October 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.

Question **Q1** is **compulsory**.

Q1. Read the following description of an exchange of cryptographic keys, and then answer Subquestions 1 and 2.

As a method of exchanging cryptographic keys in an insecure network, the Diffie-Hellman algorithm allows two users to securely exchange a key that can be used for subsequent symmetric encryption of messages.

This algorithm uses modulo operations. The modulo operation "*a* mod *b*" calculates the remainder when integer *a* is divided by integer *b*. For example, 29 mod 5 is 4, and 21 mod 3 is 0.

To establish a shared secret key, the two users must first select two numbers: a prime number *p* and an integer *g* such that the value of g^n mod *p* does not overlap in the range $1 \le n \le p-1$. In other words, each value of *n* ($n = 1, 2, \ldots, p-1$) appears in g^n mod *p*. Table 1 shows an example of such an integer pair: $p = 11$ and $g = 2$. The bottom row shows the numbers 1, 2, …, 10.

Table 1 An example of g^n mod *p* with an integer pair: $p = 11$ and $g = 2$

n	1	2	3	4	5	6	7	8	9	10
2^n	2	4	8	16	32	64	128	256	512	1024
2^n mod 11	2	4	8	5	10	9	7	3	6	1

The Diffie-Hellman algorithm is shown in Figure 1 and in the following steps:

(1) Alice and Bob select a prime number *p* and an integer *g*, as explained above.
(2) Alice selects a random number, *x* ($1 \le x \le p-1$), and calculates R1 = g^x mod *p*.
(3) Bob selects another random number, *y* ($1 \le y \le p-1$), and calculates R2 = g^y mod *p*.
(4) Alice sends R1 to Bob. Note that Alice does not send the value of *x*.
(5) Bob sends R2 to Alice. Note that Bob does not send the value of *y*.
(6) Alice calculates the shared secret key K = $(\mathrm{R}2)^x$ mod *p*.
(7) Bob calculates the shared secret key K = $(\mathrm{R}1)^y$ mod *p*.

Thus, Alice and Bob obtain the shared secret key K for the session.

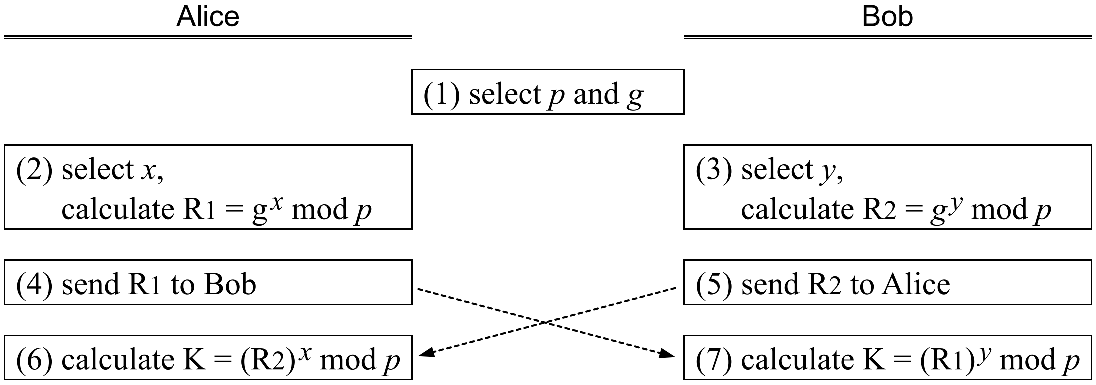

Alice
Bob
(1) select p and g
(2) select x, calculate R1 = g^x mod p
(3) select y, calculate R2 = g^y mod p
(4) send R1 to Bob
(5) send R2 to Alice
(6) calculate K = $(R2)^x$ mod p
(7) calculate K = $(R1)^y$ mod p

Figure 1 Diffie-Hellman key exchange algorithm

Subquestion 1

From the answer group below, select the correct answer to be inserted in each blank $\boxed{}$ in the following description.

A shared secret key is being exchanged between Alice and Bob using the Diffie-Hellman algorithm. Assume that they agreed upon the prime number p = 11 and the integer g = 2.
If Bob has received the public key R1 = 3 from Alice, the random number x Alice has selected is $\boxed{\text{A}}$. Also, if Alice has received the public key R2 = 9 from Bob, the random number y Bob has selected is $\boxed{\text{B}}$.
In this case, i.e., R1 = 3 and R2 = 9, the shared secret key obtained by both Alice and Bob is $\boxed{\text{C}}$, which they can use for subsequent symmetric encryption of messages.

Answer group

a) 1	b) 2	c) 3	d) 4	e) 5
f) 6	g) 7	h) 8	i) 9	j) 10

Subquestion 2

From the answer groups below, select the correct answer to be inserted in each blank $\boxed{}$ in the following description.

Alice and Bob decided to change the values of *g*, *x*, and *y*. Then, they exchanged the re-calculated R1 and R2, and finally obtained the shared secret key K.
Assume that an attacker Eve knows the three non-secret values: $p = 11$, $g = 7$, and R1 = 3. Recently, Eve obtained the value $y = 3$ by illegal means. Then, it would be possible for Eve to determine the shared secret key K as $\boxed{\quad \text{D} \quad}$.

In this case, *p* is sufficiently small in that it is possible to find x from R1 or y from R2 in a short time. However, since an efficient way to perform this calculation has not been discovered, finding x from R1 requires a brute-force search for x such that $g^x \bmod p$ = R1. Assuming that 2^{90} attempts to search for x can be made in one year, it would take approximately $\boxed{\quad \text{E} \quad}$ years to search for x if p of about 2^{120} in size is used.
For practical use, a prime number with a size of 2^{2048} ($\approx 3.2 \times 10^{616}$) or larger is used as *p*.

Answer group for D

a) 1 b) 2 c) 3 d) 4 e) 5
f) 6 g) 7 h) 8 i) 9 j) 10

Answer group for E

a) 2 b) 5 c) 10 d) 100
e) 1000 f) 1 million g) 1 billion

Concerning questions **Q2** through **Q5**, **select two** of the four questions. For each selected question, mark the Ⓢ in the selection area on the answer sheet, and answer the question. If three or more questions are selected, only the first two questions will be graded.

Q2. Read the following description of direct-mapped cache memory configuration, and then answer Subquestion.

In this question, letters in *italics*, such as *001* and *ttt* indicate binary digits, and letters in Gothic, such as 0F and xx indicate hexadecimal digits.

Cache memory is placed between the CPU and the main memory. Cache memory is designed to provide fast access to selected data from the main memory. Since the size of cache memory is much smaller than the size of main memory, the large amount of data required for a process cannot be stored in cache memory simultaneously.

[Direct-mapped cache]

A mapping is required when copying data from main memory to cache memory. A block of main memory maps to only one predetermined location in cache memory in a direct-mapped cache method, which is one of the common types of mapping functions.

Consider a simple memory model:

Main memory: 256 bytes in size, addressed 0 to 255. Each address holds 1-byte data.

Cache memory: 32 bytes (= 8 words) in size for data.

Data transfer between main memory and cache memory: In the unit of words.

In this model, a word is defined as 4 contiguous bytes in main memory starting at an address that is a multiple of 4.

Figure 1 shows the relationship between main memory and cache memory.

An 8-bit main memory address is divided into three parts: tag-bits (3 bits), line-bits (3 bits), and byte-bits (2 bits). Tag-bits *ttt* is used to identify the main memory address in the cache. Line-bits *LLL* is used for mapping; a word in main memory whose address is *aaaLLL00* (*a*: any) is always mapped to the word at line position *LLL* in the cache. Byte-bits *bb* indicate byte offset in a word.

