



April 2017

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 **(S)** 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	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
--------	-----------------------	----------------------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------


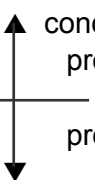



Do not open the exam booklet until instructed to do so.

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 the text.
Process	<ul style="list-style-type: none"> variable ← expression 	Assigns the value of the expression to the variable.
	<ul style="list-style-type: none"> procedure(argument, ...) 	Calls the procedure and passes / receives the argument.
		Indicates a one-way selection process. If the conditional expression is true, then the process is executed.
		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.
		Indicates a pre-test iteration process. While the conditional expression is true, the process is executed repeatedly.
		Indicates a post-test iteration process. The process is executed, and then while the conditional expression is true, the process is executed repeatedly.
		Indicates an iteration process. The initial value init (given by an expression) is stored in the variable at the start of the iteration process, and then while the conditional expression cond is true, the process is executed repeatedly. The 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	<div style="text-align: center;"> High ↑ ↓ Low </div>
Multiplication, division	×, ÷, %	
Addition, subtraction	+, -	
Relational operation	>, <, ≥, ≤, =, ≠	
Logical product	and	
Logical sum	or	

Note: With division of integers, an integer quotient is returned as a result.

The “%” operator indicates a remainder operation.

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

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

As a method of exchange of 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.

To establish a shared secret key, the two users must first select two numbers: a prime number p and an integer g that is a primitive root of p . The number g is a primitive root of p when for each n ($n = 1, 2, \dots, p-1$), and each value of n ($n = 1, 2, \dots, p-1$) appears in $g^n \bmod p$. Table 1 shows an example of primitive root when $p = 11$ and $g = 2$. The bottom row shows the numbers $1, 2, \dots, 10$.

Table 1 Example of primitive root when $p = 11$ 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
$2^n \bmod 11$	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 that is a primitive root of p .
- (2) Alice selects a random number x ($1 \leq x \leq p-1$), and calculates $R1 = g^x \bmod p$.
- (3) Bob selects another random number y ($1 \leq y \leq p-1$), and calculates $R2 = g^y \bmod p$.
- (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 $K = (R2)^x \bmod p$.
- (7) Bob calculates the shared secret key $K = (R1)^y \bmod p$.

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

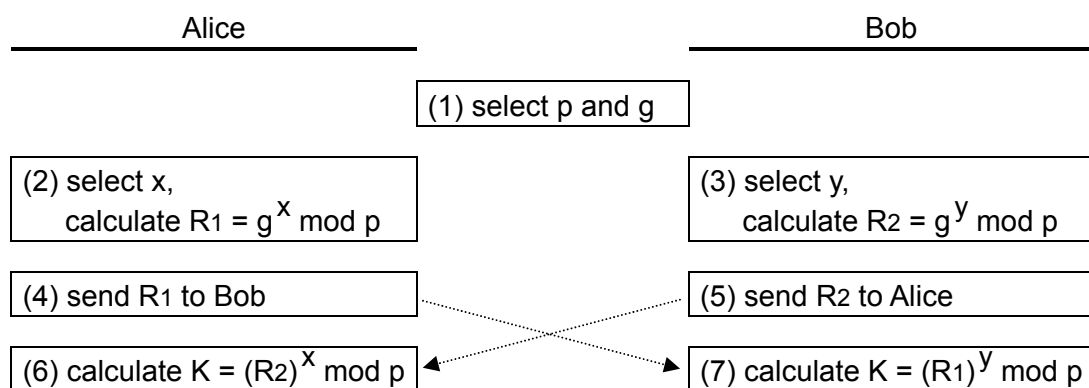


Figure 1 Diffie-Hellman key exchange algorithm

Subquestion 1

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

A shared secret key is being exchanged between Alice and Bob using this algorithm. Assume that they agreed upon the prime number $p = 11$ and the integer $g = 2$.

If Bob has received the public key $R_1 = 3$ from Alice, the random number x Alice has selected is A . Also, if Alice has received the public key $R_2 = 9$ from Bob, the random number y Bob has selected is B .

In this case, i.e. $R_1 = 3$ and $R_2 = 9$, the shared secret key obtained by both Alice and Bob is C , which they can use for subsequent symmetric encryption of messages.

Answer group

- | | | | |
|------|------|------|------|
| a) 2 | b) 3 | c) 4 | d) 5 |
| e) 6 | f) 7 | g) 8 | h) 9 |

Subquestion 2

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

Alice and Bob decided to change the values of g , x , and y . Then, they exchanged the recalculated R_1 and R_2 , and finally obtained the shared secret key K .

Assume that an attacker Eve knows the three non-secret values; $p = 11$, $g = 7$, and $R_1 = 3$. Recently, Eve obtains the value $y = 3$ by illegal means. Then, it would be possible for Eve to determine the shared secret key K .

Eve finds out that the shared secret key K is D . Furthermore, by referring to Table 2, Eve finds out that the value of x is E , and the value of R_2 is F .

Table 2 Table of $7^n \bmod 11$

n	1	2	3	4	5	6	7	8	9	10
$7^n \bmod 11$	7	5	2	3	10	4	6	9	8	1

Answer group

- | | | | |
|------|------|------|------|
| a) 2 | b) 3 | c) 4 | d) 5 |
| e) 6 | f) 7 | g) 8 | h) 9 |

Q2. Read the following description of a door controller circuit, and then answer Subquestions 1 and 2.

A sensor-based automatic door controller has three states of operation: “closed”, “opened” and “closing”. When the sensor detects a person coming towards the door or going through the door, the door enters the “opened” state. When the person has passed through the door, the sensor detects nothing and the door enters the “closing” state. Within the “closing” state, another person may approach towards the door, causing the change in the state of the door from “closing” to “opened”. The idle state “closed” implies that the door is completely closed.

Variable S represents the signal generated by the sensor, which is used as an input for the door controller circuit. The sensor generates the signal $S = 1$ when the sensor detects a person either approaching to or passing through the door; otherwise, it generates the signal $S = 0$. Figure 1 shows the operational state transition diagram of the door controller.

