April 2012

Fundamental IT Engineer Examination (Afternoon)

Questions must be answered in accordance with the following:

Question Nos.	**Q1 – Q6**	**Q7 , Q8**
Question Selection	**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 **Q7** and **Q8**, mark the Ⓢ of the question you select to answer in the "Selection Column" on your answer sheet.

   (4) **Answers**
   Mark your answers as shown in the following sample question.

   [Sample Question]
   In which month is the spring Fundamental IT Engineer Examination conducted?

   Answer group
   a) March b) April c) May d) June

   Since the correct answer is "b) April", mark your answer sheet as follows:

   [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 for pseudo-language

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

[Declaration, comment, and process]

Notation		Description
○		Declares names, types, etc. of procedures, variables, etc.
`/* text */`		Describes comments in text.
Process	• variable ← expression	Assigns the value of an expression to a variable.
	• procedure(argument, ...)	Calls the procedure and passes/receives argument.
	▲ conditional expression ↕ process ▼	Indicates a one-way selection process. If the conditional expression is `true`, then the process is executed.
	▲ conditional expression ↕ process 1 process 2 ▼	Indicates a two-way selection process. If the conditional expression is `true`, then process 1 is executed. If it is `false`, then process 2 is executed.
	■ conditional expression ↕ process ■	Indicates a pre-test iteration process. While the conditional expression is `true`, the process is executed repeatedly.
	■ ↕ process ■ conditional expression	Indicates a post-test iteration process. The process is executed, and then while the conditional expression is `true`, the process is executed repeatedly.
	■ variable: init, cond, incr ↕ process ■	Indicates an iteration process. The initial value init (given by an expression) is stored in the variable at the start of the processing, and then while the conditional expression cond is `true`, the process is executed repeatedly. An increment incr (given by an expression) is added to the variable in each iteration.

[Logical constants]

`true`, `false`

(continued on next page)

[Operators and their priorities]

Type of operation	Operator	Priority
Unary operation	+, −, not	High
Multiplication, division	×, ÷, %	↕
Addition, subtraction	+, −	
Relational operation	>, <, ≥, ≤, =, ≠	
Logical product	and	
Logical sum	or	Low

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

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.

Questions **Q1** through **Q6** are all **compulsory**. Answer every question.

Q1. Read the following description concerning a simple microprocessor, and then answer Subquestions 1 through 3.

[Block Schema]

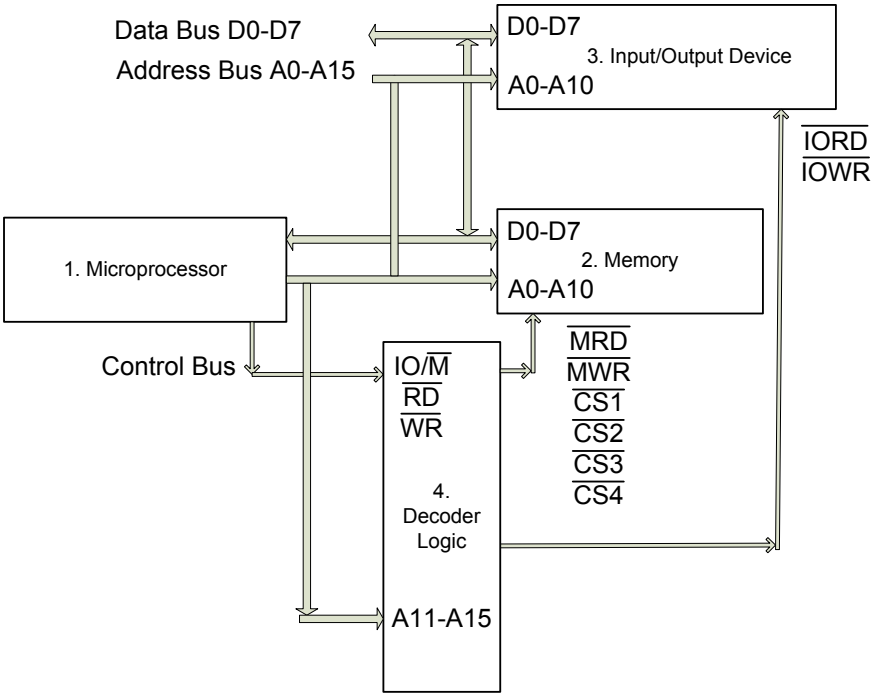


Figure 1. Block schema

The microprocessor system shown in Figure 1 includes four principal blocks:

1. The microprocessor itself.
2. Memory block that consists of 4 integrated circuits (hereinafter, chips) - one ROM chip and 3 RAM chips (Refer to Figure 2 for more information).
   The signal $\overline{CS1}$ is used for selecting ROM chip, and signals $\overline{CS2}$ to $\overline{CS4}$ are used for selecting RAM chips (See Table 2 for more information), and $\overline{CS1}$ to $\overline{CS4}$ will be decoded from higher address lines A11 to A15.
3. IO device.
4. Decoder logic block used for decoding memory read $\overline{MRD}$, memory write $\overline{MWR}$, IO read $\overline{IORD}$, and IO write $\overline{IOWR}$ from the three microprocessor's signals IO/$\overline{M}$, $\overline{RD}$, and $\overline{WR}$ (Refer to Table 1 for more information).

Here, a symbol with a bar indicates the signal on this line is 0 (called active low), and a symbol without a bar indicates the signal on this line is 1 (called active high). For example, A15, $\overline{A14}$, $\overline{A13}$, $\overline{A12}$ indicates that the signals on A15, A14, A13, A12 are 1, 0, 0, 0.

[Signal Description]