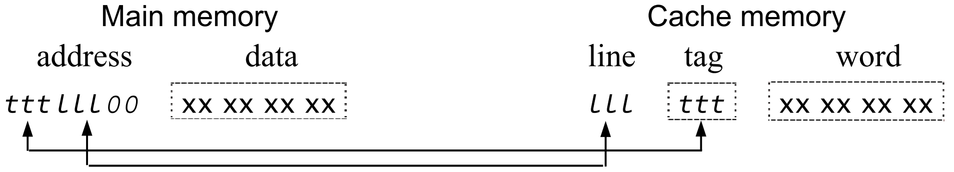
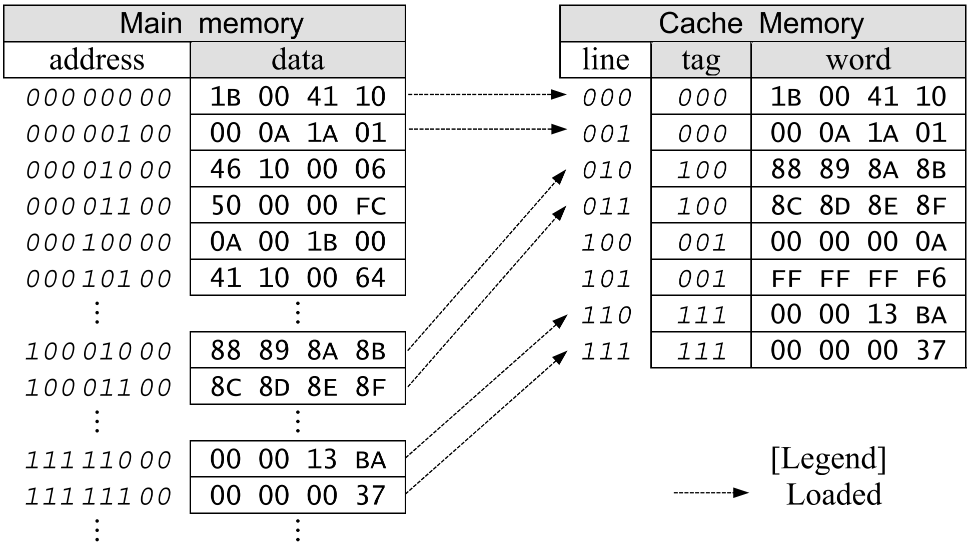


Figure 1 Relationship between main memory and cache memory

When the CPU is going to read a byte at address *tttLLLbb* that is not loaded into the cache memory, the processor transfers the word that contains the target byte from the main memory to the line location *LLL* in the cache memory with the tag-bits *ttt*.

Figure 2 shows the simple memory model explained above with sample data.
For example, when the CPU is going to read the byte 1B at address 18, $\boxed{\text{A}}$ in the cache memory.
The processor can identify the main memory address of a word in the cache memory using the line location *LLL* and tag-bits *ttt*. For example, the word FF FF FF F6 on line *101* can be identified in that it is loaded from the main memory addresses $\boxed{\text{B}}$.

Whenever the CPU needs data from the main memory, it immediately checks whether the data is loaded into the cache memory. If the data exists in the cache memory, the state is called "cache hit"; otherwise, it is called "cache miss".
Assume that the main memory and cache memory hold the data as shown in Figure 2. When the CPU reads the data successively from addresses 0, 32, 2, and 34, in this order, the sequence of occurrence of the state will be cache hit → cache miss → $\boxed{\text{C}}$.



Main memory	
address	data
000 000 00	1B 00 41 10
000 001 00	00 0A 1A 01
000 010 00	46 10 00 06
000 011 00	50 00 00 FC
000 100 00	0A 00 1B 00
000 101 00	41 10 00 64
⋮	⋮
100 010 00	88 89 8A 8B
100 011 00	8C 8D 8E 8F
⋮	⋮
111 110 00	00 00 13 BA
111 111 00	00 00 00 37
⋮	⋮

Cache Memory		
line	tag	word
000	*000*	1B 00 41 10
001	*000*	00 0A 1A 01
010	*100*	88 89 8A 8B
011	*100*	8C 8D 8E 8F
100	*001*	00 00 00 0A
101	*001*	FF FF FF F6
110	*111*	00 00 13 BA
111	*111*	00 00 00 37

Figure 2 Simple memory model with sample data

Generally, when the main memory address is divided into three parts as follows:

tag-bits	line-bits	byte-bits
← T bits →	← L bits →	← B bits →

the specification of the direct-mapped cache memory configuration can be described as shown in Table 1. It is assumed that each main memory address holds 1-byte data.

Item	Value
Address range of main memory	0 to $\boxed{\text{ D }}$ - 1
Number of words in main memory	2^{T+L}
Number of line locations in cache memory	2^{L}
Cache memory size (for word data part)	2^{L+B} bytes
Cache memory size (for tag-bits part)	$\boxed{\text{ E }}$ bits

Table 1 Specification of direct-mapped cache memory configuration

Subquestion

From the answer groups below, select the correct answer to be inserted in each blank $\boxed{\qquad}$ in the above description.

Answer group for A

a) data transfer does not occur because the word that contains the target byte already exists

b) the word 0A 00 1B 00 and tag-bits *000* are loaded into the line location *100*

c) the word 0A 00 1B 00 and tag-bits *100* are loaded into the line location *000*

d) the word 1B 00 41 10 and tag-bits *000* are loaded into the line location *100*

e) the word 1B 00 41 10 and tag-bits *100* are loaded into the line location *000*

Answer group for B

a) 52 to 55 (34 to 37)

b) 60 to 63 (3C to 3F)

c) 164 to 167 (A4 to A7)

d) 204 to 207 (CC to CF)

Answer group for C

a) cache hit → cache hit

b) cache hit → cache miss

c) cache miss → cache hit

d) cache miss → cache miss

Answer group for D and E

a) $2^{L-1} \times T$

b) $2^{T-1} \times L$

c) $2^{L} \times T$

d) $2^{T} \times L$

e) 2^{T+L-1}

f) 2^{T+L}

g) $2^{T+L+B-1}$

h) 2^{T+L+B}

Concerning questions **Q2** through **Q5**, **select two** of the four questions.

Q3. Read the following description of a thesis management system in a university, and then answer Subquestions 1 through 3.

University T has a variety of faculties. Each faculty has instructors and students.
University T holds a thesis defense meeting and graduation ceremony once a year. There is a topic registration period when students register what they will do as their graduate thesis before the thesis defense meeting is held. Students must contact an instructor in their faculty who will instruct them in their theses. Each instructor can instruct at most 5 students. Each student must be instructed by one instructor. After contacting instructors, students need to consult their instructors to determine their thesis topics. Each student will choose one topic. It is possible for them to either working on their thesis by themselves or in groups of maximum 4 students. Once a student has selected a thesis topic, he/she has to register the topic to the department of education before the registration deadline.

University T would like to build a thesis management system that enables students to register their thesis topic online and checks their grade after defending their thesis. To build this system, university T has created a simplified E-R diagram shown in Figure 1.

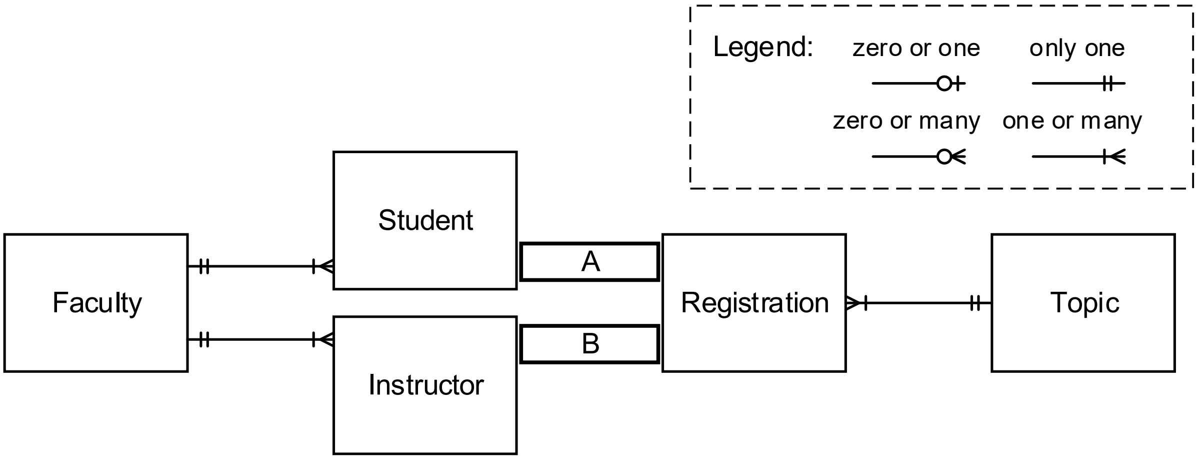


Figure 1 E-R diagram (simplified)

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

Faculty Table

ID	FacultyName
FIT	Faculty of Information Technology
FEE	Faculty of Electronical Engineering

Student Table

ID	StudentName	FacultyID	Email	DefenseTerm
S2101	Sarah	FIT	sarah@example.edu	2023
S2102	Karina	FIT	karina@example.edu	2023
S2103	Mike	FIT	mike@example.edu	2023
S2104	Selena	FEE	selena@example.edu	2023
S2105	Peter	FEE	peter@example.edu	2023
S2201	Roger	FEE	roger@example.edu	2024

Instructor Table

ID	InstructorName	Title	FacultyID
I01	Christopher	Professor	FIT
I02	Lily	Associate Professor	FIT
I03	Marie	Professor	FEE
I04	Robert	Lecturer	FEE

Topic Table

ID	TopicName
T01	Personal Area Networks using Zigbee Technology
T02	Rich Internet Application for Weekly College Timetable Generation
T03	General-purpose Online Ticket Reservation System
T04	Fiber Optic Communication

Registration Table

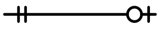
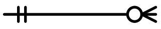
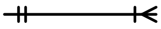
At the time of registration, the Grade field is set to `null`.

