October 2013

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.

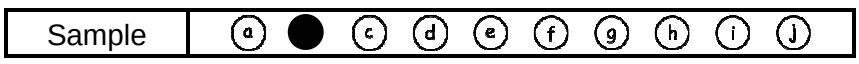
   (4) **Answers**
   Mark your answers as shown in the following sample question.

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

   Answer group
   a) September b) October c) November d) December

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

   [Sample Answer]



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	• variable ← expression	Assigns the value of the expression to the variable.
	• procedure(argument, ...)	Calls the procedure and passes / receives the 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 the process 1 is executed. If it is `false`, then the 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. 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	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.

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

Q1. Read the following description concerning instruction execution, and then answer Subquestion.

In a normal CPU, an instruction execution goes through 4 stages. The instruction is first (1) fetched; then (2) decoded and data is fetched; then (3) executed; and finally (4) the data is written-back. Only after the completion of an instruction, the next instruction can start.
To improve performance, a technique called instruction pipeline is used in computer design to increase the number of instructions executed per unit time. In a 4-stage pipeline, the next instruction starts as the previous instruction starts its next stage.
The diagram below illustrates the execution of consecutive instructions. Assuming that each stage takes 1 clock cycle. At any given point in time (clock cycle), 4 consecutive instructions are being processed. At clock cycle 5, the instructions I1 and I2 are already completed, I3 through I6 are being executed in different stages, and I7 and I8 are waiting for their turn. The diagram shows that it takes 11 clock cycles to complete 8 instructions. Without the pipeline, it will take 8×4 clock cycles = 32 clock cycles.

Instruction	Status	Clock cycle										
		0	1	2	3	4	5	6	7	8	9	10
I1	Completed	F	D	X	W							
I2	Completed		F	D	X	W						
I3	Stage 4 - Write-back			F	D	X	W					
I4	Stage 3 - Execute				F	D	X	W				
I5	Stage 2 - Decode					F	D	X	W			
I6	Stage 1 - Fetch						F	D	X	W		
I7	Waiting							F	D	X	W	
I8	Waiting								F	D	X	W

Note F: Fetch, D: Decode, X: Execute, W: Write-back.

In such a CPU that uses the 4-stage pipeline, when all instructions are executed sequentially without any delays, CPI (cycles per instruction) of 1 will be attained.

[Data hazard problem]

When instructions are sequenced such that the succeeding instruction requires the result of the previous instruction, a data hazard problem arises. In the following example, the first instruction adds the contents of R2 and R3 and stores the result into R1, and the second instruction multiplies the contents of R1 by itself and stores the result into R4.

```
R1 ← R2 + R3      /* Note: "Rn" denotes "Register n" */
R4 ← R1 × R1
```

In such a case, the preceding instruction must complete before the succeeding instruction can start. This makes the pipelining useless in terms of performance improvement.

To remedy the situation, instructions are analyzed before entering the pipeline. When a result (like R1 in the above example) is needed for the succeeding instruction entering the pipeline, the following steps are done.

(1) The result is retained in a register-in-wait after the execute stage of the preceding instruction.

(2) The decode stage of the succeeding instruction starts after the execute stage of the preceding instruction. It makes use of the register-in-wait instead of the result of the write-back.

In the following diagram, a "Bubble" (noted as B) of no operation exists in clock cycle 2.

Instruction		0	1	2	3	4	5	6
I1	R1 ← R2 + R3	F	D	X	W			
I2	R4 ← R1 × R1		F	B	D	X	W	
	Next instruction				F	D	X	W

[Control hazard problem]

In a conditional branch instruction, the succeeding instruction is not known until the instruction completes its execution. This creates a control hazard problem. In the following diagram, it will not be known if I2 or T1 should follow the instruction I1 until I1 completes its execution. In this case, T1 is the next instruction as R1 is 0.