Table 1. Microprocessor Signals

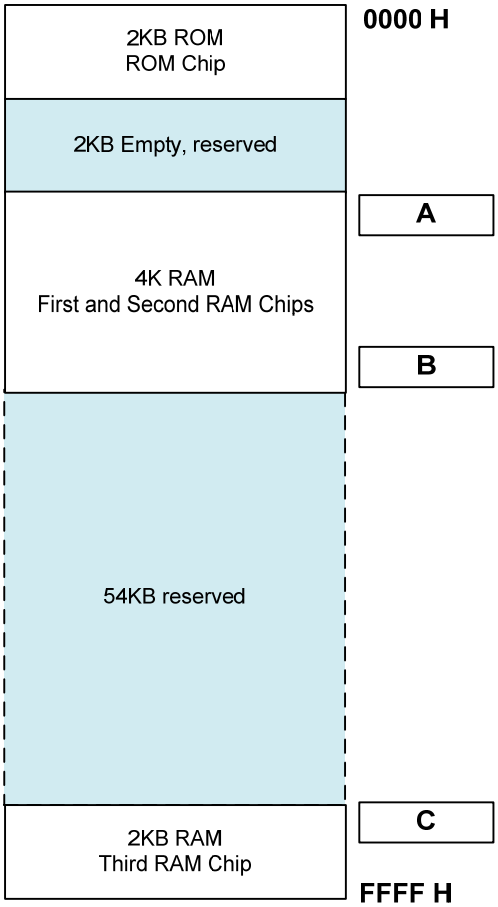
Signal Name	Symbol	Description
Address bus	A0 to A15	16 bit one-directional (output) address lines
Data bus	D0 to D7	8 bit bi-directional data lines
IO/memory Select	IO/$\overline{M}$	Control output line. This signal has two functions. 1) Select IO: when microprocessor selects IO device, the signal on this IO/$\overline{M}$ line is 1. 2) Select memory: when microprocessor selects memory, the signal on this IO/$\overline{M}$ line is 0.
Read	$\overline{RD}$	Control output line. When microprocessor reads IO device or memory (ROM or RAM), the signal on this $\overline{RD}$ line is 0 in IO/memory reading cycle.
Write	$\overline{WR}$	Control output line. When microprocessor writes IO device or memory (RAM), the signal on this $\overline{WR}$ line is 0 in IO/memory writing cycle.

[Pin Description]

Table 2. Pin description

Signal Name	Symbol	Description
Address bus	A0 to A10	11 bit address lines.
Data bus	D0 to D7	8 bit data lines.
Chip Select	$\overline{CS}$	The signal on the selected memory chip is 0 in memory reading or writing cycle. There are $\overline{CS1}$ to $\overline{CS4}$: $\overline{CS1}$ selects ROM chip, $\overline{CS2}$ selects first RAM chip, $\overline{CS3}$ selects second RAM chip, and $\overline{CS4}$ selects third RAM chip.
Read	$\overline{RD}$	The signal on this $\overline{RD}$ line is 0 in memory (ROM or RAM) reading cycle.
Write	$\overline{WR}$	The signal on this $\overline{WR}$ line is 0 in memory (RAM) writing cycle.

[Memory Map]



Note: "*nnnn* H" signifies hexadecimal value *nnnn*.
First RAM chip covers lower addresses of 4k RAM address range and
Second RAM chip covers higher addresses of 4k RAM address range.

Figure 2. Memory map

Subquestion 1

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

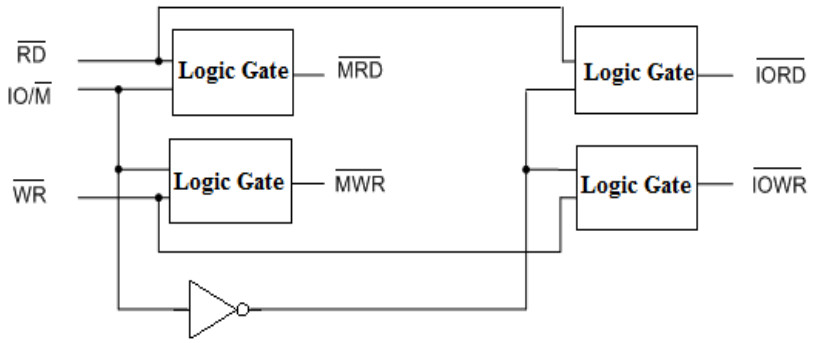
Answer group

a) 04FF H	b) 07FF H	c) 0800 H	d) 1000 H
e) 1800 H	f) 1FFF H	g) F800 H	h) FC00 H

Subquestion 2

Figure 3 shows how decoder logic works. In Figure 3, four identical logic gates are used for decoding four signals: memory read, memory write, IO read, and IO write.

From the answer group below, select the correct type of logic gates.



Note: Graphic symbol signifies NOT gate.

Figure 3. Decoding schema

Answer group

a) AND
b) NAND
c) NOR
d) OR

Subquestion 3

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

The signals $\overline{CS1}$ to $\overline{CS4}$ are decoded from address lines A11 to A15. The logic function for $\overline{CS1}$ is $\overline{A15}$, $\overline{A14}$, $\overline{A13}$, $\overline{A12}$, $\overline{A11}$, for $\overline{CS2}$ is [D], for $\overline{CS3}$ is [E], and for $\overline{CS4}$ is A15, A14, A13, A12, A11.

Answer group

a) $\overline{A15}$, $\overline{A14}$, $\overline{A13}$, $\overline{A12}$, A11
b) $\overline{A15}$, $\overline{A14}$, $\overline{A13}$, A12, $\overline{A11}$
c) $\overline{A15}$, $\overline{A14}$, $\overline{A13}$, A12, A11
d) $\overline{A15}$, $\overline{A14}$, A13, $\overline{A12}$, $\overline{A11}$
e) $\overline{A15}$, $\overline{A14}$, A13, A12, $\overline{A11}$
f) $\overline{A15}$, A14, $\overline{A13}$, $\overline{A12}$, $\overline{A11}$
g) $\overline{A15}$, A14, A13, A12, $\overline{A11}$

Q2. Read the following description concerning the processing by a compiler, and then answer Subquestions 1 through 4.

A compiler is software intended to translate a source program written in a programming language into an object program. In the processing of the compiler, the syntax analysis generates a syntax tree while reading the characters or terms produced by the lexical analysis, and analyzes whether the sequence of the characters or terms follows the grammar.

Subquestion 1

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

The syntax of an equation consisting of binary operators is represented by a binary tree. To generate an object program from the binary tree, a depth-first search is done on the binary tree and the operator of each node is evaluated in a return path. The figure shows an example of a syntax tree and the search order for an equation.

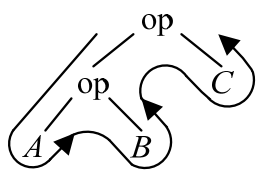


Fig. Example of a syntax tree and the search order

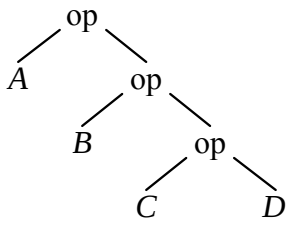
As a result of the search, the example of the syntax tree in the figure is interpreted to perform the operation op for *A* and *B* and then perform the operation op for the result and *C*.

When the equation includes parentheses, precedence is given to the operation within the parentheses, and the rest of the equation is evaluated from left to right. In this case, the syntax tree for the following equation is [ ] .