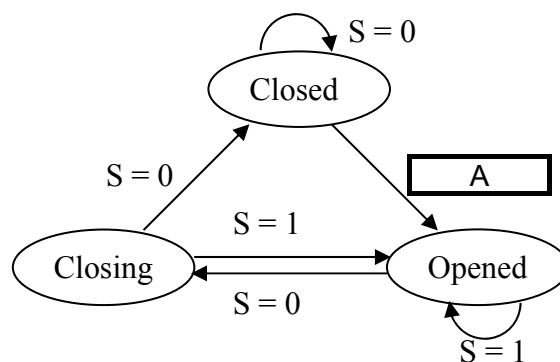


Figure 1 State transition diagram of the door controller

In designing the door controller circuit, 2-digit binary numbers 00, 01, and 10 are assigned to the states “closed”, “opened”, and “closing”, respectively. The higher bit is denoted by Q_H and the lower bit by Q_L . For example, the state “closing” (10) is represented as $Q_H = 1$ and $Q_L = 0$. With this notation, Table 1 is equivalent to Figure 1.

Table 1 State transition table of the door controller

Current state		Signal	Next state	
Q_H	Q_L	S	Q_H	Q_L
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	B	

A hardware engineer creates the program `SetNextState` that determines the next states of Q_H and Q_L based on Table 1. The program is executed at regular intervals triggered by the clock-generated pulses. Assume that during the program execution, the contents of Q_H , Q_L , and S are not changed nor fetched by external processes.

- Global: Bit type: Q_H , Q_L , S
- Program: `SetNextState`
 - $Q_H \leftarrow$ C
 - $Q_L \leftarrow S$
 - return

Subquestion 1

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

Answer group for A

a) $S = 0$

b) $S = 1$

Answer group for B

a) 0 0

b) 0 1

c) 1 0

d) 1 1

Answer group for C

a) $((\text{not } Q_H) \text{ or } Q_L) \text{ and } (\text{not } S)$

b) $(Q_H \text{ and } S) \text{ or } (Q_L \text{ and } (\text{not } S))$

c) $(\text{not } Q_L) \text{ and } S$

d) $Q_L \text{ and } (\text{not } S)$

Subquestion 2

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

The door controller system consists of three components as shown in Figure 2.

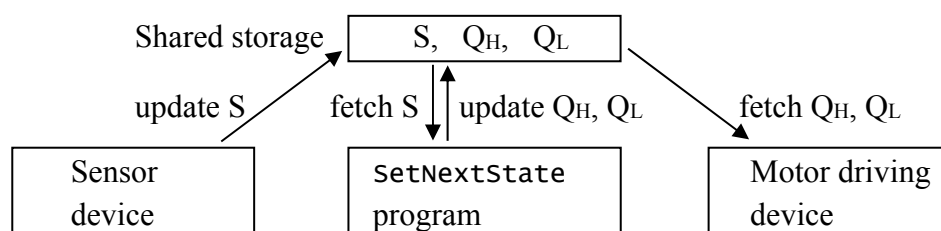


Figure 2 Door controller system

The door controller system operates normally on condition that the execution of the three components is well controlled. The description below explains a problem that may occur when the program `SetNextState` and the motor driving device operate independently.

The program `SetNextState` first updates Q_H and then updates Q_L . Therefore, there is a very short time lag between the updates of Q_H and Q_L . During this time lag, if the motor driving system fetches the values of Q_H and Q_L , it may result to a problem. As an example, the motor driving system receives an undefined state 11 ($Q_H = 1$ and $Q_L = 1$) when the program updates the state of the door D.

By E, for example, the time lag problem mentioned above can be resolved.

Answer group for D

- | | |
|-------------------------------|-------------------------------|
| a) from “closed” to “opened” | b) from “closing” to “closed” |
| c) from “closing” to “opened” | d) from “opened” to “closing” |

Answer group for E

- a) changing the data format of Q_H and Q_L so that they can be fetched or updated by one hardware instruction
- b) changing the execution logic of the motor driving device so that it will ignore the undefined state 11
- c) changing the execution sequence of the program `SetNextState` so that the program first updates Q_L and then updates Q_H
- d) replacing the shared storage device with a new one so that Q_H and Q_L can be fetched or updated much faster than the current device

Q3. Read the following description of resource access control, and then answer Subquestion.

Resource access control is an important task that must be provided by operating systems. When designing a mechanism of process execution control that needs resource access control, the “semaphore” concept is useful.

Semaphore is a special variable that holds the number of copies of the specific resource that is currently available. Working on the specific semaphore named S , two functions $\text{wait}(S)$ and $\text{signal}(S)$ are provided. A process that needs the resource executes $\text{wait}(S)$, and a process that supplies the resource executes $\text{signal}(S)$.

(1) Function $\text{wait}(S)$ performs the following operations:

Decrement the value of S by 1.

If $S < 0$ after the decrement, block the process that is executing $\text{wait}(S)$.

(2) Function $\text{signal}(S)$ performs the following operations:

Increment the value of S by 1.

If $S \leq 0$ after the increment, unblock one process that has been blocked by $\text{wait}(S)$.

Figures 1 through 3 show how semaphore S works upon three processes P_1 , P_2 and R . The system shown in the Figures consists of a processor, ready queue, wait queue, and semaphore S . P_1 and P_2 need the resource and R supplies the resource. When P_1 or P_2 executes $\text{wait}(S)$, the process is interrupted and placed into the wait queue or ready queue depending on the value of S . When the process is loaded again for execution, the rest of the process is executed without executing $\text{wait}(S)$, which is deleted from the system. R executes $\text{signal}(S)$ at the end of its process, and then R is deleted from the system.

Figure 1 shows the current state where P_2 is running. R and P_1 are in the ready queue, and no process is in the wait queue. The resource supplied by R is not available because $S = 0$.

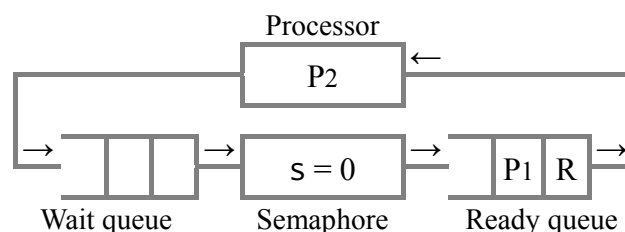


Figure 1 P_2 is running

After P_2 executes $\text{wait}(S)$ in Figure 1, S is changed to -1 , and P_2 is blocked because $S < 0$. Subsequently, P_2 is placed into the wait queue, and R is picked up from the ready queue for execution (Figure 2).