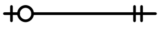
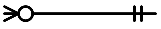
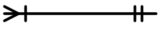
ID	StudentID	TopicID	InstructorID	Grade
R01	S2101	T02	I02	
R02	S2102	T03	I01	
R03	S2103	T03	I01	
R04	S2105	T04	I03	

Subquestion 1

From the answer group below, select the correct answer to be inserted in each blank □ in Figure 1.

Answer group for A and B

a) 
b) 
c) 

d) 
e) 
f) 

Subquestion 2

From the answer groups below, select the correct answer to be inserted in each blank □ in the SQL statement SQL1.

The defense meeting in 2023 will be held in December. The topic registration period has already started and the deadline for registration is getting close.

The department of education is planning to send a reminder email to the students.

The following SQL statement SQL1 outputs the students whose defense term is 2023 and who have not registered thesis topics.

```
-- SQL1 --
SELECT s.ID, s.StudentName, s.Email
FROM Student s [    C    ] Registration r
     ON s.ID = r.StudentID
WHERE s.DefenseTerm = 2023 AND [    D    ]
```

When SQL1 is executed using the sample data shown in the table definitions, SQL1 outputs the following result:

ID	StudentName	Email
S2104	Selena	selena@example.edu

Note:

INNER JOIN returns the values on the matched rows from the left and right tables.

LEFT OUTER JOIN returns the values on all rows from the left table and the matched rows from the right table. Columns of unmatched rows are set to NULL.

RIGHT OUTER JOIN returns the values on all rows from the right table and the matched rows from the left table. Columns of unmatched rows are set to NULL.

Answer group for C

a) `INNER JOIN`   b) `LEFT OUTER JOIN`

c) `RIGHT OUTER JOIN`

Answer group for D

a) `r.ID IS NOT NULL`   b) `r.ID IS NULL`

c) `s.ID IS NOT NULL`   d) `s.ID IS NULL`

Subquestion 3

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

At the thesis defense meeting, committees assess and mark students' theses. After the meeting, these grades are put into the Registration Table.
The following table shows the current content of the Registration table. It is assumed that four tables other than the Registration Table remain unchanged.

Registration Table

ID	StudentID	TopicID	InstructorID	Grade
R01	S2101	T02	I02	85
R02	S2102	T03	I01	90
R03	S2103	T03	I01	90
R04	S2105	T04	I03	80
R05	S2104	T01	I04	85

University T gives excellent thesis awards to the students at the graduation ceremony. The department of education determines the award receivers using the SQL statements SQL2 (not shown) and SQL3.
First, SQL2 (not shown) modifies the Registration Table by removing the ID, TopicID, and InstructorID columns, adding the StudentName and FacultyID columns, and sorting rows by StudentID. Then SQL2 outputs the result as the Result Table as follows:

Result Table

StudentID	StudentName	FacultyID	Grade
S2101	Sarah	FIT	85
S2102	Karina	FIT	90
S2103	Mike	FIT	90
S2104	Selena	FEE	85
S2105	Peter	FEE	80

Next, SQL3 determines the students who will receive the excellent thesis award from the Result Table created by SQL2 and Faculty Table.

```
-- SQL3 --
SELECT f.FacultyName, st.StudentID, st.StudentName, st.Grade
FROM (SELECT r.FacultyID   AS FacultyID,
             r.StudentID   AS StudentID,
             r.StudentName AS StudentName,
             r.Grade       AS Grade
      FROM Result r
      INNER JOIN (SELECT FacultyID, MAX(Grade) AS MaxGrade
                  FROM Result
                  GROUP BY Result.FacultyID
                 ) AS mg ON r.FacultyID = mg.FacultyID
                      AND r.Grade = mg.MaxGrade
     ) AS st
INNER JOIN Faculty f ON f.ID = st.FacultyID
```

When SQL3 is executed using the sample data shown above, SQL3 determines E as the excellent thesis award receiver.

Answer group for E

a) one student whose student ID is S2102
b) two students whose student ID’s are S2102 and S2103
c) two students whose student ID’s are S2102 and S2104
d) three students whose student ID’s are S2102, S2103, and S2104
e) three students whose student ID’s are S2102, S2104, and S2105
f) four students whose student ID’s are S2102, S2103, S2104, and S2105

Concerning questions **Q2** through **Q5**, **select two** of the four questions.

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

Company U is a small-sized travel agent. It has two departments: Sales and Support. Figure 1 shows the network topology of Company U's internal network with two LANs. Sales department has a Sales LAN with 4 PCs (Sales1, Sales2, Sales3, Sales4) which are connected to the layer-2 switch L2SW#1. Support department has a Support LAN with 3 PCs (Support1, Support2, Support3) which are connected to the layer-2 switch L2SW#2.

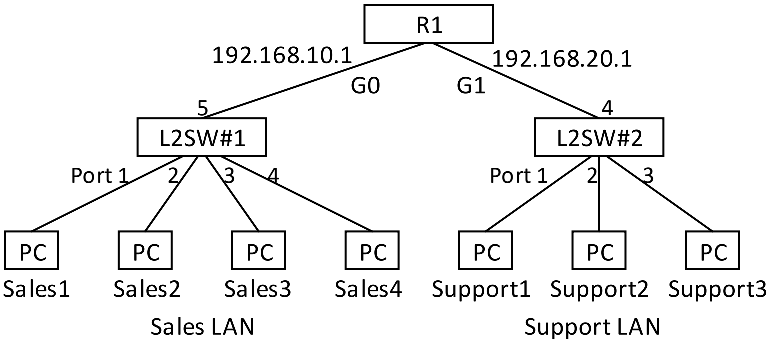


Figure 1 Network topology of Company U's internal network with two LANs

[MAC address table]

A network switch has several ports where PCs may be connected. Each device connected to the switch identifies itself by the layer-2 address, which is the MAC address of the device. Hence, each switch keeps track of the devices connected to it using a mapping table known as the MAC address table. Table 1 shows a typical MAC address table of a switch.

Table 1 Typical MAC address table of a switch

Port	MAC address
1	00:00:5E:00:53:A2
2	00:00:5E:00:53:3B

The MAC address table can be configured manually by the network administrator. Alternatively, a switch can dynamically learn the MAC addresses of the devices connected to its ports. In the case of dynamic learning, the MAC address table is empty at the beginning. The switch does not have any idea which device is connected to which of its ports. Therefore, if the switch receives any frame in port 1, the switch broadcasts/floods the frame to all other ports (except port 1) since it does not have any knowledge regarding the port number of the PC having the destination MAC address.

[Address learning by a switch]

When a frame is received by a switch, the source MAC address is used to populate the MAC address table. For example, if Sales1 PC sends a frame to Sales4 PC, Sales1's source MAC address is mapped with port number 1 of L2SW#1. Thus, the switch's MAC address table is populated based on the source MAC address in a frame. When the destination MAC address of a frame is already learned by the switch, such frame is directed forwarded through the corresponding port number of the switch. Thus, frame forwarding is done based on the MAC address table of the switch.

[ARP request/reply]

When a PC wants to send a frame to another PC in the same LAN, it must produce a layer-2 frame. Figure 2 shows the format of a layer-2 frame.