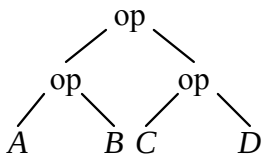
Equation: *A* op (*B* op *C*) op *D*

Answer group

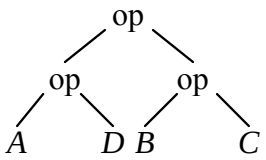
a)



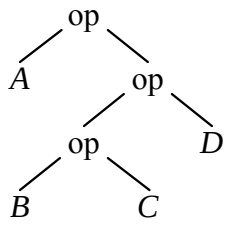
b)



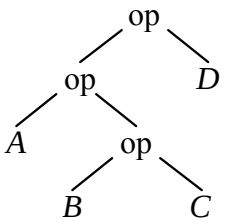
c)



d)



e)



Subquestion 2

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

The grammar of a programming language is defined by syntax rules. The syntax rules of an equation define not only the syntax of the equation but also the operator precedence and associativity. As an example, consideration is given to the syntax rules of an equation that uses operators op1 and op2 having different precedence, as well as parentheses.

[Syntax rules of an equation]

(1) Equation → Term | Equation op1 Term

(2) Term → Factor | Term op2 Factor

(3) Factor → Name | (Equation)

(4) Name → *A* | *B* | *C* | *D* | *E*

"→" indicates that the syntax element on the left side is defined on the right side.

"|" means "or".

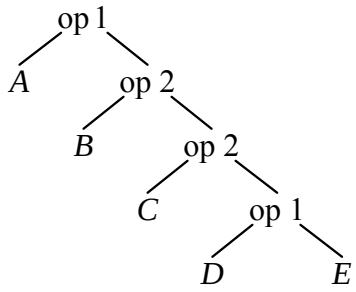
If a depth-first search is done, the execution order of the operators op1 and op2 included in the given equation can be represented with a syntax tree that can be generated based on the [Syntax rules of an equation]. For example, when the [Syntax rules of an equation] are

applied to the equation below, the resulting syntax tree is [ ].

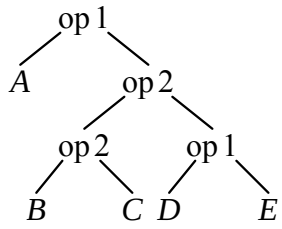
Equation: A op1 B op2 C op2 (D op1 E)

Answer group

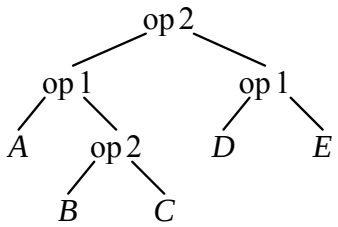
a)



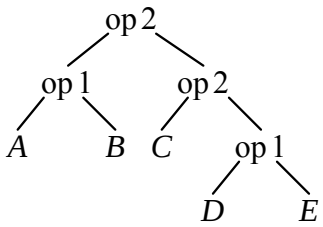
b)



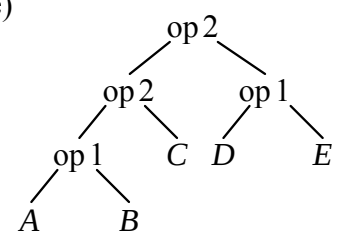
c)



d)



e)



Subquestion 3

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

Other than a syntax tree, the postfix notation (reverse Polish notation), which is equivalent to the binary tree notation, can be used to represent the result of the syntax analysis. In the postfix notation, the two operands used in an operation are written before the operator. And the postfix notation can be considered to represent the operation sequence obtained by searching the syntax tree. For instance, when the operator op in the example shown in the figure of Subquestion 1 is addition (+), the equation is represented as $A\ B + C +$ in the postfix notation. This means to add A and B and then add the result and C.

When the equation $A \times B + C \times D + E$ is represented in the postfix notation, it is [ ]. Here, addition (+) and multiplication (×) correspond to the operators op1 and

op2 of Subquestion 2, respectively, and the operator precedence and associativity comply with the [Syntax rules of an equation].

Answer group

a) $A\ B\ C\ D \times \times + E +$

b) $A\ B\ C \times + D \times E +$

c) $A\ B \times C\ D \times + E +$

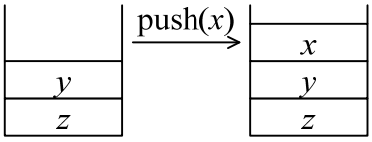
d) $A\ B \times C + D \times E +$

Subquestion 4

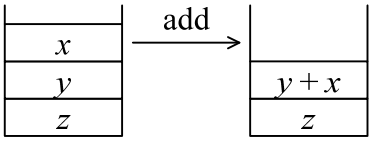
From the answer group below, select the correct answer to be inserted in the blank [ ] in the following description.

An equation represented in the postfix notation can be evaluated from left to right by using a stack. When the equation $A + B + C \times D$ is evaluated after it is converted to the postfix notation, the order of stack operations is [ ]. Here, three functions described below are available for stack operations. Also, addition (+) and multiplication (×) correspond to the operators op1 and op2 of Subquestion 2, respectively, and the operator precedence and associativity comply with the [Syntax rules of an equation].

push(x): Pushes x to the stack.



add: Pops two operands from the stack, adds them, and then pushes the result to the stack.



mul: Pops two operands from the stack, multiplies them, and then pushes the result to the stack.

Answer group

a) push(A) → add → push(B) → add → push(C) → mul → push(D)

b) push(A) → push(B) → add → push(C) → mul → push(D) → add

c) push(A) → push(B) → add → push(C) → push(D) → mul → add

d) push(A) → push(B) → push(C) → push(D) → add → add → mul

Q3. Read the following description concerning a database for managing telephone usage, and then answer Subquestion.

Dream Merchant Company uses PBX Call Monitor System for managing telephone usage in the company.
Call log data from the PBX phone unit is imported into the database, and the system creates various reports from that data. Examples of such reports are:

- Monthly summary of telephone usage by each extension
- Call charges by each extension
- Monthly summary of telephone cost by each department

Call log data is loaded into `Calls` table.
`Calls` table contains the following information.

`Calls` Table:

`Call_Date`	Date of call transaction (yyyymmdd)
`Call_Time`	Time of call transaction (hhmmss)
`Ext_No`	Internal extension number from which the call is made (numeric)
`To_Phone`	External phone number to which the call is made (string)
`Tariff`	Tariff code for external phone (1: Local, 2: Mobile, 3: Foreign)
`Duration`	Duration of the call in seconds (numeric)
`Currency`	Currency code for call charge (1: Local, 2: USD)
`Amount`	Call charge for this call (numeric)

`Ext_All` view and `Ext_Intl` view are created base on `Department` table and `Extension` table. `Ext_All` view contains the following information for all the extension lines, while `Ext_Intl` view contains the same information for only the extension lines that are permitted to make foreign calls.