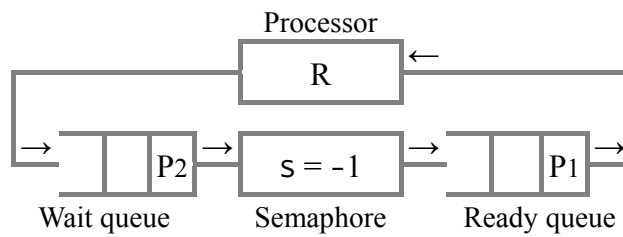


Figure 2 After P2 executes wait(S) in Figure 1

After R executes signal(S) in Figure 2, S is changed to 0, and R is ended and deleted. P2, which has been blocked by wait(S), is unblocked because $s \leq 0$. Subsequently, P2 is moved to the ready queue, and then P1 is picked up from the ready queue for execution (Figure 3).

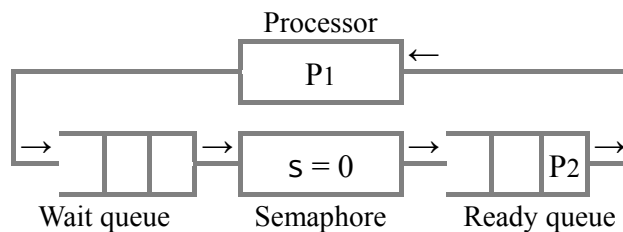


Figure 3 After R executes signal(S) in Figure 2

Subquestion

From the answer groups below, select the correct answer to be inserted in each blank in Figures 6 and 7.

Figures 4 through 7 show another example with four processes P1, P2, P3 and R. P1, P2 and P3 need the resource and R supplies the resource.

Figure 4 shows the current state where P2 is running. P3, R, and P1 are in the ready queue, and no process is in the wait queue. Here, P1, which is in the ready queue, has already executed wait(S).

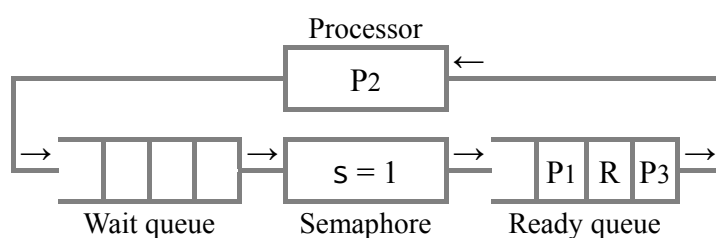


Figure 4 P2 is running

After P2 executes `wait(S)` in Figure 4, `s` is changed to 0, and P2 is not blocked because $s \geq 0$. Subsequently, P2 is placed into the ready queue because it is not blocked, and P3 is picked up from the ready queue for execution (Figure 5).

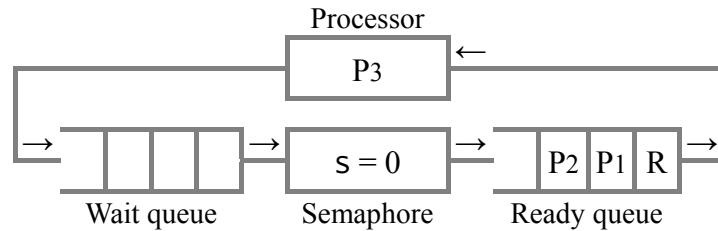


Figure 5 After P2 executes `wait(S)` in Figure 4

After P3 executes `wait(S)` in Figure 5, `s` is changed, and the queues are updated if necessary. R is picked up from the ready queue for execution (Figure 6).

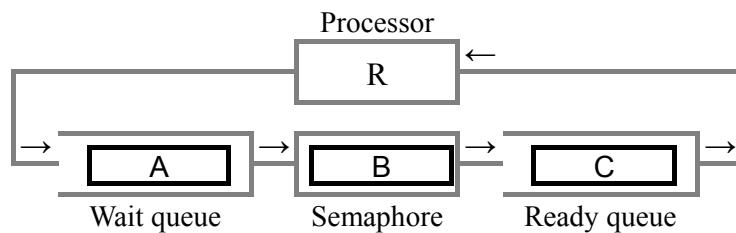
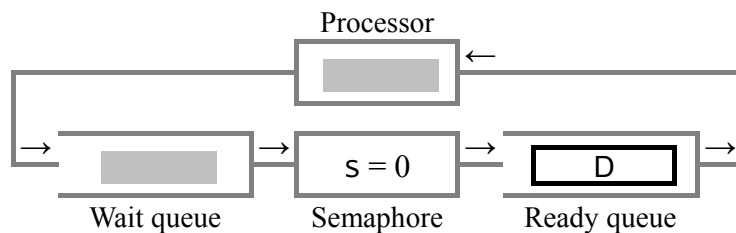


Figure 6 After P3 executes `wait(S)` in Figure 5

After R executes `signal(S)` in Figure 6, `s` is changed, and the queues are updated if necessary. The next process is picked up from the ready queue for execution (Figure 7).



Note: Shaded parts are not shown.

Figure 7 After R executes `signal(S)` in Figure 6

Answer group for A, C and D

a)

--	--	--	--

c)

			P3
--	--	--	----

e)

		P3	P2
--	--	----	----

b)

			P2
--	--	--	----

d)

		P2	P1
--	--	----	----

f)

	P3	P2	P1
--	----	----	----

Answer group for B

a) $s = -1$

c) $s = 1$

b) $s = 0$

d) $s = 2$

- Q4.** Read the following description of a relational database used in an airline company, and then answer Subquestions 1 and 2.

Fly4You is an airline company that operates domestic and international flights. The airline records information of its daily flights in the table Flight. The following table shows the structure of the table Flight with its sample data.

Table: Flight

FlightID	Segment Number	Origin Airport	Depart Time	Dest Airport	Arrive Time	Meal	Flying Time	Miles	Aircraft
4U221	1	MDL	08:30	RGN	09:55	-	1:25	357	Y747
4U221	2	RGN	12:30	DAC	14:15	L	2:15	605	Y747
4U321	1	BKK	08:00	KUL	11:10	B	2:10	748	Y747
4U380	1	ULN	09:45	NRT	15:30	L	4:45	1887	X757
4U380	2	NRT	18:15	MNL	22:20	D	5:05	1879	X757
4U536	1	MNL	10:30	SGN	12:50	L	3:20	1001	X757

Each flight is identified by flight ID. There are two types of flights: direct flight and one-stop flight. One-stop flight is a flight that stops at one intermediate airport before arriving at the final destination airport.