The sender PC knows its own MAC address. However, the sender may not know the destination MAC address, even though it knows the destination IP address. Therefore, the sender has to resolve the MAC address of the destination PC based on the destination IP address using the ARP protocol.

Destination MAC Address	Source MAC address	Payload

Figure 2 Layer-2 frame format

In the ARP protocol, the source device sends a broadcast message, known as an ARP request message, to find out the MAC address of the destination device. All the devices other than the destination machine ignore this ARP request since it is not intended for them. However, the destination device replies to the source device with a unicast message known as an ARP reply. Thus, the source device obtains the MAC address of the destination device.

If the destination device is in the same LAN, the source device resolves the MAC address of the destination device and sends the frame directly. However, if the destination device is located in another remote LAN, the source device first sends the frame to the default gateway address (which is the interface IP address of the router). Once it is received by the gateway, it then forwards the frame to the next link in a similar fashion.

Table 2 and Table 3 show the IP addresses and MAC addresses of the PCs in Sales-LAN and Support LAN.

The default gateway address of the Sales LAN is 192.168.10.1, and the default gateway address of the Support LAN is 192.168.20.1.

Table 2 IP addresses and MAC addresses of PCs in Sales LAN

PC name	IP address	MAC Address
Sales1	192.168.10.2/24	00:00:5E:00:53:11
Sales2	192.168.10.3/24	00:00:5E:00:53:22
Sales3	192.168.10.4/24	00:00:5E:00:53:33
Sales4	192.168.10.5/24	00:00:5E:00:53:44

Table 3 IP addresses and MAC addresses of PCs in Support LAN

PC name	IP address	MAC Address
Support1	192.168.20.2/24	00:00:5E:00:53:55
Support2	192.168.20.3/24	00:00:5E:00:53:66
Support3	192.168.20.4/24	00:00:5E:00:53:77

Table 4 shows the IP addresses and MAC addresses of the interfaces of the router R1.

Table 4 IP addresses and MAC addresses of router R1

Device Name	Interface	IP address	MAC Address
R1	Gigabit 0 (G0)	192.168.10.1/24	00:00:5E:00:53:AA
	Gigabit 1 (G1)	192.168.20.1/24	00:00:5E:00:53:BB

Subquestion 1

From the answer groups below, select the correct answer to be inserted in each blank $\boxed{}$ in the following description.

Assume that the MAC address table of L2SW#1 is empty. When Sales1 sends a packet to Sales3, L2SW#1 forwards the ARP request to the port number(s) $\boxed{\text{ A }}$.

When Sales1 wants to send a packet to Support2 for the first time, Sales1 first sends $\boxed{\text{ B }}$ to try to find out the $\boxed{\text{ C }}$ of the $\boxed{\text{ D }}$ because Sales1 and Support2 are not in the same network.

Answer group for A

a) 1, 2, 3, and 4   b) 1, 2, 3, 4, and 5   c) 2, 3, 4, and 5
d) 3 only   e) 5 only

Answer group for B

a) Address learning   b) ARP reply
c) ARP request   d) MAC address table

Answer group for C and D

a) default gateway   b) IP address   c) L2SW#1
d) L2SW#2   e) MAC address   f) Sales1
g) Support2

Subquestion 2

From the answer groups below, select the correct answer to be inserted in each blank $\boxed{}$ in the following description. Here, the answers to be inserted in E1 and E2 should be selected as the correct combination from the answer group for E. Also, the answers to be inserted in F1 and F2 should be selected as the correct combination from the answer group for F.

After ARP resolution as performed in Subquestion 1, if Sales1 wants to send a packet to Support1, then the packet will be encapsulated at the data link layer of Sales1 as follows:

Frame Header		IP Header		
Source MAC	Destination MAC	Source IP	Destination IP	Data
00:00:5E:00:53:11	E1	192.168.10.2	E2	Some data

When Support1 receives the packet sent from Sales1, the packet at Support1 will look like as follows:

Frame Header		IP Header		
Source MAC	Destination MAC	Source IP	Destination IP	Data
F1	00:00:5E:00:53:55	F2	192.168.20.2	Some data

Answer group for E

	E1	E2
a)	00:00:5E:00:53:55	192.168.10.1
b)	00:00:5E:00:53:55	192.168.20.1
c)	00:00:5E:00:53:55	192.168.20.2
d)	00:00:5E:00:53:AA	192.168.10.1
e)	00:00:5E:00:53:AA	192.168.20.1
f)	00:00:5E:00:53:AA	192.168.20.2
g)	00:00:5E:00:53:BB	192.168.10.1
h)	00:00:5E:00:53:BB	192.168.20.1
i)	00:00:5E:00:53:BB	192.168.20.2

Answer group for F

	F1	F2
a)	00:00:5E:00:53:11	192.168.10.1
b)	00:00:5E:00:53:11	192.168.10.2
c)	00:00:5E:00:53:11	192.168.20.1
d)	00:00:5E:00:53:AA	192.168.10.1
e)	00:00:5E:00:53:AA	192.168.10.2
f)	00:00:5E:00:53:AA	192.168.20.1
g)	00:00:5E:00:53:BB	192.168.10.1
h)	00:00:5E:00:53:BB	192.168.10.2
i)	00:00:5E:00:53:BB	192.168.20.1

Concerning questions **Q2** through **Q5**, **select two** of the four questions.

Q5. Read the following description of the test design for software, and then answer Subquestions 1 through 3.

Company V, a software development company, is reviewing its testing method to reduce the number of bugs left uncorrected in the programs developed by the company.

[Description of the testing method used in Company V]

Company V primarily tests the programs developed by the company using control flow testing, which is a white box testing method.

Control flow testing focuses on the smallest units that form a program, such as commands, paths, and decision conditions. Test cases and test data are prepared in accordance with the coverage criteria defined during test planning, and then the behavior of the developed programs is checked.

The coverage criteria include statement coverage, where all the statements are executed at least once in the test, and decision condition coverage (hereinafter, branch coverage), where all the paths after all the branches are executed at least once.

Company V uses branch coverage as its coverage criteria.

[Description of the short-cut evaluation]

A decision condition for a branch includes a single condition that evaluates only one condition and a multiple condition that evaluates two or more single conditions combined with “and” or “or”. The following example illustrates single conditions and a multiple condition.

Example:

$$\underbrace{\underbrace{\texttt{(a > b)}}_{\text{Single condition}}\ \texttt{and}\ \underbrace{\texttt{(b} \geq \texttt{0)}}_{\text{Single condition}}}_{\text{Multiple condition}}$$

Usually, when a program is executed, short-cut evaluation is applied to a multiple condition. In short-cut evaluation, single conditions that constitute a multiple condition are evaluated in order of precedence. Once the result of the multiple condition is determined, the remaining single conditions are not evaluated. For example, in the case of a multiple condition in which two single conditions are combined with “and”, if the evaluation result of the first single condition is false, then the evaluation result of the multiple condition is false, regardless of the evaluation result of the second single condition. Therefore, in this case, it is not necessary to evaluate the second single condition.

Subquestion 1

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

Figure 1 shows a sample program to be tested, and Table 1 shows sample test cases for this program. When the program is tested with the test cases according to the testing method used in Company V, the test result reveals that [A] in test case 1 and [B] in test case 2. Here, the short-cut evaluation is applied to multiple conditions.

```
FUNCTION func(INT: x, INT: a, INT: b, INT: c, INT: d) {
   IF (x > 10) {
      func1();
      IF ((a < 10) or (b < 20)) {
         func2();
      } ELSE {
         func3();
      }
      IF ((c > 10) and (d > 10)) {
         func4();
      } ELSE {
         func5();
      }
   }
}
```

Figure 1 Sample program to be tested

Table 1 Sample test cases

	Test data				
Variable	x	a	b	c	d
Test case 1	11	9	19	10	10
Test case 2	11	10	20	11	11

Answer group for A and B

a) b < 20 is not evaluated

b) b < 20 and c > 10 are not evaluated

c) b < 20 and d > 10 are not evaluated

d) c > 10 is not evaluated

e) c > 10 and d > 10 are not evaluated

f) d > 10 is not evaluated

g) all single conditions are evaluated

Subquestion 2

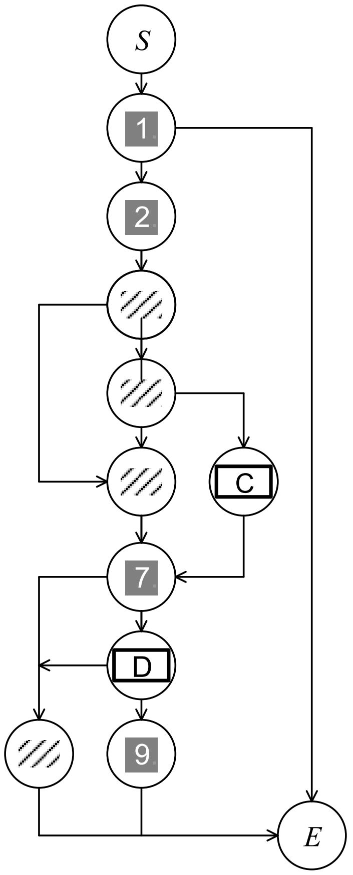
The control structure of a program can be described with a control flow graph. In a control flow graph, processes are divided into serial instructions, iteration instructions, and branch instructions, and each of them is placed in a process block (hereinafter, a node) that is connected with a directed line segment (hereinafter, an edge) in the sequence of process execution. Here, a multiple condition is divided into the respective single conditions, and they are placed in the control flow graph.