`Ext_All` View and `Ext_Intl` View:

`Ext_No`	Internal extension number (e.g. 101, 102, …) (numeric)
`Ext_Name`	Name of the extension (e.g. X101, X102, …) (string)
`Dept_ID`	Departmant ID (e.g. 11, 12, …) (numeric)
`Dept_Name`	Name of the department (e.g. Sales, Service, …) (string)

Subquestion 1

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

The following SQL statement is created. It selects only the extension lines that are permitted to make foreign calls and lists the foreign calls made by those extensions in March 2012, with order of longest duration on top. Durations are rounded up to nearest minutes.

```
SELECT Ext_Name, Call_Date, [      A      ], To_Phone, Amount
  FROM [      B      ] ON C.Ext_No = D.Ext_No
    WHERE Tariff = 3 AND Call_Date BETWEEN '20120301' AND '20120331'
    ORDER BY [      A      ] DESC
```

Sample output from the above SQL statement:

Ext Name	Call Date	Duration (min)	To Phone	Amount
X201	20120302	10	65 627272	46.5
X205	20120303	7	43 774848	37.5
...	...	...	...	...

Answer group for A

a) `CEIL(Duration/60)`
b) `FLOOR(Duration/60)`
c) `ROUND((Duration+30)/60,0)`
d) `ROUND((Duration+50)/60,0)`

Answer group for B

a) `Calls C INNER JOIN Ext_Intl E`
b) `Calls C RIGHT OUTER JOIN Ext_Intl E`
c) `Ext_Intl E FULL OUTER JOIN Calls C`
d) `Ext_Intl E INNER JOIN Calls C`

Subquestion 2

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

The following SQL statement is created to list the local and mobile call counts and their call charges in March 2012, by each extension, order by department name. Calls with Amount = 0 (they are incoming calls or toll-free calls) are not included in the list.

```
SELECT Dept_Name, Ext_Name,
       [          C          ], -- Local, Mobile
       [          D          ], -- Total
       [          E          ]  -- Amount
  FROM Calls C, Ext_All E
    WHERE C.Ext_No = D.Ext_No AND Amount <> 0
      AND Call_Date BETWEEN '20120301' AND '20120331'
    ORDER BY Dept_Name, Ext_Name
```

Sample output from the above SQL statement:

Department	Ext Name	Local	Mobile	Total	Amount
Admin	X203	10	3	13	25.0
Service	X302	20	10	30	70.0
Service	X303	30	7	37	65.0
...	...	...	...	...	...

Answer group

a) `COUNT(CASE WHEN Tariff = 1 THEN Tariff ELSE 0 END),`
`COUNT(CASE WHEN Tariff = 2 THEN Tariff ELSE 0 END)`

b) `COUNT(CASE WHEN Tariff < 3 THEN Tariff ELSE 0 END)`

c) `SUM(CASE WHEN Tariff = 1 OR 2 THEN Amount ELSE 0 END)`

d) `SUM(CASE WHEN Tariff = 1 OR 2 THEN Tariff ELSE 0 END)`

e) `SUM(CASE WHEN Tariff = 1 THEN 1 ELSE 0 END),`
`SUM(CASE WHEN Tariff = 2 THEN 1 ELSE 0 END)`

f) `SUM(CASE WHEN Tariff = 1 THEN Tariff ELSE 0 END),`
`SUM(CASE WHEN Tariff = 2 THEN Tariff ELSE 0 END)`

g) `SUM(CASE WHEN Tariff IN (1, 2) THEN 1 ELSE 0 END)`

h) `SUM(CASE WHEN Tariff IN (1, 2) THEN Amount ELSE 0 END)`

i) `SUM(CASE WHEN Tariff IN (1, 2) THEN Tariff ELSE 0 END)`

Q4. Read the following description concerning a company's network, and then answer Subquestion.

Company A's internal network consists of two LANs, LAN1 and LAN2, as shown in the Figure. Two LANs are connected to Router1.
For LAN1, the network address 192.168.10.0 and the subnet mask 255.255.255.0 are set. For LAN2, the network address 172.17.0.0 and the subnet mask 255.255.0.0 are set. Switch1 in LAN1 and Switch2 in LAN2 are connected to Router1. On Router1, the interface connected to LAN1 has the IP address 192.168.10.100, and the interface connected to LAN2 has the IP address 172.17.10.100.
In LAN1, PC1, PC2 and Server1 are connected to Switch1. Currently, DHCP Service, Web Service and DNS Service are running on Server1. PC1 and PC2 receive IP address, subnet mask, etc. from Server1. In LAN 2, PC3 and PC4 are connected to Switch2. Static IP addresses are assigned to PC3 and PC4. All PCs can communicate with each other within each separated LAN.