A direct flight is registered in one row of record with segment number 1. One-stop flight is registered in two rows of records with segment numbers 1 and 2. For example, in the above sample data, flight 4U221 is a one-stop flight from MDL to DAC via RGN. The first record (segment number 1) contains the flight information from the origin airport MDL to the intermediate airport RGN, and the second record (segment number 2) contains the flight information from the intermediate airport RGN to the destination airport DAC.

Subquestion 1

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

To control flight schedules at intermediate airports, the management wants to generate a report that displays all of the one-stop flights registered in the table Flight.

For this purpose, the SQL statement “SQL1” is created.

```
-- SQL statement "SQL1"
SELECT FlightID, SegmentNumber, OriginAirport, DepartTime,
       DestAirport, ArriveTime
FROM Flight
WHERE FlightID IN ( SELECT FlightID FROM Flight
                   GROUP BY 
                   HAVING  = 2 )
```

When the table Flight contains the above sample data, SQL1 generates the following report:

FlightID	Segment Number	Origin Airport	Depart Time	Dest Airport	Arrive Time
4U221	1	MDL	08:30	RGN	09:55
4U221	2	RGN	12:30	DAC	14:15
4U380	1	ULN	09:45	NRT	15:30
4U380	2	NRT	18:15	MNL	22:20

Answer group

- a) COUNT(*)
- b) FlightID
- c) SegmentNumber
- d) SUM(FlightID)
- e) SUM(SegmentNumber)

Subquestion 2

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

Fly4You airline manages flight reservation information in table Reservation. The table Reservation maintains the latest number of seats reserved by seat classes. There are two types of seat classes; business class and economy class. A record of the table Reservation indicates the reservation status of one segment of one flight on a specific day.

Furthermore, different types of aircraft have different number of seats by seat classes. Fly4You airline manages the number of seats by seat classes by aircraft in the table Aircraft. The structures of the table Reservation and table Aircraft are as follows:

Reservation (FlightID, SegmentNumber, FlightDate,
BusinessSeatsReserved, EconomySeatsReserved)

Aircraft (Aircraft, BusinessSeats, EconomySeats)

To reduce flight costs, Fly4You airline wants to allocate smaller aircrafts for high-vacancy-rate flights routes. For this purpose, the SQL statement “SQL2” is created, which generates a report that displays all of the flight segments whose vacancy rate (percentage of seats that are not reserved) of economy class is 50% and higher on a specific day, in the descending order of the vacancy rate. Here, the specific date is given by the host variable :SpDate.

```
-- SQL statement "SQL2"
SELECT F.FlightID, F.SegmentNumber, OriginAirport, DepartTime,
       DestAirport, ArriveTime,  AS VacantRate
FROM Reservation R
JOIN Flight F
  ON 
JOIN Aircraft A
  ON F.Aircraft = A.Aircraft
WHERE  AND  >= 50
ORDER BY VacantRate DESC
```

An example of the report created by “SQL2” is as follows:

FlightID	Segment Number	Origin Airport	Depart Time	Dest Airport	Arrive Time	Vacant Rate
4U380	1	ULN	09:45	NRT	15:30	71
4U536	1	MNL	10:30	SGN	12:50	64
4U221	2	RGN	12:30	DAC	14:15	57
4U321	1	BKK	08:00	KUL	11:10	50

Answer group for C

- a) $(100 - \text{EconomySeatsReserved}) / \text{EconomySeats} * 100$
- b) $1 - \text{EconomySeatsReserved} / \text{EconomySeats} * 100$
- c) $100 * \text{EconomySeatsReserved} / \text{EconomySeats}$
- d) $100 - 100 * \text{EconomySeatsReserved} / \text{EconomySeats}$

Answer group for D and E

- a) $\text{F.Aircraft} = \text{A.Aircraft}$
- b) $\text{FlightDate} = \text{:SpDate}$
- c) $\text{R.FlightID} = \text{F.FlightID}$
- d) $\text{R.FlightID} = \text{F.FlightID}$ AND $\text{R.SegmentNumber} = \text{F.SegmentNumber}$
- e) $\text{R.SegmentNumber} = \text{F.SegmentNumber}$

- Q5.** Read the following description of processing orders for purchase, and then answer Subquestions 1 and 2.

Company X is a chain of stores located in several cities. Weekly, each of the stores places orders for their supplies to a purchasing office where suppliers also send their quotations. All orders are placed and fulfilled weekly.

Given that there are several stores in every city, the purchasing office consolidates orders for every city. The suppliers selected are in the same city the order is placed, and those with the lowest price are given priority. This practice yielded enormous profit to Company X. The manager creates a program to match the orders with the suppliers and create a file containing the orders to be placed.

[Program Description]

- (1) The file Order contains the information on quantities to be ordered by city and by item.

File: Order (O_City, O_Item, O_Qty)

The fields from left to right are the city code, item code, and quantity. The quantity is the consolidated quantity ordered from each store in the city. The file Order is a sequential file. Records are sorted in ascending order of the city code and item code.

- (2) The file Supply contains the information on stocked quantities and offered prices by city, by item, and by supplier.

File: Supply (S_City, S_Item, S_SupID, S_Qty, S_Price)

The fields from left to right are the city code, item code, supplier ID, quantity, and price. Quantity is the maximum stocked quantity the supplier can deliver for a week. The file Supply is a sequential file. Records are sorted in ascending order of the city code, item code, and price.

- (3) The file Purchase will contain the information on quantities and prices to be purchased by city, by item, and by supplier.

File: Purchase (P_City, P_Item, P_SupID, P_Qty, P_Price)

The fields from left to right are the city code, item code, supplier ID, quantity, and price. If two or more suppliers exist for a specific city and item, the supplier with the lowest price is given priority. The file Purchase is a sequential file created by the program.

- (4) Figure 1 shows the process flow with sample data.