Figure 2 is prepared by assigning node numbers 1 through 10 to the sample program in Figure 1. Figure 3 shows the corresponding control flow graph. A circle indicates a node, and an arrow indicates an edge. The node numbers in Figure 3 correspond to the node numbers in Figure 2. Nodes *S* and *E* in Figure 3 are special nodes that indicate the entry and exit of the program, respectively, and there are no corresponding processes in the sample program.

From the answer group below, select the appropriate node number to be inserted in each blank □ in Figure 3.

```
FUNCTION func(INT: x, INT: a, INT: b,
                      INT: c, INT: d) {
   IF 1(x > 10) {
      2func1();
      IF (3(a < 10) or 4(b < 20)) {
         5func2();
      } ELSE {
         6func3();
      }
      IF (7(c > 10) and 8(d > 10)) {
         9func4();
      } ELSE {
         10func5();
      }
   }
}
```

Figure 2 Sample program with node numbers



Note: Shaded parts ▨ are not shown.

Figure 3 Control flow graph

Answer group for C and D

a) 3 b) 4 c) 5

d) 6 e) 8 f) 10

Subquestion 3

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

For the testing of the program shown in Figure 1, in the case of the branch coverage used by Company V, the minimum number of test cases required is E.

Furthermore, there is a method for making test cases by extracting paths from a control flow graph. The minimum number of paths (*P*) that cover all the edges and nodes of a control flow graph is determined with the following expression:

$$P = \text{Number of edges} - \text{Number of nodes} + 2$$

By conducting a test for *P* test cases that correspond to the extracted paths, it is possible to assure higher coverage than branch coverage.

With regard to the control flow graph in Figure 3, the value of *P* is F. To reduce the number of bugs left uncorrected in its programs, Company V decides to test the programs with test cases based on control flow graphs.

Answer group for E and F

a) 2 b) 3 c) 4

d) 5 e) 6 f) 7

Question **Q6** is **compulsory**.

Q6. Read the following description of programs that use the Bitap algorithm to search for a string, and then answer Subquestions 1 through 3.

[Program Description]

The function `BitapMatch` is a program that uses the Bitap algorithm to search for a string. The Bitap algorithm has the characteristic of using a bit sequence that is defined for each individual character in the comparison between a string that is to be searched (hereinafter, target string) and a search string.

In this question, 16-bit Binary type constants in binary are noted with the leading zeroes omitted. For example, for the value 0000000000010101, the leading zeroes are omitted, and the value is noted as "10101"B.

(1) The function `BitapMatch` uses two character arrays: `Text[]`, which contains the target string and `Pat[]`, which contains the search string. Both array indexes start at 1.
The `i`-th character of `Text[]` is noted as `Text[i]` and the `i`-th character of `Pat[]` is noted as `Pat[i]`. The target and search strings are composed of upper-case Roman alphabets only, and both have a maximum length of 16 characters.
Figure 1 shows an example of storage when the target string `Text[]` is "AACBBAACABABAB" and the search string `Pat[]` is "ACABAB".

Element number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`Text[]`	A	A	C	B	B	A	A	C	A	B	A	B	A	B

Element number	1	2	3	4	5	6
`Pat[]`	A	C	A	B	A	B

Figure 1 Example of storage of the target string and the search string

(2) The function `BitapMatch` calls the function `GenerateBitMask`.
For each character from "A" through "Z", the function `GenerateBitMask` generates a bit sequence (hereinafter, bit mask) according to the search string and stores it in the 16-bit Binary type array `Mask[]` with 26 elements.
In `Mask[1]` it stores the bit mask that corresponds to the character "A", in `Mask[2]` it stores the bit mask that corresponds to the character "B", and so on. So, in `Mask[1]` through `Mask[26]`, it stores the bit masks that correspond to the characters "A" through "Z".

The function GenerateBitMask first initializes each element of Mask[] to "0"B; then, for each i that is equal to or greater than 1 and equal to or less than the number of characters in Pat[], it sets 1 to the value of the i-th bit from the lowest position of the value stored in Mask[Index(Pat[i])], which is the element of Mask[] that corresponds to Pat[i].
The function Index returns the integer n (1 ≦ n ≦ 26) when it is called with the *n*-th upper case letter in alphabetical order that has been specified in the argument.

(3) In the example where Pat[] is "ACABAB" as shown in Figure 1, Mask[] becomes as shown in Figure 2 when the function GenerateBitMask is executed.

Mask[1]	0000000000010101	Bit mask that corresponds to "A"
Mask[2]	A	Bit mask that corresponds to "B"
Mask[3]	0000000000000010	Bit mask that corresponds to "C"
Mask[4]	0000000000000000	Bit mask that corresponds to "D"
⋮	⋮	
Mask[26]	0000000000000000	Bit mask that corresponds to "Z"

Figure 2 Value of Mask[] that corresponds to Pat[] shown in Figure 1

(4) The specifications of the arguments and the return value of the function GenerateBitMask are shown in Table 1.

Table 1 Specifications of the arguments and return value of the function GenerateBitMask

Arguments / return value	Data type	Input / Output	Description
Pat[]	CHAR	Input	One-dimensional array that stores the search string
Mask[]	16-bit Binary	Output	One-dimensional array that stores the bit masks that correspond to the characters "A" through "Z"
Return value	INT	Output	Number of characters in the search string

(5) In the programs, the following three operators "|", "&", and "<<" are used:

a | *b* obtains the 16-bit logical sum (OR) for each pair of bits in the corresponding bit positions of two 16-bit binary data, *a* and *b*.

a & *b* obtains the 16-bit logical product (AND) for each pair of bits in the corresponding bit positions of two 16-bit binary data, *a* and *b*.

a << *b* executes a logical left shift of the 16-bit binary data *a* by *b* bits.

[Program 1]

```
FUNCTION: GenerateBitMask(CHAR: Pat[], 16-bit Binary: Mask[]) {
   INT: i, PatLen

   PatLen ← number of characters in Pat[];
   FOR (i ← 1; i ≤ 26; i ← i + 1) {
      Mask[i] ← [   B   ];      /* Initialize */
   }
   FOR (i ← 1; i ≤ PatLen; i ← i + 1) {
      Mask[Index(Pat[i])] ← ([   C   ]) | Mask[Index(Pat[i])];
   }
   return (PatLen);
}
```

Subquestion 1

From the answer groups below, select the correct answer to be inserted in each blank □ in the program description and Program 1.

Answer group for A

a) 0000000000000101   b) 0000000000101000

c) 0001010000000000   d) 1010000000000000

Answer group for B

a) "0"B   b) "1"B

c) "1"B << (PatLen - 1)   d) "1"B << PatLen

e) "1111111111111111"B

Answer group for C

a) "1"B   b) "1"B << (PatLen - 1)

c) "1"B << (i - 1)   d) "1"B << PatLen

e) "1"B << i

Subquestion 2

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

[Description of the Function BitapMatch]

(1) The function receives Text[] and Pat[], and from the smallest element number of Text[], it searches for a string that matches Pat[]. If a match is found, it returns the element number of the Text[] element corresponding to the first character in the matching string. If no match is found, it returns -1.