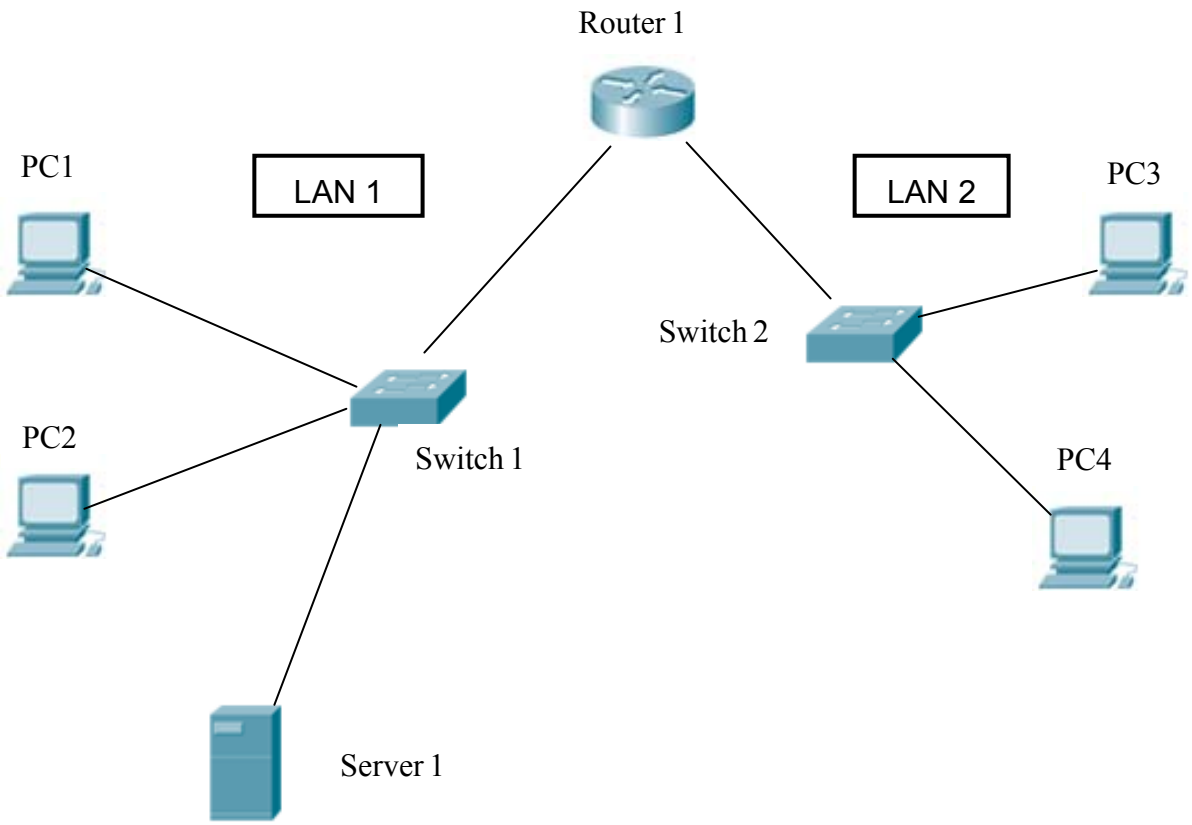


Figure. Company A's internal network

The following table shows the network configuration information.

	IP Address	Subnet mask	Default Gateway	DNS	MAC Address
PC4	172.17.10.2 Static	255.255.0.0	172.17.10.100	192.168.10.1	00-50-56-c0-00-01
PC3	172.17.10.1 Static	255.255.0.0	172.17.10.100	192.168.10.1	00-50-56-c0-00-08
PC2	192.168.10.2 Dynamic	255.255.255.0	192.168.10.1	192.168.10.1	00-0c-29-42-fa-77
PC1	192.168.10.3 Dynamic	255.255.255.0	192.168.10.1	192.168.10.1	00-0c-29-42-fa-9f
Server1	192.168.10.1 Static	255.255.255.0	192.168.10.1	192.168.10.1	00-0c-29-42-fa-8b

Subquestion

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

All PCs can communicate with each other within each separated LAN, but cannot communicate with PCs on the other LAN. To make communication across two LANs possible, [A] must be changed.

When PC2 is detached from LAN1 and attached to LAN2, it is necessary to [B].

The following table is the excerpt from the routing table of Router1 which is set correctly. The table shows that packets to destination address [C] are sent from the interface 192.168.10.100.

Network Destination	Interface
[C]	192.168.10.100

ARP is a protocol for mapping an IP address to a MAC address. To obtain a MAC address, it sends a broadcast ARP packet that contains the IP address which corresponds to that MAC address. Among the nodes that receive the packet, only the node whose IP address is identical with that of ARP packet sends its MAC address back to the sender of the packet. Though, there are cases where MAC addresses cannot be obtained. For example, in Company A's network, PC1 cannot obtain the MAC address of [D].

Answer group for A

a) default gateway of all PCs
b) default gateway of Server1
c) IP address range for PC3 and PC4
d) subnet mask of PC3 and PC4

Answer group for B

a) assign the static IP address, subnet mask, and set default gateway as 172.17.10.100
b) assign the static IP address, subnet mask, and set default gateway as 192.168.10.100
c) change the default gateway of PC2 to 172.17.10.100
d) receive the IP address from Server1

Answer group for C

a) 172.17.0.0
b) 172.17.10.100
c) 192.168.10.0
d) 192.168.10.1

Answer group for D

a) PC3
b) Router1
c) Server1

Q5. Read the following description concerning a program design, and then answer Subquestions 1 and 2.

AZ supermarket decides to create a program for calculating the working time of the employees each day. Since AZ supermarket opens 24 hours, a start working time (hereinafter, start time) and a finish working time (hereinafter, finish time) differ from one employee to another. According to the office regulations, elapsed working time must be less than 24 hours, and employees must take break(s) when they work 5 hours or more. The total break time that must be taken is prescribed in the office regulations and it depends on the elapsed working time.

[Program Description]

The program is divided into a few relatively independent tasks. An overview of each task is shown in the Figure.

Task No.	Program Name	Description
1	PGM1	Login Displays a login screen, and prompts the employee to enter the employee ID and password.
2	PGM2	Input the start time and finish time Prompts the employee to enter today's start time and finish time, in 24-hours "hhmm" format. Then checks the validity of the input data. The start time and finish time are stored into the global integer variables `StartHHMM` and `FinishHHMM` respectively. For example, if the start time is 5:30 PM, `StartHHMM` will have the integer value 1730.
3	PGM3	Calculate the elapsed working time Calculates the elapsed working time from the values of `StartHHMM` and `FinishHHMM`. If `StartHHMM` > `FinishHHMM`, it means that his/her job extended for two days across 0:00 midnight.
4	PGM4	Display the elapsed working time Displays the elapsed working time on the screen. There are 3 display formats; hours and minutes, hours only, and minutes only. Examples: Start time / Finish time / Display contents 09:00 / 17:30 / `8 hr 30 min` 09:00 / 12:00 / `3 hr` 09:00 / 09:50 / `50 min`
5	PGM5	Calculate the break time Calculates the break time. Office regulations prescribe that an employee must have 1-hour break time after every 5-hours continuous work. For example, if the elapsed working time is 4 hours, there is no break time, and if the elapsed working time is 13 hours, 2 hours out of 13 hours are considered as the break time.

6	PGM6	Display the break time Displays the total break time on the screen.
7	PGM7	Calculate the actual working time Calculates the actual working time as: Actual working time = Elapsed working time – Break time
8	PGM8	Save the data into the payroll database Saves the input data and calculated data into the payroll database, after the employee confirms the data.
9	PGM9	Logoff Displays a logoff screen, and prompts the employee to enter logoff.

Figure. Overview of each task

Subquestion 1

From the answer groups below, select the correct answer to be inserted into each blank [ ] in the following program. If needed, select the same answer twice or more.

The program PGM3 for task No.3, as shown below, is created. The program calculates the elapsed working time using the values of StartHHMM and FinishHHMM, and stores the result into the global integer variable ElapsedHHMM in 24-hours "hhmm" format.
As for the operators ÷ and %, refer to the note of [Operators and their priorities] in Notations used for pseudo-language.

[Program PGM3]

```
○ program PGM4
○ integer: StartMin, FinishMin, ElapsedMin

StartHHMM > FinishHHMM
  • FinishHHMM ← [ A ]

• StartMin  ← (StartHHMM  ÷ 100) × 60 + (StartHHMM  % 100)
• FinishMin ← (FinishHHMM ÷ 100) × 60 + (FinishHHMM % 100)
• ElapsedMin ← FinishMin - StartMin
• ElapsedHHMM ← (FinishMin ÷ [ B ]) × [ C ]
                + (FinishMin % [ D ])
```

Answer group for A

a) 1440 - StartHHMM
b) 2400 - StartHHMM
c) FinishHHMM + 1440
d) FinishHHMM + 2400

Answer group for B through D

a) 24
b) 60
c) 100
d) 600
e) 1440
f) 2400

Subquestion 2

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

(1) The program testing is an important activity to verify the correctness of the program. In order to test the program PGM4 for task No.4, the following 9 cases of test data, A through I, are provided to check if the result is displayed correctly. If a tester has a limit of time for testing the program PGM4, the tester should provide, for example, the test data which contains the cases [E] at least.