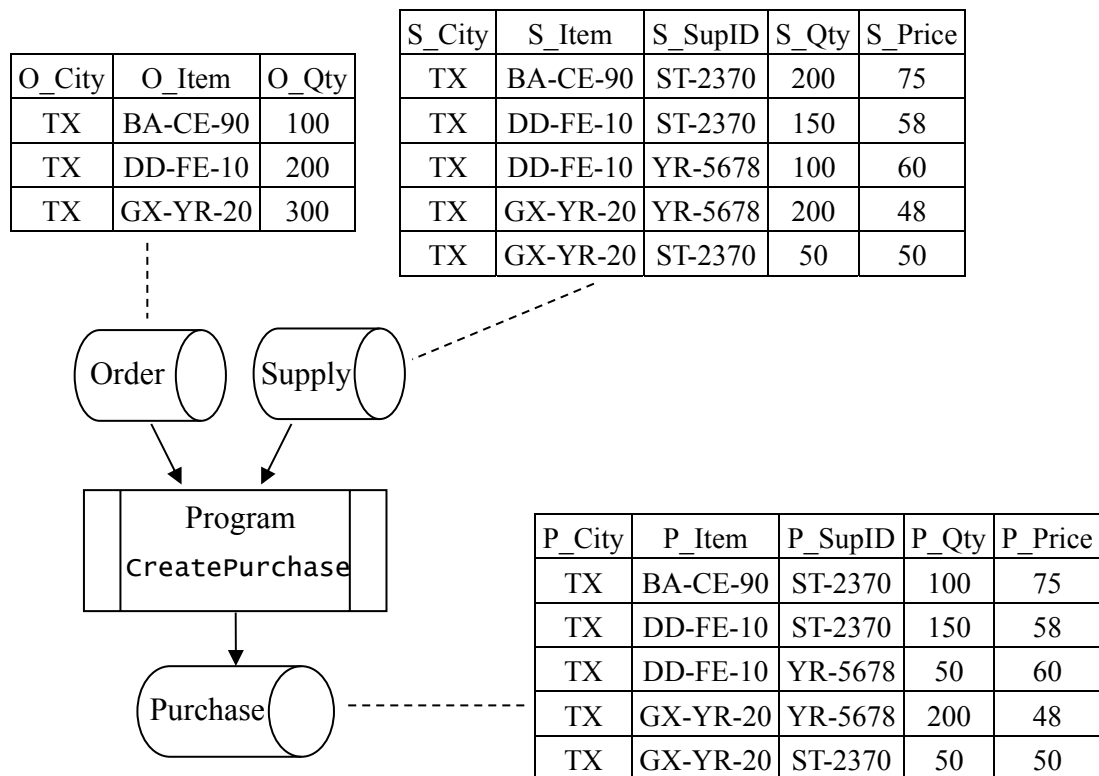


Figure 1 process flow with sample data

In Figure 1, for example, items BA-CE-90 and DD-FE-10 are processed as follows:

BA-CE-90: The purchasing office wants to purchase 100 BA-CE-90s, and supplier ST-2370 has enough (200) stocks; therefore, 100 BA-CE-90s are ordered from ST-2370.

DD-FE-10: The purchasing office wants to purchase 200 DD-FE-10s, but the first supplier ST-2370 has only 150 stocks; therefore, 150 DD-FE-10s are ordered from ST-2370, and the remaining 50 are ordered from the second supplier YR-5678.

(5) In the program, the following statements are used:

- `open filename) atEOF.eof_process)`

Open the file *filename*. In subsequent read operations, if end of file is reached, *eof_process* is executed. The value "|EOF|" is higher than any other city code.

- `read(filename) into(field1, field2, ...)`

Read the next record from the file *filename*. The fields of the record are stored into the variables *field1*, *field2*,

- `write(filename) from(value1, value2, ...)`

Write the next record into the file *filename*. The fields of the record are constructed from the values *value1*, *value2*,

- `close file(filename)`

Close the file *filename*.

[Program]

○ Program: CreatePurchase

○ String type: O_City, O_Item, S_City, S_Item, S_SupID

○ Numeric type: O_Qty, S_Qty, S_Price

- open file(Order) atEOF(O_City ← "|EOF|")
- open file(Supply) atEOF(S_City ← "|EOF|")
- open file(Purchase)
- read(Order) into(O_City, O_Item, O_Qty)
- read(Supply) into(S_City, S_Item, S_SupID, S_Qty, S_Price)

■ O_City ≠ "|EOF|"

▲ O_City = S_City and O_Item = S_Item

■ O_City = S_City and O_Item = S_Item and O_Qty > 0

▲ O_Qty ≤ S_Qty

- write(Purchase) from(S_City, S_Item, S_SupID, A, S_Price)

• O_Qty ← 0

- write(Purchase) from(S_City, S_Item, S_SupID, S_Qty, S_Price)

• O_Qty ← B

- read(Supply) into(S_City, S_Item, S_SupID, S_Qty, S_Price)

α →

■

β →

- read(Order) into(O_City, O_Item, O_Qty)

▲

γ →

C

- read(Order) into(O_City, O_Item, O_Qty)

▲

δ →

D

- read(Supply) into(S_City, S_Item, S_SupID, S_Qty, S_Price)

■

- close file(Purchase)
- close file(Supply)
- close file(Order)

Subquestion 1

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

Answer group for A and B

- a) O_Qty
- c) S_Qty

- b) O_Qty - S_Qty
- d) S_Qty - O_Qty

Answer group for C and D

- a) (O_City < S_City) and (O_Item < S_Item)
- b) (O_City < S_City) or (O_City = S_City and O_Item < S_Item)
- c) (O_City < S_City) or (O_Item < S_Item)
- d) (O_City > S_City) and (O_Item > S_Item)
- e) (O_City > S_City) or (O_City = S_City and O_Item > S_Item)
- f) (O_City > S_City) or (O_Item > S_Item)

Subquestion 2

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

To control out of stock of items, the purchasing office wants to print the following error messages (1) and (2):

- (1) "Item *ii* for city *cc*: cannot be fulfilled fully"
- (2) "Item *ii* for city *cc*: cannot be fulfilled partially"

- (1) The case where the file Supply does not contain the record that has item *ii* and city *cc*.
- (2) The case where the total of each supplier's quantity for item *ii* and city *cc* is less than the ordered quantity.

To implement message (1), insert the following statement to the program immediately before the read statement pointed by E.

```
• print("Item ", O_Item, " for city ", O_City,
      ": cannot be fulfilled fully")
```

To implement message (2), insert the following statements to the program immediately before the read statement pointed by F.

```
↑ O_Qty > 0
| • print("Item ", O_Item, " for city ", O_City,
|   ": cannot be fulfilled partially")
↓
```

Answer group for E and F

- a) α
- b) β
- c) γ
- d) δ

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

The rod cutting problem is a classic problem in combinatorial optimization. There is a rod with a length of N units. This rod can be cut into several pieces with length smaller than N . Here, the prices of all pieces are given. The problem is to determine the maximum revenue obtainable by cutting up the rod and selling the pieces. Figure 1 shows an example of a rod with a length of 4 and the total revenues corresponding to the possible combinations of cutting up the rod. In this case, the value of an optimal solution (maximum revenue) is 13, and the sizes of the pieces for an optimal solution to cut off are 1 and 3.