(2) In the example shown in Figure 1, the string between Text[7] and Text[12] matches Pat[], so it returns 7.

(3) Table 2 shows the specifications of the arguments and the return value of the function BitapMatch.

Table 2 Specifications of the arguments and return value of the function BitapMatch

Arguments / return value	Data type	Input / Output	Description
Text[]	CHAR	Input	One-dimensional array that stores the target string
Pat[]	CHAR	Input	One-dimensional array that stores the search string
Return value	INT	Output	If the search string is found in the target string, return the element number of the Text[] element that corresponds to the first character in the matching string. If the search string is not found, return -1.

[Program 2]

```
FUNCTION: BitapMatch(CHAR: Text[], CHAR: Pat[]) {
   16-bit Binary: Goal, Status, Mask[26]
   INT: i, TextLen, PatLen

   TextLen ← number of characters in Text[];
   PatLen ← GenerateBitMask(Pat[], Mask[]);
   Status ← "0"B;
   Goal ← "1"B << (PatLen - 1);
   FOR (i ← 1; i ≤ TextLen; i ← i + 1) {
      Status ← (Status << 1) | "1"B;                /*** α ***/
      Status ← Status & Mask[Index(Text[i])];       /*** β ***/
      IF ((Status & Goal) ≠ "0"B) {
         return (i - PatLen + 1)
      }
   }
   return (-1)
}
```

The function BitapMatch is executed. The values shown in Figure 1 are stored in Text[] and Pat[].

Table 3 shows the transition in the values of i, Mask[Index(Text[i])], and Status immediately after the execution of the line commented /*** β ***/ (hereinafter, line β) in Program 2.

For example, the value of Status immediately after the execution of line β when i is 1 is "1"B, so the value of Status immediately after the execution of line α when i is 2 is "11"B, which is a bitwise logical sum of "10"B (the value obtained by a logical left shift of "1"B by 1 bit) and "1"B. Next, given the fact that the value of Mask[Index(Text[2])] is "10101"B, the value of Status immediately after the execution of line β when i is 2 is D.

In the same way, given the fact that the value of Status immediately after the execution of line β when i is 8 is "10"B, the value of Mask[Index(Text[9])] immediately after the execution of line α when i is 9 is E, so the value of Status immediately after the execution of line β is F.

Table 3 Transition in the values of i, Mask[Index(Text[i])], and Status immediately after the execution of line β in Program 2 for the example of storage in Figure 1

i	1	2	…	8	9	…
Mask[Index(Text[i])]	"10101"B	"10101"B	…	"10"B	E	…
Status	"1"B	D	…	"10"B	F	…

Answer group for D through F

a) "0"B	b) "1"B	c) "10"B	d) "11"B
e) "100"B	f) "101"B	g) "10101"B	

Subquestion 3

From the answer groups below, select the correct answer to be inserted in each blank □ in the following description concerning the extension of the function GenerateBitMask. Here, assume that B in Program 3 contains the correct answer for B in Subquestion 1.

In order to enable the specification of regular expression, such as the expression shown in Table 4 in the search string, the function `GenerateBitMask` is extended to create the function `GenerateBitMaskRegex`.

Table 4 Regular expression

Symbol	Description
`[]`	Represents a character that matches one of the characters noted in the `[]`. For example, "`A[XYZ]B`" represents "`AXB`", "`AYB`", and "`AZB`".

[Program 3]

```
  FUNCTION: GenerateBitMaskRegex(CHAR: Pat[], 16-bit Binary: Mask[]) {
     INT: i, OriginalPatLen, PatLen, Mode

     OriginalPatLen ← number of characters in Pat[];
     PatLen ← 0;
     Mode ← 0;
     FOR (i ← 1; i ≤ 26; i ← i + 1) {
        Mask[i] ←    B    ;      /* Initialize */
     }
     FOR (i ← 1; i ≤ OriginalPatLen; i ← i + 1) {
        IF (Pat[i] = "[") {
           Mode ← 1;
           PatLen ← PatLen + 1;
        }
        ELSE {
           IF (Pat[i] = "]") {
              Mode ← 0;
           }
           ELSE {
              IF (Mode = 0) {
                 PatLen ← PatLen + 1;
              }
              Mask[Index(Pat[i])] ← Mask[Index(Pat[i])] |
                                    ("1"B << (PatLen - 1));
           }
        }
     }
     return (PatLen);
  }
```

Assume that “`AC[BA]A[ABC]A`” is stored in `Pat[]`, and the function `GenerateBitMaskRegex` is called. In this situation, the bit mask `Mask[1]` that corresponds to “`A`” is `"111101"B`, and the return value of the function `GenerateBitMaskRegex` is $\boxed{\text{G}}$. Furthermore, nesting `[]` in the string that is stored in `Pat[]` is not possible, but if “`AC[B[AB]AC]A`” is mistakenly stored in `Pat[]` and the function `GenerateBitMaskRegex` is called, `Mask[1]` becomes $\boxed{\text{H}}$.

Answer group for G

a) 4 b) 6 c) 9 d) 13

Answer group for H

a) `"1001101"B` b) `"1010100001"B`

c) `"1011001"B` d) `"101111"B`

e) `"110011"B` f) `"111101"B`

Concerning questions **Q7** and **Q8**, **select one** of the two questions.
Then, mark the Ⓢ in the selection area on the answer sheet, and answer the question.
If two questions are selected, only the first question will be graded.

Q7. Read the following description of a C program, and then answer Subquestions 1 through 3.

Combat sports usually have classification to avoid mismatches and for obvious safety reasons. For example, one of the men freestyle wrestling rules has six weight classes: 57, 65, 74, 86, 97, and 125 kg. A wrestler can participate in a weight class greater than or equal to his weight according to the rule. In this question, it is assumed that a wrestler participates in the lightest class of all the possible classes. For example, a wrestler weighing 58 kg is allowed to participate in all classes except for the 57 kg class, but he shall participate in the 65 kg class.

[Program Description]

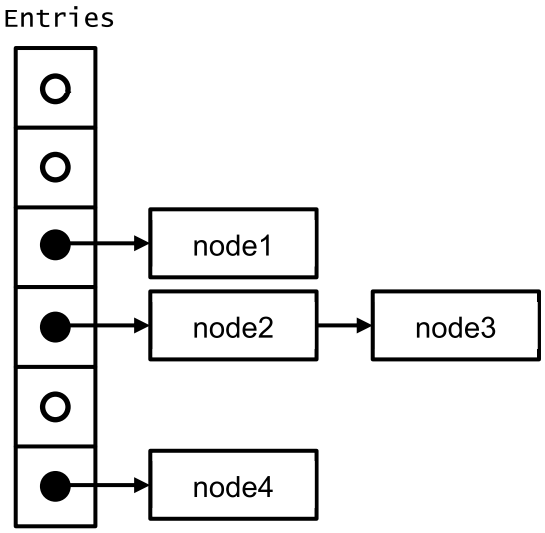
This is a program that accepts the request from wrestlers for participating in a wrestling game, classifies the wrestlers according to their weight, and displays all the requests accepted.

(1) The program receives the number of wrestlers and then the name and weight of the wrestlers from `stdin`. The number of wrestlers and the weight consist of decimal numbers of up to 3 digits, and the name consists of alphabets of up to 50 characters. It is assumed that all input data satisfy these specifications. Table 1 shows an example of information about four wrestlers.

Table 1 An example of information about four wrestlers

Name	Weight (kg)
Anthony	85
Bob	68
Charles	77
Daniel	103

(2) The data structure for storing information about wrestlers is the array of linked lists named `Entries`. Here, `Entries[0]`, `Entries[1]`, …, `Entries[5]` correspond to the weight classes of 57, 65, …, 125 kg, respectively. Each element of `Entries` points to a linked list that holds information about all wrestlers participating in the corresponding weight class, or `NULL` if no wrestlers are in the class.
Figure 1 illustrates the data structures for storing information about wrestlers.



○ indicates NULL pointer.

Figure 1 The data structures for storing information about wrestlers

(3) A node of the linked list represents a wrestler's information, which is a `struct` named `Wrestler` consisting of three fields: `name`, `weight`, and `next`.

```
struct Wrestler {
    char name[51];
    int weight;
    struct Wrestler *next;
};
typedef struct Wrestler Wrestler_t;
```

The first two fields hold a wrestler's name and his weight. The field `next` is the pointer to a node holding the next wrestler's information, or `NULL` if no more wrestler is present in the class.