A: Start time = 0000, Finish time = 0820
B: Start time = 0020, Finish time = 0400
C: Start time = 0900, Finish time = 0900
D: Start time = 0945, Finish time = 1805
E: Start time = 1120, Finish time = 1130
F: Start time = 2215, Finish time = 1615
G: Start time = 2240, Finish time = 1200
H: Start time = 2320, Finish time = 0000
I: Start time = 2340, Finish time = 0040

(2) In the program design phase, a designer should confirm that necessary and sufficient specification information is supplied or not. For example, as for the program PGM5 for task No.5, the program specification information (the description for task No.5 in the Figure) is not sufficient, and the program logic cannot be determined for some working cases, such as [F].

Answer group for E

a) A, C, E, H
b) A, F, G, I
c) B, C, F, G
d) B, D, E, H
e) C, E, F, I
f) C, F, G, H

Answer group for F

a) the start time is 09:00 and finish time is 15:01
b) the start time is 09:00 and finish time is 22:00
c) the start time is 22:00 and finish time is 02:59
d) the start time is 22:00 and finish time is 09:30

Q6. Read the following description of a program |and the program itself, and then answer Subquestions 1 and 2.

[Program Description]

The program `Sieve` is an implementation of Eratosthenes' method named "Sieve of Eratosthenes".

The following procedure finds all prime numbers up to a given integer n ($n > 1$).

1. Provide an array of n elements; indexed from 1 to n.
2. Set 0 to the first element of the array, and set 1 to all other elements of the array.
3. Initially, let p be equal to 2, the first prime number.
4. Replace all multiples of p, which are less than or equal to n in the array, with 0s.
5. Find the first non-zero number that is greater than p remaining in the array (this number is the next prime). Replace the value of p with this number.
6. If $p \times p \leq n$, go to step 3.
7. All prime numbers up to n are identified.

In this implementation, some elements of the array will be set to 0 several times.

The following table shows an example of array contents after the sieve is made on the numbers 1 through 10. In this example, numbers 2, 3, 5 and 7 are identified as prime numbers.

Array contents	0	1	1	0	1	0	1	0	0	0
index	1	2	3	4	5	6	7	8	9	10

The parameters in the following table are passed to the program `Sieve`.

Parameter	Description
`n`	An integer variable that specifies the range of prime number search.
`numbers[ ]`	An integer array sized `n`. After processing, prime numbers up to `n` are obtained in this array.

The sub-program `print(`*expression*`)` is used to output the value of *expression*.

[Program]

```
○ program Sieve(integer: n, integer: numbers[n])
○ integer: i, n, p, s

• numbers[1] ← 0
■ i: 2, i ≤ n, 1
  • numbers[i] ← 1
■
• p ← 2
■ p × p ≤ n
  • s ← p + p
  ■ s ≤ n
    • numbers[s] ← 0
    • s ← [      A      ]
  ■
  • p ← p + 1
  ■ p ≤ n and numbers[p] = 0
    • p ← [      B      ]
  ■
■
■ i: 2, i ≤ n, 1     /* Print prime numbers up to n */
  ▲ numbers[i] = 1
  ▼ • print([      C      ])
■
/* End of the program Sieve */
```

Subquestion 1

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

Answer group

a) `i`	b) `numbers[i]`
c) `p + 1`	d) `p + 2`
e) `p + p`	f) `s + 1`
g) `s + p`	h) `s + s`

Subquestion 2

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

(1) During the execution of the program `Sieve`, some elements of the array will be set to zero several times. For example, when n is equal to 50, the array element `numbers[8]` is set to zero [D] time(s), and `numbers[30]` is set to zero [E] time(s).

(2) The amount of memory required for the implementation of "Sieve of Eratosthenes" is expressed as [F] in O-notation.

(3) Assuming that the program `Sieve` is executed on the numbers 1 through n, and the prime numbers 2, 3, ..., q are found. Thus, q is the maximum prime number that is less than or equal to n. Now, consider another given number X which is greater than n. For determining whether X is prime number or not, a simple algorithm is used. This algorithm checks if X can be divided by any one of the prime numbers 2, 3, …, q. When this algorithm is used, the maximum value of X which can be determined whether it is a prime number or not will be [G].

Answer group for D and E

a) 0 b) 1 c) 2 d) 3 e) 4

Answer group for F

a) $O(1)$ b) $O(n \log n)$
c) $O(n)$ d) $O(n^2)$
e) $O(n^3)$

Answer group for G

a) $2n$ b) n^2 c) nq d) $2q$ e) q^2

Concerning questions **Q7** and **Q8**, **select one** of the two questions.
Then, mark (S) 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 the program itself, and then answer Subquestion.

[Program Description]

This is a program that performs multiplication of two integers *a* and *b*, and obtains its product *prod* using an efficient method known as bit-pair recoding. The bit-pair recoding is a technique used by many computers to speed up the multiplication operation. Basically, multiplication is performed through repeated additions and shift operations. The bit-pair recoding technique recodes multiplier bits by considering contiguous 3 bits at a time. It reduces the maximum number of summands to n/2 for n-bit operands in computing for the product of two integers *a* and *b*. Bits are shifted two places at a time, while examining the last 3 bits of *b* at a time to determine the respective operations to be executed in each shift. Table 1 shows the respective operations by 3-bit pattern.

Table 1. Operations by 3-bit pattern

Bit pattern in *b*			Operation
0	0	0	No operation
0	0	1	1 x *a*
0	1	0	2 x *a* - *a*
0	1	1	2 x *a*
1	0	0	-2 x *a*
1	0	1	-2 x *a* + *a*
1	1	0	-1 x *a*
1	1	1	No operation

For the first check, only the last 2 bits of *b* are examined. Table 2 shows the respective initial operations by 2-bit pattern.

Table 2. Initial operations by 2-bit pattern

Bit pattern in *b*		Operation
0	0	No operation
0	1	1 x *a*
1	0	-2 x *a*
1	1	-1 x *a*

The function `main` performs multiplication operation using the bit-pair recoding technique for all the pairs of integers stored in array `data` and displays the products.
The algorithm used in the program is as follows:

Step 1. Examine the last 2 bits of *b*, perform the appropriate operation in Table 2, and set the obtained value to *prod*.
Step 2. Shift *a* to the left by 2 bits, and shift *b* to the right by 1 bit.
Step 3. Repeat Step 3a and Step 3b until all bits in *b* are examined.
Step 3a. Examine the last 3 bits of *b*, perform the appropriate operation in Table 1, and add the obtained value to *prod*.
Step 3b. Shift *a* to the left by 2 bits, and shift *b* to the right by 2 bits.