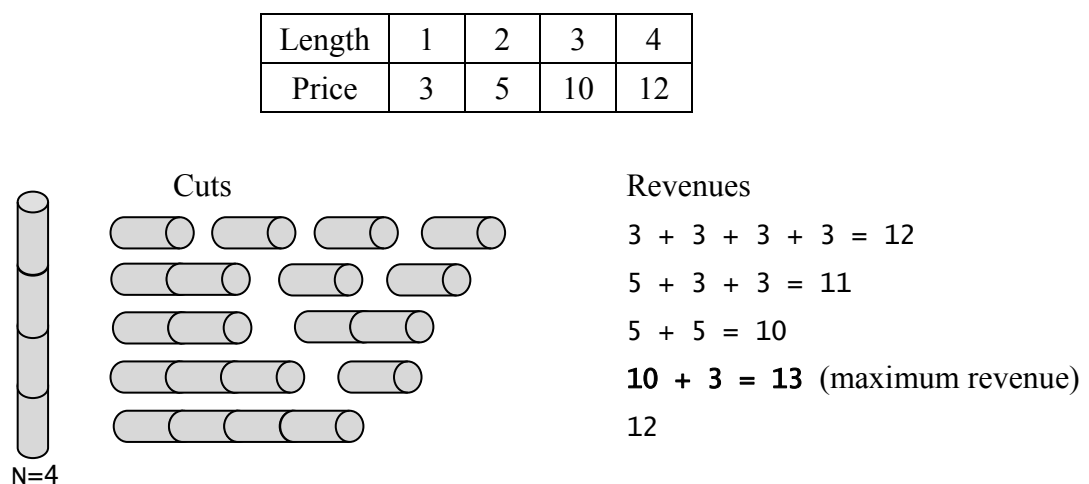


Figure 1 Sample rod with a length of $N = 4$

[Program Description]

- (1) N contains the length of the rod.
- (2) Indexes of arrays $P[]$ and $S[]$ start at 1. An index of array $R[]$ starts at 0.
- (3) $P[i]$ contains the price of the piece of length i .
- (4) $R[]$ contains a list of values of an optimal solution of each sub-problem. $R[i]$ contains the value of an optimal solution when the length of the rod is i . In Figure 1, for example, when the length of the rod is 2, the optimal solution is 6 ($3 + 3$); therefore, $R[2]$ is 6 in this case.
- (5) The subprogram `rodCut()` finds an optimal solution of the problem using the bottom-up method of dynamic programming. To solve the problem, the subprogram `rodCut()` solves the same type of sub-problems in smaller sizes starting with the smallest and increasing according to their size. When solving a particular sub-problem, all of the smaller sub-problems have already been solved, and their solutions are saved in array $R[]$. To solve the problem of size 1, a solution of the problem of size 0 is needed. Therefore, the index of array $R[]$ starts at 0.

- (6) $S[]$ contains a list of the optimal size of the first piece to cut off in each sub-problem. This array helps to reconstruct the sizes of the pieces in an optimal solution.
- (7) Table 1 shows the execution results of the subprogram `rodCut()` in the case shown in Figure 1.

Table 1 Execution results in the case shown in Figure 1

N	1	2	3	4
P[N]	3	5	10	12
R[N]	3	6	10	13
S[N]	1	1	3	1

- (8) The subprogram `print()` displays the contents of an array, and the subprogram `displayPiece()` displays the complete list of sizes for each piece in an optimal solution from array $S[]$. In the case shown in Figure 1, the sizes of the piece of an optimal solution to cut off are 1 and 3. Hence, the subprogram `displayPiece()` displays the result "1 3 ".

[Program]

○ Program: `RodCutSolver`

○ Global: Integer type: $N \leftarrow 4$

○ Global: Integer type: $P[N] \leftarrow \{3, 5, 10, 12\}$

○ Global: Integer type: $R[N+1], S[N]$

$\left. \begin{array}{l} \\ \\ \end{array} \right] \leftarrow \alpha$

○ Subprogram: `rodCut()`

○ Integer type: I, J, Q

• $R[0] \leftarrow 0$

■ $I: 1, I \leq N, 1$

• $Q \leftarrow 0$

■ $J: 1, J \leq \boxed{A}, 1$

▲ $Q < P[J] + R[\boxed{B}]$

• $Q \leftarrow P[J] + R[\boxed{B}]$

• $S[I] \leftarrow J$

▼

■

• $R[I] \leftarrow Q$

■

○ Subprogram: `displayPiece()`

■ $N > 0$

• `print($S[N] + " "$)`

• $N \leftarrow \boxed{C}$

■

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 and B

- | | |
|----------|----------|
| a) I | b) I - J |
| c) J | d) J - I |
| e) N | f) N - I |
| g) N - J | |

Answer group for C

- | | |
|-----------------|--------------------|
| a) N - 1 | b) N - S[N] |
| c) N - S[N - 1] | d) S[N] - S[N - 1] |
| e) S[N - S[N]] | |

Subquestion 2

From the answer groups below, select the correct answer to be inserted in each blank in Table 2. Here, assume that correct answers are inserted in blanks A through C in the program.

To simulate another case of data, two statements marked by " $\leftarrow \alpha$ " are replaced with the following statements:

- Global: Integer type: $N \leftarrow 5$
- Global: Integer type: $P[N] \leftarrow \{3, 5, 10, 14, 16\}$

Then, the subprogram rodCut() is executed. Table 2 shows the execution results.

Table 2 Execution results in the case described above

N	1	2	3	4	5
P[N]	3	5	10	14	16
R[N]	3	6	<input type="text"/>	14	<input type="text"/>
S[N]	1	1	3	4	<input type="text"/>

Answer group for D and E

- | | | | |
|-------|-------|-------|-------|
| a) 8 | b) 10 | c) 11 | d) 13 |
| e) 16 | f) 17 | g) 18 | h) 19 |

Answer group for F

- | | | | |
|------|------|------|------|
| a) 1 | b) 2 | c) 4 | d) 5 |
|------|------|------|------|

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

Then, mark the **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 Subquestions 1 and 2.

[Program Description]

This program counts words from a sentence. Here, a word means a character string (hereinafter, a string) with no white-space in it. For instance, in the input string “I love C programming”, there are four words, i.e. “I”, “love”, “C” and “programming”.