(4) The program proceeds as follows:

Step 1 Initialize `Entries` and take the number of wrestlers from `stdin`.

Step 2 Repeat (i) through (iii) for each wrestler.

(i) Take the wrestler's name and weight from `stdin`.

(ii) Create a node holding the wrestler's information.

(iii) Find the appropriate linked list for the node created in (ii) from `Entries` and insert the node at the end of the list. If no weight class is found, take another weight for the node and find the appropriate linked list again.

Step 3 Output information in the linked list. If a weight class does not contain any nodes, output "`<empty>`" instead of wrestler's information.

(5) When the program is executed using the sample data in Table 1, the output is as follows:

```
How many wrestlers want to participate: 4
Enter the name of the wrestler: Anthony
Enter the weight for Anthony: 85
Enter the name of the wrestler: Bob
Enter the weight for Bob: 68
Enter the name of the wrestler: Charles
Enter the weight for Charles: 77
Enter the name of the wrestler: Daniel
Enter the weight for Daniel: 103
Result:
 57 kg class: <empty>
 65 kg class: <empty>
 74 kg class: [Bob 68]
 86 kg class: [Anthony 85] [Charles 77]
 97 kg class: <empty>
125 kg class: [Daniel 103]
```

(6) Table 2 shows the functions used in the program.

Table 2 Functions used in the program

Function Name	Description
`void Initialize(void)`	Initializes `Entries`.
`Wrestler_t *AcceptEntry(void)`	Accepts a participation entry from a wrestler.
`int  FindWeightClass(int)`	Finds appropriate weight class for a wrestler of a given weight. Returns -1 if not found.
`void Insert(Wrestler_t *, int)`	Inserts given node in the linked list specified by the given index.
`void Display(void)`	Displays all information of wrestlers registered in the linked list.
`void Deallocate(void)`	Cleans up dynamically allocated memories.
`void *malloc(size_t)`	(Standard library function) Dynamically allocates a given size of memory and returns a pointer that points to that memory.
`Void free(void *)`	(Standard library function) Deallocates memory allocated by `malloc`.

(7) Table 3 shows the variables used in the program.

Table 3 Variables used in the program

Variable Name	Description
`Wrestler_t *Entries[NUM_CLASSES]`	An array of linked lists where each element represents weight classes of 57, 65, 74, 86, 97, and 125 kg respectively.
`const int UpperLimits[NUM_CLASSES]`	An array that stores the upper limit for each weight class.

[Program]

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NUM_CLASSES 6

struct Wrestler {
    char name[51];
    int weight;
    struct Wrestler *next;
};
typedef struct Wrestler Wrestler_t;

Wrestler_t *Entries[NUM_CLASSES];
const int UpperLimits[NUM_CLASSES] = {57, 65, 74, 86, 97, 125};

void Initialize(void);
Wrestler_t *AcceptEntry(void);
int FindWeightClass(int);
void Insert(Wrestler_t *, int);
void Display(void);
void Deallocate(void);

int main() {
    Wrestler_t *wrestler;
    int n, i, index;

    Initialize();
    printf("How many wrestlers want to participate: ");
    scanf("%3d", &n);
    for (i = 0; i < n; i++) {
        wrestler = AcceptEntry();
        index = FindWeightClass(wrestler->weight);
        while (    A    ) {
            printf("%s's weight exceeds %d kg. Re-enter the weight: ",
                   wrestler->name, UpperLimits[NUM_CLASSES - 1]);
            scanf("%3d", &wrestler->weight);
            index = FindWeightClass(wrestler->weight);
        }
        Insert(wrestler, index);
    }
    Display();
    Deallocate();
    return 0;
}

void Initialize(void) {
    int i;
    for (i = 0; i < NUM_CLASSES; i++) {
        Entries[i] = NULL;
    }
}

Wrestler_t *AcceptEntry(void) {
    Wrestler_t *wrestler = (Wrestler_t *)malloc(    B    );
    printf("Enter the name of the wrestler: ");
    scanf("%50s", wrestler->name);
    printf("Enter the weight for %s: ", wrestler->name);
    scanf("%3d", &wrestler->weight);
    wrestler->next = NULL;
    return wrestler;
}

int FindWeightClass(int weight) {
    int i;
    for (i = 0; i < NUM_CLASSES; i++) {
        if (    C    ) {
            return i;
        }
    }
    return -1;
}
```

```
void Insert(Wrestler_t *wrestler, int i) {
    Wrestler_t *curr = Entries[i];
    if (curr == NULL) {                              /*** α ***/
        [    D    ] = wrestler;
    } else {
        while (curr->next != NULL) {                 /*** β ***/
            curr = curr->next;
        }
        [    E    ] = wrestler;                      /*** γ ***/
    }
}

void Display(void) {
    Wrestler_t *curr;
    int i;
    printf("Result:\n");
    for (i = 0; i < NUM_CLASSES; i++) {
        curr = Entries[i];
        printf("%3d kg class:", UpperLimits[i]);
        if (curr == NULL) {
            printf("<empty>\n");
            continue;
        }
        while (curr != NULL) {
            printf("[%s %d] ", curr->name, curr->weight);
            curr = curr->next;
        }
        printf("\n");
    }
}

void Deallocate(void) {
    int i;
    Wrestler_t [    F    ];
    for (i = 0; i < NUM_CLASSES; i++) {
        curr = Entries[i];
        while (curr != NULL) {
            next = curr->next;
            free(curr);
            curr = next;
        }
    }
}
```

Subquestion 1

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

Answer group for A

a) `index != -1` b) `index != 0`
c) `index == -1` d) `index == 0`

Answer group for B

a) `sizeof(Wrestler_t *)` b) `sizeof(Wrestler_t)`
c) `Wrestler_t *` d) `Wrestler_t`

Answer group for C

a) `UpperLimits[i] < weight` b) `UpperLimits[i] <= weight`
c) `UpperLimits[i] > weight` d) `UpperLimits[i] >= weight`

Answer group for D and E

a) `Entries[i + 1]` b) `Entries[i - 1]`
c) `Entries[i]` d) `curr`
e) `curr->next` f) `curr->next->next`

Answer group for F

a) `&curr, &next` b) `&curr, next`
c) `*curr, *next` d) `*curr, next`
e) `curr, next`

Subquestion 2

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

The program is modified as follows to order the nodes in each linked list in ascending order of weight instead of the order of registration.

(1) Replace the line designated by `/*** α ***/` with the following two lines.

```
if (curr == NULL || curr->weight >= wrestler->weight) {
    wrestler->next = curr;        /*** δ ***/
```

(2) Replace the line designated by `/*** β ***/` with the following two lines.

```
while ((curr->next != NULL) &&
       (     G     )) {
```

(3) Insert the following line immediately before the line designated by `/*** γ ***/`.

```
wrestler->next = curr->next;
```

Answer group for G

a) `curr->next->weight < wrestler->weight`

b) `curr->next->weight >= wrestler->weight`

c) `curr->weight < wrestler->weight`

d) `curr->weight >= wrestler->weight`

Subquestion 3

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

It is assumed that the program after modification in Subquestion 2 is executed, and it receives information about five wrestlers shown in Table 4. The line designated by `/*** δ ***/` will be executed H times.

Table 4 Information about five wrestlers

Name	Weight (kg)
Eric	88
Faraday	72
Gregory	91
Harold	97
Issacs	68

Answer group for H

a) 2 b) 3 c) 4 d) 5

Concerning questions **Q7** and **Q8**, **select one** of the two questions.

Q8. Read the following description of Java programs, and then answer Subquestions 1 and 2. See the end of this booklet for the explanation of the APIs used in the Java programs.

[Program Description]

The purpose of this program is to manage "what you need to do" (hereinafter, a ToDo).

(1) The class `ToDo` represents a ToDo. The subject, the deadline, and the priority are specified by the constructor. The deadline is a character string that consists of either 8 numeric characters that represent year, month, and date, or 12 numeric characters that represent year, month, date, hours, and minutes (hereinafter, date and time). For example, October 22, 2023 is represented by the string "`20231022`", and October 22, 2023, 1:00 p.m. is represented by the string "`202310221300`". Here, it is assumed that there is no error in date and time.

This class has methods to get the subject, the deadline, and the priority, methods to set and get the state, and the field `id` that identifies the ToDo.

The enumeration `Priority` represents the priority of the ToDo. The values are `LOW`, `MIDDLE`, and `HIGH` in ascending order of priority.

The enumeration `State` represents the state of the ToDo. The values are `NOT_YET_STARTED`, `STARTED`, and `DONE`.

(2) The class `ToDoList` holds a list of ToDos.

This class guarantees that the list does not contain ToDo's whose field `id` values are the same.

This class has the method `add` to add a ToDo, the method `update` to update a ToDo, and the method `select` to return a list of ToDo's that matches conditions.

When a ToDo that has been held in the list is specified as the argument of the method `add`, and when a ToDo that is not held in the list is specified as the argument of the method `update`, these methods do nothing.

Zero or more conditions can be specified as the argument of the method `select`. When one or more conditions are specified, the method returns the list of ToDo's that satisfy all the conditions. When no condition is specified, the method returns the list of all ToDo's that has been held.

(3) The interface `Condition` is a functional interface that represents the conditions to select ToDo's. The method `test` returns `true` when the condition is satisfied.

(4) The class `ToDoListTester` is a class that is used for testing.

[Program 1]