Assuming that the values of *a*, *b* and *prod* do not exceed the size of an integer variable.
The following example illustrates how *prod* = 48 is obtained from *a* = 8 and *b* = 6.

Step 1:

a =	8		0	0	0	1	0	0	0
b =	6		0	0	0	0	1	1	0
prod =	0	before							
prod =	-16	after							

Step 2:

a =	32		0	1	0	0	0	0	0
b =	3		0	0	0	0	0	1	1
prod =	-16								

Step 3: Loop

Step 3a:

a =	32		0	1	0	0	0	0	0
b =	3		0	0	0	0	0	1	1
prod =	-16	before							
prod =	48	After							

Step 3b:

a =	128	1	0	0	0	0	0	0	0
b =	0		0	0	0	0	0	0	0
prod =	48								

The program terminates with *prod* = 48, as there are no more bits in *b* to be examined.

[Program]

```
#include <stdio.h>

int product (int a, int b);

int main (void) {
    int data[][2] = {{2, 1}, {-25,5}, {3, 4}, {-2, -3}, {50,50}};
    int i, prod;

    for (i = 0; i < [    A    ]; i++) {
        printf("A = %d B = %d\n", data[i][0], data[i][1]);
        prod = product (data[i][0], data[i][1]);
        printf("Product is %d\n\n", prod);
    }
    return 0;
}

int product (int a, int b) {
    int i, prod;

    prod = 0;
    /* check for special cases */
    switch ([    B    ]) {
        case 0 : break;
        case 1 : prod = [    C    ]; break;
        case 2 : prod = [    D    ]; break;
        case 3 : prod = -a; break;
        default : break;
    }
    a = [    E    ];
    b = b >> 1;
    for (i = 1; i < sizeof(int)*4; i++) {
        /* check bit-pair pattern */
        switch ([    F    ]) {
            case 0 : break;
            case 1 : prod = prod + a; break;
            case 2 : prod = [    G    ]; break;
            case 3 : prod = prod + 2 * a; break;
            case 4 : prod = prod - 2 * a; break;
            case 5 : prod = [    H    ]; break;
            case 6 : prod = prod - a; break;
            case 7 : break;
            default : break;
        }
```

```
        b = b >> 2;
        a = [    E    ];
    }
    return prod;
}
```

Subquestion

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

Answer group for A

a) `sizeof(data) * 2`
b) `sizeof(data) * 4`
c) `sizeof(data) * sizeof(int) / 2`
d) `sizeof(data) * sizeof(int) / 4`
e) `sizeof(data) / sizeof(int) / 2`
f) `sizeof(data) / sizeof(int) / 4`

Answer group for B and F

a) `b & 0011`
b) `b && 0011`
c) `b & 0x3`
d) `b && 0x3`
e) `b & 0111`
f) `b && 0111`
g) `b & 0x7`
h) `b && 0x7`

Answer group for C and D

a) `a`
b) `-a`
c) `a + 2`
d) `-a + 2`
e) `a - 2`
f) `-a - 2`
g) `2 * a`
h) `-2 * a`

Answer group for E

a) `a >> 2`
b) `a * 2`
c) `a >> 4`
d) `a * 4`

Answer group for G and H

a) `prod + a`
b) `prod - a`
c) `prod + a + 2`
d) `prod - a + 2`
e) `prod + a - 2`
f) `prod - a - 2`
g) `prod + 2 * a`
h) `prod - 2 * a`

Q8. Read the following description of a Java program and the program itself, and then answer Subquestion.

[Program Description]

Consider the problem of a trapped mouse that tries to find its way to an exit in a maze. Figure 1 shows an example of a maze, and Figure 2 shows the internal expression of Figure 1.

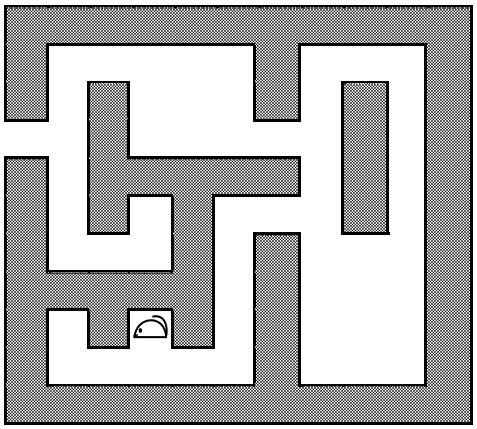


Figure 1. A mouse in a maze

```
1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 1 0 0 0 1
1 0 1 0 0 0 1 0 1 0 1
e 0 1 0 0 0 0 0 1 0 1
1 0 1 1 1 1 1 0 1 0 1
1 0 1 0 1 0 0 0 1 0 1
1 0 0 0 1 0 1 0 0 0 1
1 1 1 1 1 0 1 0 0 0 1
1 0 1 m 1 0 1 0 0 0 1
1 0 0 0 0 0 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1
```

Figure 2. Internal expression

The mouse hopes to escape from the maze by systematically trying all the routes. If it reaches a dead end, it retracts its steps to the last position and begins to try at least one or more untried path. For each position, the mouse can go in one of four directions: right, left, down, up. Regardless of how close it is to the exit, it always tries the open paths in this order, which may lead to some unnecessary detours. By retaining information that allows for resuming the search after a dead end is reached, the mouse uses a method called backtracking.

The maze is implemented as a two-dimensional `char` array in which passages are marked with 0s, walls by 1s, exit position by the letter e, and the initial position of the mouse by the letter m (Figure 2). In this program, the maze problem is slightly generalized by allowing the exit to be in any position of the maze and allowing passages to be on the borderline. To protect itself from falling off the array by trying to continue its path when an open cell is reached on one of the borderlines, the mouse also has to constantly check whether it is in such borderline position or not. To avoid it, the program automatically puts a frame of 1s around the maze entered by the user.

The program uses two stacks: one to initialize the maze and the other to implement backtracking. The user enters maze data one line at a time. The maze entered by the user

can have any number of rows and any number of columns. The only assumption the program makes is that all rows are of the same length and it uses only these characters: any number of 1s, any number of 0s, one e, and one m. The rows are pushed on the stack `mazeRows` in the order they are entered after attaching one 1 at the beginning and one 1 at the end. After all rows are entered, the size of the array store can be determined, and then the rows from the stack are transferred to the array.