- (1) An input string is stored in the string `line` (see the program). The length of the input string is greater than 0 and less than 100. Each word in the input string will have at most 50 characters. The input string does not start with a white-space. The words in the input string are separated by exactly one white-space in between them and the last word of the input string does not have any white-space at the end of it.
- (2) The program outputs all the unique words in the input string in ASCII-code order (uppercase characters come before lowercase characters) followed by the frequency of their occurrence separated by a white-space between them. The words are grouped in the ascending order of the frequency of the words, that is, the words with the lowest frequency will appear first.
- (3) When the string `line` contains the following string,

I love C programming and my friend loves C programming too

the program outputs the following list of words:

```
I 1
and 1
friend 1
love 1
loves 1
my 1
too 1
C 2
programming 2
```

- (4) In the program, there is one user defined structure named `word`. This structure has an array of character variable `str` to store unique word, and an integer variable `frequency` to store the frequency of the unique word. Here, the array of structure `word[]` is a global variable.
- (5) The following four functions are used in the program.
- (i) `void analyse(char* line)`
This function separates each word in the string `line` and adds them in the array `word`.
 - (ii) `void setCount(char* s)`
This function checks the uniqueness of the current word `s`. If `s` is a new word, then it is added to the array `word`, and the `frequency` is set to 1. If `s` is not unique, then the `frequency` in the existing word entry is increased by 1. In both cases, if the `frequency` is greater than the `highestFreq`, then `highestFreq` is updated with the value of `frequency`.
 - (iii) `void sortBywords()`
This function sorts the array `word` in ASCII-code order of the variable `str`.
 - (iv) `void outputwords()`
This function outputs the words in the array `word`. Here, the words with lower `frequency` appear early in ASCII-code order.
- (6) The following two library functions are used in the program.
- (i) `int strcmp(const char* s1, const char* s2)`
This function compares the string pointed by `s1` and the string pointed by `s2`. When `s1 < s2`, it returns a negative value; when `s1 = s2`, it returns 0; when `s1 > s2`, it returns a positive value.
 - (ii) `char* strcpy(char* s1, const char* s2)`
This function copies the string pointed by `s2` (including the null character) to the character array destination `s1`. It returns the pointer to the character array `s1`.

[Program]

```
#include <stdio.h>
#include <string.h>

#define MAX_WORD_LEN 50 + 1
#define MAX_INPUT_LEN 100 + 1
```



```

void analyse(char*);
void setCount(char*);
void sortByWords();
void outputWords();

struct word {
    char str[MAX_WORD_LEN];
    int frequency;
} word[MAX_INPUT_LEN];

int wordCount, highestFreq;

int main() {
    char line[MAX_INPUT_LEN];

    strcpy(line, "Apple apple red apple green apple");
    analyse(line);
    sortByWords();
    outputWords();
    return 0;
}

void analyse(char* line) {
    int i = 0, j = 0;
    char s[MAX_WORD_LEN];

    wordCount = 0;
    highestFreq = 0;

    while (line[i]) {
        if (  ) {
            s[j] = '\0';
            setCount(s);
            j = 0;
            i++;
        } else {
            ;
        }
    }
    s[j] = '\0';
    setCount(s);
}

```

```

void setCount(char* s) {
    int i;
    for(i = 0; i < wordCount; i++) {
        if (!strcmp(word[i].str, s)) {
            word[i].frequency++;
            if (word[i].frequency > highestFreq)
                highestFreq = word[i].frequency;    //  $\leftarrow \alpha$ 
            return;
        }
    }
    strcpy(word[wordCount].str, s);                //  $\leftarrow \beta$ 
    word[wordCount].frequency = 1;
    if ( C ) {
        highestFreq = 1;
    }
    wordCount++;
}

void sortByWords() {
    int i, j;
    struct word temp;
    for(i = 0; i < wordCount - 1; i++) {
        for(j = i + 1; j < wordCount; j++) {
            if (strcmp(word[i].str, word[j].str) > 0) {
                temp = word[j];
                word[j] = word[i];
                word[i] = temp;
            }
        }
    }
}

void outputwords() {
    int i, j;
    for(i = 1; i <= D; i++) {
        for(j = 0; j < E; j++) {
            if (word[j].frequency == i)
                printf("%s %d\n", word[j].str, word[j].frequency);
        }
    }
}

```

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) <code>line[i] == ' '</code> | b) <code>line[i] == '0'</code> |
| c) <code>line[i] == '\0'</code> | d) <code>s[j] == ' '</code> |
| e) <code>s[j] == '0'</code> | f) <code>s[j] == '\0'</code> |

Answer group for B

- | | |
|------------------------------------|------------------------------------|
| a) <code>s[++j] = line[++i]</code> | b) <code>s[j++] = line[i++]</code> |
| c) <code>s[j++] = line[i]</code> | d) <code>s[j] = line[i++]</code> |
| e) <code>s[j] = line[i]</code> | |

Answer group for C

- | | |
|------------------------------------|--|
| a) <code>highestFreq > 1</code> | b) <code>wordCount < highestFreq</code> |
| c) <code>wordCount == 0</code> | d) <code>wordCount == highestFreq</code> |
| e) <code>wordCount > 0</code> | f) <code>wordCount > highestFreq</code> |

Answer group for D and E

- | | |
|-----------------------------------|-----------------------------------|
| a) <code>highestFreq</code> | b) <code>highestFreq - 1</code> |
| c) <code>wordCount</code> | d) <code>wordCount - 1</code> |
| e) <code>word[i].frequency</code> | f) <code>word[j].frequency</code> |

Subquestion 2

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

The program is executed with the input string "Apple apple red apple green apple". Until the program is terminated, the statement marked by " $\leftarrow \alpha$ " will be executed F time(s), and the statement marked by " $\leftarrow \beta$ " will be executed G time(s).

Answer group

- | | | |
|------|------|------|
| a) 1 | b) 2 | c) 3 |
| d) 4 | e) 5 | f) 6 |

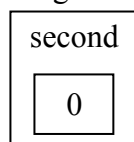
- Q8.** Read the following description of Java programs and the programs themselves, and then answer Subquestions 1 and 2.

[Program Description]

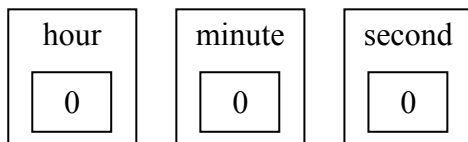
The class `CascadedCounter` represents a multi-purpose cascaded modular counter, which can be used to model a counting mechanism. These mechanisms are used in a digital clock, taxi-meter, up/down counter, and many others. For example, in case of using the model to count time, the hour, minute, and second will be represented by individual modular counting objects, which will be linked / cascaded so that when counting up, after 59 seconds, the seconds becomes 0 and the minute becomes 1. The methods to manipulate the model are invoked / called from the main method. All values that are set are automatically changed to modulo LIMIT. For example, the second is limited to 59 (modulo 60). Setting/increasing it to 60 will automatically make it zero.