```
import java.util.UUID;

public class ToDo {
   public enum Priority { LOW, MIDDLE, HIGH }
   public enum State { NOT_YET_STARTED, STARTED, DONE }

   // Regular expression that matches a string of 8 or 12 numeric characters
   private static final String DEADLINE_PATTERN = "\\d{8}|\\d{12}";

   private final String id;
   private String subject;
   private String deadline;
   private Priority priority;
   private State state;

   private ToDo(String subject, String deadline, Priority priority,
                String id, State state) {
      if (!deadline.matches(DEADLINE_PATTERN)) {
         throw new IllegalArgumentException();
      }
      this.subject = subject;
      this.deadline = deadline;
      this.priority = priority;
      this.id = id;
      this.state = state;
   }

   public ToDo(String subject, String deadline, Priority priority) {
      this(subject, deadline, priority,
           UUID.randomUUID().toString(), State.NOT_YET_STARTED);
   }

   public ToDo(ToDo todo) {
      this(todo.subject, todo.deadline,
           todo.priority, todo.id, todo.state);
   }

   public String getSubject() { return subject; }
   public String getDeadline() { return deadline; }
   public Priority getPriority() { return priority; }
   public State getState() { return state; }
   public void setState(State state) { this.state = state; }
   public int hashCode() { return id.hashCode(); }
   public boolean equals(Object o) {
      return o instanceof ToDo && [   A   ];
   }

   public String toString() {
      return String.format(
                "Subject: %s, Deadline: %s, Priority: %s, State: %s",
                subject, deadline, priority, state);
   }
}
```

[Program 2]

```
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class ToDoList {
   private List<ToDo> todoList = new ArrayList<>();

   public void add(ToDo todo) {
      if ([   B   ]) {
         todoList.add(new ToDo(todo));
      }
   }

   public void update(ToDo todo) {
      int index = todoList.indexOf(todo);
      if (index [   C   ]) {
         todoList.set(index, todo);
      }
   }

   public List<ToDo> select(Condition... conditions) {
      return todoList.stream().
         filter(t -> Stream.of(conditions).[   D   ]).
         // Generate a list from the stream
         collect(Collectors.toList());
   }
}
```

[Program 3]

```
import java.util.function.Predicate;

public interface Condition [    E    ] Predicate<ToDo> {}
```

[Program 4]

```
public class ToDoListTester {
   public static void main(String[] args) {
      ToDoList list = new ToDoList();
      list.add(new ToDo("Send the e-mail",
                        "202310231500", ToDo.Priority.HIGH));
      list.add(new ToDo("Reserve a hotel room",
                        "20231025", ToDo.Priority.LOW));
      list.add(new ToDo("Purchase tickets",
                        "20231030", ToDo.Priority.MIDDLE));
      list.add(new ToDo("Create the report",
                        "20231028", ToDo.Priority.HIGH));
      list.add(new ToDo("Set up the meeting",
                        "202311201400", ToDo.Priority.HIGH));
      list.update(new ToDo("Purchase a PC",
                           "20231121", ToDo.Priority.HIGH));
      list.select().forEach(todo -> {
            todo.setState(ToDo.State.STARTED);
            list.update(todo);
         });
      Condition condition1 =
         todo -> todo.getDeadline().compareTo("20231101") < 0;
      Condition condition2 =
         todo -> todo.getPriority().equals(ToDo.Priority.HIGH);
      list.select(condition1, condition2).
         forEach(System.out::println);
   }
}
```

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) `((ToDo) o).id.equals(id)`
b) `(ToDo) o.id.equals(id)`
c) `id.equals(id)`
d) `o.id.equals(id)`

Answer group for B

a) `!todoList.contains(todo)`
b) `!todoList.isEmpty()`
c) `todoList.contains(todo)`
d) `todoList.isEmpty()`

Answer group for C

a) `!= -1`
b) `< todoList.size()`
c) `== -1`
d) `>= todoList.size()`

Answer group for D

a) `allMatch(c -> c.test(t))`
b) `allMatch(c -> t.test(c))`
c) `anyMatch(c -> c.test(t))`
d) `anyMatch(c -> t.test(c))`

Answer group for E

a) `extends`
b) `implements`
c) `super`
d) `throws`

Subquestion 2

Figure 1 shows the execution result of Program 4. From the answer group below, select the correct answer to be inserted in each blank □ in Figure 1.

```
[    F    ], State: STARTED
[    G    ], State: STARTED
```

Figure 1 Execution result of Program 4

Answer group for F and G

a) `Subject: Create the report, Deadline: 20231028, Priority: HIGH`
b) `Subject: Purchase a PC, Deadline: 20231121, Priority: HIGH`
c) `Subject: Purchase tickets, Deadline: 20231030, Priority: MIDDLE`
d) `Subject: Reserve a hotel room, Deadline: 20231025, Priority: LOW`
e) `Subject: Send the e-mail, Deadline: 202310231500, Priority: HIGH`
f) `Subject: Set up the meeting, Deadline: 202311201400, Priority: HIGH`

■ Explanation of APIs used in the Java programs

`java.lang`

`public final class String`

The `String` class represents character strings.

Methods

`public boolean matches(String regex)`

Determines whether or not this string matches the specified regular expression.

For example, the regular expression "`\\d{8}|\\d{12}`" matches a string of 8 or 12 numeric characters.

Parameter: `regex` - The regular expression

Return value: `true` if this string matches the specified regular expression
Otherwise, `false`

`public int compareTo(String str)`

Compares this string and the specified string lexicographically.

Parameter: `str` - The string

Return value: 0 if this string is equal to the specified string
A value less than 0 if this string is lexicographically less than the specified string
A value greater than 0 if this string is lexicographically greater than the specified string

`java.util`

`public final class UUID`

The `UUID` class represents a universally unique identifier (UUID), which is a 128-bit value.

Method

`public static UUID randomUUID()`

Randomly generates a universally unique identifier.

Return value: A randomly generated unique identifier `java.util.function`

`public interface Predicate<T>`

A functional interface that represents a predicate (Boolean-valued function) of one argument.

Method

`public boolean test(T t)`

Evaluates this predicate on the given argument.

Parameter: `t` - The input argument

Return value: `true` if the input argument matches the predicate
Otherwise, `false`

`java.util.stream`

`public interface Stream<T>`

An interface that supports functional-style operations on streams of elements.

Methods

`public boolean allMatch(Predicate<? super T> predicate)`

Returns whether all elements of this stream match the provided predicate.

Parameter: `predicate` - The function

Return value: `true` if all elements of the stream match the provided predicate
Otherwise, `false`

`public boolean anyMatch(Predicate<? super T> predicate)`

Returns whether any elements of this stream match the provided predicate.

Parameter: `predicate` - The function

Return value: `true` if any elements of the stream match the provided predicate
Otherwise, `false`

`public Stream filter(Predicate<? super T> predicate)`

Returns a stream consisting of the elements of this stream that match the given predicate.

Parameter: `predicate` - The function

Return value: A stream consisting of the elements that match the given predicate.

Company names and product names appearing in the test questions are trademarks or registered trademarks of their respective companies. Note that the ® and ™ symbols are not used within the text.