<T[]>`

   d) `Bag`   e) `Bag<T>`   f) `Bag<T[]>`

   g) `T`   h) `T[]`

Answer group for C through F

   a) `0`   b) `1`

   c) `bag.length`   d) `bag.length - 1`

   e) `bag.length++`   f) `numberOfEntries`

   g) `numberOfEntries++`   h) `numberOfEntries--`

Subquestion 2

Four more methods are added to the class `ArrayBag`.

Table 2 summarizes the descriptions of the additional four methods.

Table 2 Descriptions of the additional four methods

Method	Description
`int indexOf(T entry)`	Find and return the first index of the specified `entry` in the bag. If the `entry` does not exist, return `-1`.
`T get(int index)`	Return the element at the position specified by the argument.
`T remove(int index)`	Remove the element at the position specified by the argument. If the element is successfully removed, return the removed value. Otherwise, return `null`. If the removed element is not the last element in the bag, move the last element to the removed position and set the last element to `null`.
`boolean removeEntry(T entry)`	Find and remove the first occurrence of the entry and return `true`. Otherwise, return `null`. If the removed element is not the last element in the bag, move the last element to the removed position and set the last element to `null`.

Program 3 represents the added four methods in the class `ArrayBag`.

[Program 3]
```java
    public int indexOf(T entry) {
        for (int i = 0; i < numberOfEntries; i++) {
            if (entry.equals(bag[i])) {
                return i;
            }
        }
        return -1;
    }

    public T get(int index) {
        if (index < 0 || index >= numberOfEntries) {
            throw new ArrayIndexOutOfBoundsException();
        }
        return bag[index];
    }
```

- 41 -

```
public T remove(int index) {
    T result = null;
    if (index >= 0 && index < numberOfEntries) {
        result =    G    ;
          G    =    H    ;
          H    = null;
        numberOfEntries--;
    }
    return result;
}

public boolean removeEntry(T entry) {
    T result = remove(indexOf(entry));
    return entry.equals(result);
}
```

From the answer group below, select the correct answer to be inserted in each blank
[   ] in Program 3.

Answer group for G and H

a) bag[bag.length - 1]   b) bag[bag.length]

c) bag[index - 1]   d) bag[index]

e) bag[numberOfEntries - 1]   f) bag[numberOfEntries]