The following is an example of the process of a multi-purpose cascaded modular counter:

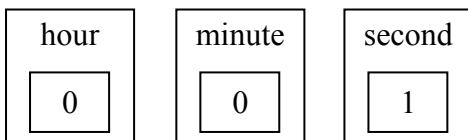
- (1) Create to count second and setting its limit to 59 (modulo 60),



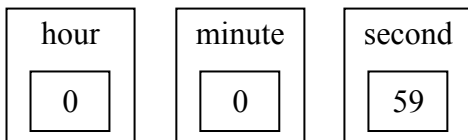
- (2) To count hour and minute, after attaching two more counters to second,



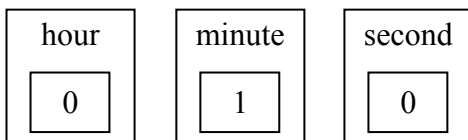
- (3) After counting up once,



- (4) After counting up 58 times,



- (5) After counting up once,



The class `CascadedCounter` implements the following functions:

- Increase/decrease the value of counter to count up/down
- Attach additional modules to counter, e.g. attach “minute” in front of “second”
- Set the counter to certain value
- Reset the counter

Subquestion 1

From the answer groups below, select the correct answer to be inserted in each blank in [Program 1].

At first, the class `Counter` that represents single counter and `Tester1` that is the tester class of class `Counter` are implemented.

The class `Counter` throws `IllegalArgumentException()` if a negative integer is given to variable `value` or non-positive integer is given to variable `limit`.

The following statements are outputted when class `Tester1` is executed.

```
second(s): 0
second(s): 59
second(s): 30
second(s): 59
second(s): 0
second(s): 1
```

[Program 1]

```
public class Counter {
    private int value, limit;
    private String label;

    Counter(int value, int limit, String label) {
        if (limit <= 0) {
            throw new IllegalArgumentException();
        }
        this.limit = limit;
        setValue(value);
        this.label = label;
    }

    public int getValue() {
        return value;
    }
}
```

```

public void setValue(int value) {
    if (value < 0){
        throw new IllegalArgumentException();
    }
    this.value = A;
}

public void resetValue() {
    setValue(0);
}

public int getLimit() {
    return limit;
}

public void countUp() {
    setValue(getValue() + 1);
}

public void countDown() {
    if (B) {
        setValue(getValue() - 1);
    } else {
        setValue(getLimit() - 1);
    }
}

public String toString() {
    return label + ": " + getValue();
}
}

```

[Program 2]

```
public class Tester1 {  
    public static void main(String[] args) {  
        Counter c = new Counter(0, 60, "second(s)");  
        System.out.println(c);  
        c.countDown();  
        System.out.println(c);  
        c.setValue(30);  
        System.out.println(c);  
        c.setValue(59);  
        System.out.println(c);  
        c.countUp();  
        System.out.println(c);  
        c.countUp();  
        System.out.println(c);  
    }  
}
```

Answer group for A

- | | |
|------------------|------------------|
| a) limit - value | b) value % limit |
| c) value - limit | d) value / limit |

Answer group for B

- | | |
|----------------------------|-----------------------------|
| a) getValue() < getLimit() | b) getValue() <= getLimit() |
| c) getValue() > 0 | d) getValue() >= 0 |

Subquestion 2

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

In addition to the classes from Subquestion 1, there are two classes in the following programs: CascadedCounter and Tester2.

The following statements are outputted when the class Tester2 is executed.

```
hour(s): 0   minute(s): 0   second(s): 59  
hour(s): 0   minute(s): 1   second(s): 0  
hou(s): 0   minute(s): 59   second(s): 59  
hour(s): 1   minute(s): 0   second(s): 0  
hour(s): 0   minute(s): 59   second(s): 59
```

[Program 3]

```
public class CascadedCounter extends Counter {
    private Counter leftSide;

    CascadedCounter(int value, int limit, String label) {
        super(value, limit, label);
    }

    public void countUp() {
        if (  &&  ) {
            leftSide.countUp();
        }
        super.countUp();
    }

    public void countDown() {
        if (  &&  ) {
            leftSide.countDown();
        }
        super.countDown();
    }

    public void resetValue() {
        super.resetValue();
        if (  ) {
            leftSide.resetValue();
        }
    }

    public void connectLeft(Counter leftSide) {
        this.leftSide = leftSide;
    }

    public void disconnectLeft() {
        this.leftSide = null;
    }

    public String toString() {
        String str = "";
        if (  ) {
            str += leftSide;
        }
        str += "    " + (  );
        return str;
    }
}
```


[Program 4]

```
public class Tester2 {  
    public static void main(String[] args) {  
        CascadedCounter sec = new CascadedCounter(59, 60, "second(s)");  
        CascadedCounter min = new CascadedCounter(0, 60, "minute(s)");  
        CascadedCounter hr = new CascadedCounter(0, 24, "hour(s)");  
        sec.connectLeft(min);  
        min.connectLeft(hr);  
        System.out.println(sec);  
        sec.countUp();  
        System.out.println(sec);  
        min.setValue(59);  
        sec.setValue(59);  
        System.out.println(sec);  
        sec.countUp();  
        System.out.println(sec);  
        sec.countDown();  
        System.out.println(sec);  
    }  
}
```

Answer group for C and E

- a) `(getValue() + 1) == getLimit()`
- b) `getValue() < getLimit()`
- c) `getValue() <= getLimit()`
- d) `getValue() == 0`
- e) `getValue() == getLimit()`
- f) `getValue() > 0`

Answer group for D

- | | |
|--|--|
| a) <code>leftSide != null</code> | b) <code>leftSide == null</code> |
| c) <code>super.leftSide != null</code> | d) <code>super.leftSide == null</code> |

Answer group for F

- | | |
|----------------------------|-------------------------------------|
| a) <code>leftSide</code> | b) <code>leftSide.toString()</code> |
| c) <code>super</code> | d) <code>super.toString()</code> |
| e) <code>this</code> | f) <code>this.toString()</code> |
| g) <code>toString()</code> | |