Instruction		0	1	2	3	4	5	6	7	8
I1	GOTO T1 IF R1=0	F	D	X	W					
I2	R3 ← R2 ÷ R1									
…	…									
T1 (case 1)	R3 ← 0		B	B	B	F	D	X	W	
T1 (case 2)	R3 ← 0		F	D	X	W				

To improve performance, BTB (branch target buffer) is introduced. Once a branch instruction is executed, its target instruction is stored into BTB. When the branch instruction jumps to the target instruction not found in BTB, the fetch stage of the target instruction starts after the write-back stage of the branch instruction (see "T1 (case 1)" in the above diagram). When the branch instruction jumps to the target instruction found in BTB, the fetch stage of the target instruction starts after the fetch stage of the branch instruction (see "T1 (case 2)" in the above diagram).

Subquestion

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

(1) The set of instructions shown below is executed. Assuming that BTB is empty at first, fetch stage of the first instruction I1 starts at clock cycle 0, and each stage of each instruction takes 1 clock cycle.

I1	R7 ← 2
I2	R1 ← R2 + R3
I3	R4 ← R3 × R3
I4	R5 ← R2 ÷ R1
I5	R0 ← R5
I6	R2 ← R4 × R5
I7	R7 ← R7 − 1
I8	GOTO I2 IF R7>0
I9	R1 ← 0

When the first 8 instructions, I1 through I8, are executed, there will be [A] bubbles inserted in the stages of the instructions.

The 9th instruction is I2, because the 8th instruction I8 causes the branch. At this point, the fetch stage of the 9th instruction I2 will be at clock cycle [B].

When the sequence of instructions I6 and I7 are exchanged, the number of bubbles will decrease by [C] for each iteration.

(2) When on the average, the instructions have a 25% data hazard, with 20% of the instructions are executed as conditional branches, and BTB gets a 75% hit-ratio, the effective CPI is [D]. Assuming that each stage of each instruction takes 1 clock cycle.

Answer group for A and C

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

Answer group for B

a) 9 b) 10 c) 11 d) 12 e) 13

Answer group for D

a) 1.3 b) 1.4 c) 1.45 d) 1.7 e) 1.85

Q2. Read the following description concerning a flight reservation database, and then answer Subquestions 1 and 2.

An airline company decides to develop a database to manage the flight reservation information.

Figure 1 shows the relational schema of the flight reservation information. The relation "Reservation" keeps the reservation information of each flight of the airline company. Table 1 shows the meanings and constraints of the attributes. Table 2 shows the major functional dependences between the attributes.

```
Flight ( FlightID, FlightNumber, FlightDate, DepartureTime, ArrivalTime,
         OriginAirport, DestinationAirport, AirplaneType )
Passenger ( PassengerNumber, PassengerName, Address, ContactInformation,
         EmailAddress, MembershipNumber )
Reservation ( ReservationNumber, PassengerNumber, FlightID, SeatType,
         ReservationDate, DeadlineDate, Fulfillment )
SeatInAirplane ( AirplaneType, SeatType, Quantity )
```

Figure 1 Relational schema of the flight reservation information

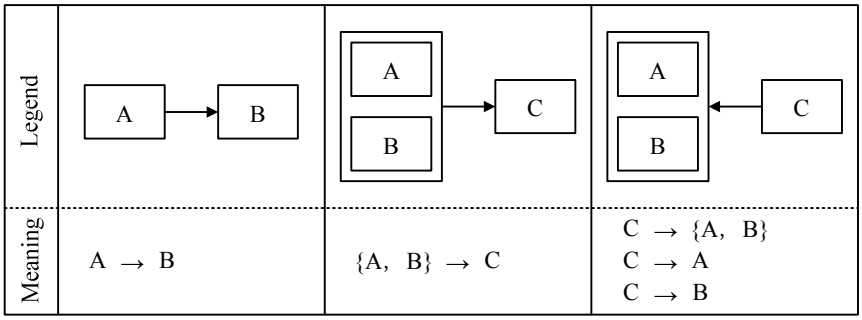
Table 1 Meanings and constraints of the Attributes (abbreviated in part)