The second stack, `mazeStack`, is used in the process of escaping the maze. To remember untried paths for subsequent tries, the positions of the untried neighbors of the current position (if any) are stored on a stack and always in the same order, first upper neighbor, then lower, then left, and finally right. After stacking the open avenues on the stack, the mouse takes the topmost position and tries to follow it by first storing untried neighbors and then trying the topmost position and so forth, until it reaches the exit or exhausts all possibilities and finds itself trapped. To avoid falling into an infinite loop of trying paths that have already been investigated, each visited position of the maze is marked with a period.

Example:

1) After the user enters the maze.

```
1100
000e
00m1
```

   The maze is immediately surrounded with a frame of 1s.

```
111111
111001
1000e1
100m11
111111
```

   `entryCell` and `currentCell` are initialized to (3, 3) and `exitCell` to (2, 4) (Figure 3a).

2) Because `currentCell` is not equal to `exitCell`, all four neighbors of the current cell (3, 3) are tested, and only two of them are candidates for processing, namely, (3, 2) and (2, 3); therefore they are pushed onto the stack. The stack is checked to see whether it contains any position, and because it is not empty, the topmost position (3, 2) becomes current (Figure 3b).
3) `currentCell` is still not equal to `exitCell`; therefore the two viable options accessible from (3, 2) are pushed onto the stack, namely, positions (2, 2) and (3, 1). Note that the position holding the mouse is not included in the stack. After the

current position is marked as visited, the situation in the maze is as in Figure 3c. Now, the topmost position, (3, 1), is popped off the stack, and it becomes the value of `currentCell`. The process continues until the exit is reached, as shown step by step in Figure 3d through 3f.

Note that in step four (Figure 3d), the position (2, 2) is pushed onto the stack, although it is already there. However, this poses no danger, because when the second instance of this position is popped from the stack, all the paths leading from this position have already been investigated using the first instance of this position on the stack. Note also that the mouse makes a detour, although there is a shorter path from its initial position to the exit.

stack	(3 2) (2 3)	(3 1) (2 2) (2 3)	(2 1) (2 2) (2 3)	(2 2) (2 2) (2 3)	(2 3) (2 2) (2 3)	(2 4) (1 3) (2 2) (2 3)	(1 3) (2 2) (2 3)
`currentCell`	(3 3)	(3 2)	(3 1)	(2 1)	(2 2)	(2 3)	(2 4)
maze	111111 111001 1000e1 100m11 111111	111111 111001 1000e1 10.m11 111111	111111 111001 1000e1 1..m11 111111	111111 111001 1.00e1 1..m11 111111	111111 111001 1..0e1 1..m11 111111	111111 111001 1...e1 1..m11 111111	111111 111001 1...e1 1..m11 111111
	(a)	(b)	(c)	(d)	(e)	(f)	(g)

Figure 3. An example of processing a maze

[Program]

```
import java.io.PrintStream;
import java.util.Stack;

public class Maze {
   private int rows = 0, cols = 0;
   private char[][] store;
   private Cell currentCell, exitCell = new Cell(),
                entryCell = new Cell();
   private final char EXIT_MARKER = 'e', ENTRY_MARKER = 'm',
                      VISITED = '.';
   private final char PASSAGE = '0', WALL = '1';
   private Stack<Cell> mazeStack = new Stack<Cell>();
```

```
public Maze() {
   int row = 0, col = 0;
   Stack<String> mazeRows = new Stack<String>();
   String[] mazeData = new String[] {
      "1100",
      "000e",
      "00m1"
   };

   for (int index = 0; index < mazeData.length; index++) {
      row++;
      mazeData[index] = WALL + mazeData[index] + WALL;
      cols = mazeData[index].length();
      [      A      ];
      if (mazeData[index].indexOf(EXIT_MARKER) != -1) {
         exitCell.x = row;
         exitCell.y = mazeData[index].indexOf(EXIT_MARKER);
      }
      if (mazeData[index].indexOf(ENTRY_MARKER) != -1) {
         entryCell.x = [      B      ];
         entryCell.y = mazeData[index].indexOf(ENTRY_MARKER);
      }
   }

   rows = row;
   store = new char[rows + 2][];
   store[0] = new char[cols];
   for (; !mazeRows.isEmpty(); row--) {
      [      C      ] = mazeRows.pop().toCharArray();
   }
   store[rows + 1] = new char[cols];
   for (col = 0; col < cols; col++) {
      store[0][col] = WALL;
      store[rows + 1][col] = WALL;
   }
}

private void display(PrintStream out) {
   for (int row = 0; row <= rows + 1; row++) {
      out.println(store[row]);
   }
   out.println();
}
```

```
private void pushUnvisited(int row, int col) {
   if (store[row][col] == PASSAGE ||
       store[row][col] == EXIT_MARKER) {
      [      D      ];
   }
}

void exitMaze(PrintStream out) {
   currentCell = entryCell;
   out.println();
   while (!currentCell.isSame(exitCell)) {
      int row = [      E      ];
      int col = [      F      ];
      display(System.out); // print a snapshot
      if (!currentCell.isSame(entryCell)) {
         store[row][col] = VISITED;
      }
      [      G      ];
      pushUnvisited(row + 1, col);
      pushUnvisited(row, col - 1);
      [      H      ];
      if (mazeStack.isEmpty()) {
         display(out);
         out.println("Failure");
         return;
      } else {
         currentCell = mazeStack.pop();
      }
   }
   display(out);
   out.println("Success");
}

// nested class
static class Cell {
   int x, y;

   Cell() {
   }

   Cell(int i, int j) {
      x = i;
      y = j;
   }
```

```
        boolean isSame(Cell cell) {
            return x == cell.x && y == cell.y;
        }
    }

    public static void main(String args[]) {
        (new Maze()).exitMaze(System.out);
    }
}
```

Subquestion

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

Answer group for A through C

a) `cols`
b) `mazeRows.pop(mazeData[index])`
c) `mazeRows.push(mazeData[index])`
d) `mazeRows.push(row)`
e) `mazeStack.push(row)`
f) `row`
g) `store[cols]`
h) `store[row]`

Answer group for D through F

a) `currentCell.col`
b) `currentCell.row`
c) `currentCell.x`
d) `currentCell.y`
e) `mazeStack.pop(new Cell(row, col))`
f) `mazeStack.push(new Cell(col))`
g) `mazeStack.push(new Cell(row, col))`
h) `mazeStack.push(row)`

Answer group for G and H

a) `entryCell= currentCell`
b) `pushUnvisited(row - 1, col)`
c) `pushUnvisited(row - 1, col + 1)`
d) `pushUnvisited(row, col + 1)`
e) `pushUnvisited(row + 1, col - 1)`
f) `pushUnvisited(x, y)`