Attribute	Meaning and Constraints
FlightID	The generated sequence number that uniquely identifies the flight.
FlightNumber	The alpha-numeric code, such as FE123, that uniquely identifies the flight within a day. It is possible to have several flights with the same flight number on different flight dates.
FlightDate	The date that the flight departs from the origin airport.
DepartureTime	The time that the flight departs from the origin airport.
ArrivalTime	The time that the flight arrives at the destination airport.
OriginAirport	The airport which the flight departs from.
DestinationAirport	The airport which the flight arrives at.
AirplaneType	The type of the airplane of the flight.
PassengerNumber	The number that uniquely identifies the passenger.
PassengerName	The name of the passenger.
Address	The address of the passenger.
ContactInformation	The contact information of the passenger.
EmailAddress	The e-mail address of the passenger.
MembershipNumber	The membership number of the passenger.
ReservationNumber	The number that uniquely identifies the reservation.

ReservationDate	The date that the reservation has been performed.
DeadlineDate	The deadline date that the passenger has to issue the flight ticket.
Fulfillment	The flag indicating whether the passenger has already issued the flight ticket or not.
SeatType	First class (F), Business class (B), or Economy class (E).
Quantity	The number of seats by airplane type and seat type.

Table 2 Major functional dependencies between the attributes

1.	FlightID → FlightNumber, FlightDate, DepartureTime, ArrivalTime, OriginAirport, DestinationAirport, AirplaneType
2.	{ FlightNumber, FlightDate } → FlightID, DepartureTime, AirplaneType
3.	{ FlightNumber, DepartureTime } → ArrivalTime
4.	FlightNumber → OriginAirport, DestinationAirport
5.	PassengerNumber → PassengerName, Address, ContactInformation, EmailAddress, MembershipNumber
6.	PassengerName → Address, EmailAddress, MembershipNumber
7.	MembershipNumber → PassengerName, Address, EmailAddress
8.	EmailAddress → PassengerName, Address, MembershipNumber
9.	ReservationNumber → PassengerNumber, FlightID, SeatType, ReservationDate, DeadlineDate, Fulfillment
10.	{ PassengerNumber, FlightID } → ReservationNumber, SeatType, ReservationDate, DeadlineDate, Fulfillment
11.	{ AirplaneType, SeatType } → Quantity



Subquestion 1

From the answer group below, select the correct relation that is in third normal form for the relation “Flight”. Here, the underlined attributes are the primary keys of the relation.

Answer group

a) (FlightID, FlightNumber, FlightDate),
(DepartureTime, OriginAirport, ArrivalTime, DestinationAirport, AirplaneType)

b) (FlightID, FlightNumber, FlightDate, DepartureTime),
(FlightNumber, FlightDate, AirplaneType),
(FlightNumber, DepartureTime, ArrivalTime),
(FlightNumber, OriginAirport, DestinationAirport)

c) (FlightID, FlightNumber, FlightDate, DepartureTime, AirplaneType),
(FlightNumber, DepartureTime, ArrivalTime),
(FlightNumber, OriginAirport, DestinationAirport)

d) (FlightID, FlightNumber, FlightDate, DepartureTime, AirplaneType),
(FlightNumber, DepartureTime, ArrivalTime, OriginAirport, DestinationAirport)

e) (FlightID, FlightNumber, FlightDate, DepartureTime, ArrivalTime,
OriginAirport, DestinationAirport, AirplaneType)

Subquestion 2

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

An SQL statement is created to check which flights from Bangkok to Tokyo on October 27, 2013 still have some unoccupied business class seats (seat type “B”). The following list is a sample output that shows the flight numbers and the number of available seats.

Flight numbers	Number of available seats
FE641	10
FE643	2
FE677	0

```
SELECT [ A ]
FROM [ B ]
WHERE OriginAirport = 'Bangkok' AND
      DestinationAirport = 'Tokyo' AND
      Flight.FlightDate = '2013-10-27' AND
      SeatType = 'B' AND
      Flight.AirplaneType = SeatInAirPlane.AirplaneType AND
      [ C ]
GROUP BY FlightNumber, Quantity
```

Answer group for A

a) `FlightID, COUNT(*)`
b) `FlightID, MAX(Quantity)`
c) `FlightID, MAX(Quantity)-COUNT(*)`
d) `FlightNumber, COUNT(*)`
e) `FlightNumber, MAX(Quantity)`
f) `FlightNumber, MAX(Quantity)-COUNT(*)`

Answer group for B

a) `Flight, Passenger, Reservation`
b) `Flight, Reservation`
c) `Flight, Reservation, SeatInAirplane`
d) `Flight, SeatInAirplane`
e) `Reservation`

Answer group for C

a) `Flight.FlightID = Reservation.FlightID`
b) `Flight.FlightID = Reservation.FlightID AND`
   `Reservation.PassengerNumber = Passenger.PassengerNumber`
c) `Flight.FlightID = Reservation.FlightID AND`
   `Reservation.PassengerNumber = Passenger.PassengerNumber AND`
   `Reservation.SeatType = SeatInAirplane.SeatType`
d) `Flight.FlightID = Reservation.FlightID AND`
   `Reservation.SeatType = SeatInAirplane.SeatType`
e) `Reservation.SeatType = SeatInAirplane.SeatType`

Q3. Read the following description concerning a communication network, and then answer Subquestions 1 through 3.

Organization A uses 192.168.23.0/24 IP addresses and has several segments of networks that are connected to each other using router R1.
Figure 1 shows the network configuration in organization A, and Table 1 shows the routing table of router R1.

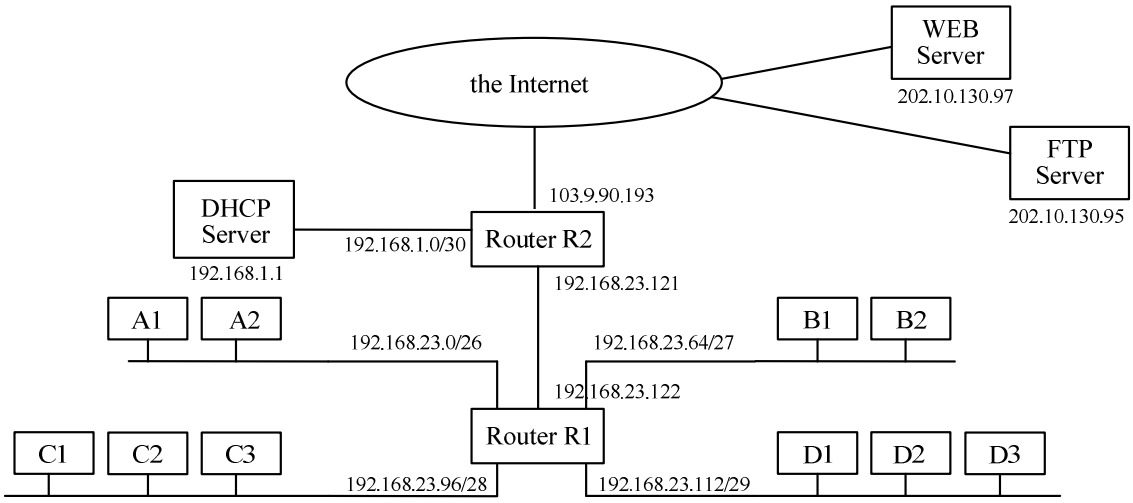


Figure 1 Network configuration in organization A

Table 1 Routing table of router R1

Mask	Network address	Next hop	Interface
/26	192.168.23.0	-	m0
/27	192.168.23.64	-	m2
/28	192.168.23.96	-	m3
/29	192.168.23.112	-	m4
/30	192.168.23.120	-	m5
/30	192.168.1.1	192.168.23.121	m5
/any	any	192.168.23.121	m5

Subquestion 1

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

(1) Host A2 sends the DHCPDISCOVER message to the DHCP server for requesting the addresses. The router R1 received this request message via interface m0. Then, the router R1 forwards it via interface [A].

(2) The organization A is planning to create a new subnet with 30 customers. In order to build this subnet, organization A's network administrator assigns network address [B] with the subnet mask [C] for this subnet.

Answer group for A

a) m0 b) m2 c) m3 d) m4 e) m5

Answer group for B

a) 192.168.23.30
b) 192.168.23.32
c) 192.168.23.124
d) 192.168.23.128

Answer group for C

a) 255.255.255.128
b) 255.255.255.192
c) 255.255.255.224
d) 255.255.255.240
e) 255.255.255.248
f) 255.255.255.252

Subquestion 2

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

It was found that the host C1 could not access to other network.
The host C1 successfully sent a ping message to the local network. However, the host C1 unsuccessfully sent a ping message to the 192.168.23.112/29 network.
IP configuration of the host C1 is shown in Figure 2.
From these pieces of information, it was concluded that [D].

```
C:\>ipconfig
IP configuration
Ethernet adapter Local Area Connection:
IPv4 Address:    192.168.23.98
Subnet Mask:     255.255.255.240
Default Gateway: 192.168.23.96
```

Figure 2 IP configuration of host C1

Answer group

a) the default gateway is incorrectly configured
b) the IP address on the host C1 is incorrectly configured
c) the local LAN cable is damaged
d) the subnet mask on the host C1 is incorrectly configured

Subquestion 3

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

A user of the host A2 uses a Web browser to connect to the WEB server. When the user sends a request to the WEB server, the host A2 creates the following IP packet.

Destination IP address:	202.10.130.97
Source IP Address:	192.168.23.2
Destination Port:	TCP port 80
Source Port:	TCP port 5000

When the packet arrives at the router R2, the NAT (network address translation) function of the router R2 translates the packet to the following, and sends it to the WEB server.

Destination IP address:	(*not shown*)
Source IP Address:	(*not shown*)
Destination Port:	TCP port 80
Source Port:	TCP port 1025

When the NAT function receives its response from the WEB server, the received IP packet will contain the following.

Destination IP address:	[E]
Source IP Address:	(*not shown*)
Destination Port:	[F]
Source Port:	TCP port 80

Answer group for E

a) 103.9.90.193
b) 192.168.1.1
c) 192.168.23.2
d) 202.10.130.97

Answer group for F

a) TCP port 0
b) TCP port 80
c) TCP port 1025
d) TCP port 5000

Q4. Read the following description concerning a digital signature, and then answer Subquestions 1 and 2.

In public-key cryptography, generation and verification processes for a digital signature are as follows.

Encryption and signing processes by a sender:

(1) Generate the message digest of the plain message. The message digest is generated using the message digest function.
(2) Encrypt the message digest. The resulting encrypted message digest is the digital signature.
(3) Encrypt the plain message.
(4) Transmit the encrypted message and the digital signature to the receiver.

Decryption and verification processes by a receiver:

(1) Decrypt the received encrypted message. The resulting decrypted message is the received plain message.
(2) Generate the message digest of the received plain message using the same message digest method used by the sender.
(3) Decrypt the received digital signature. The resulting decrypted message digest is the message digest generated by the sender.
(4) Compare both message digests generated in steps (2) and (3) above. If they are equal, the digital signature is verified. Otherwise, it is not verified.

Subquestion 1

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

Figure 1 shows the scheme of the above processes, and Table 1 shows the notations used in Figure 1.

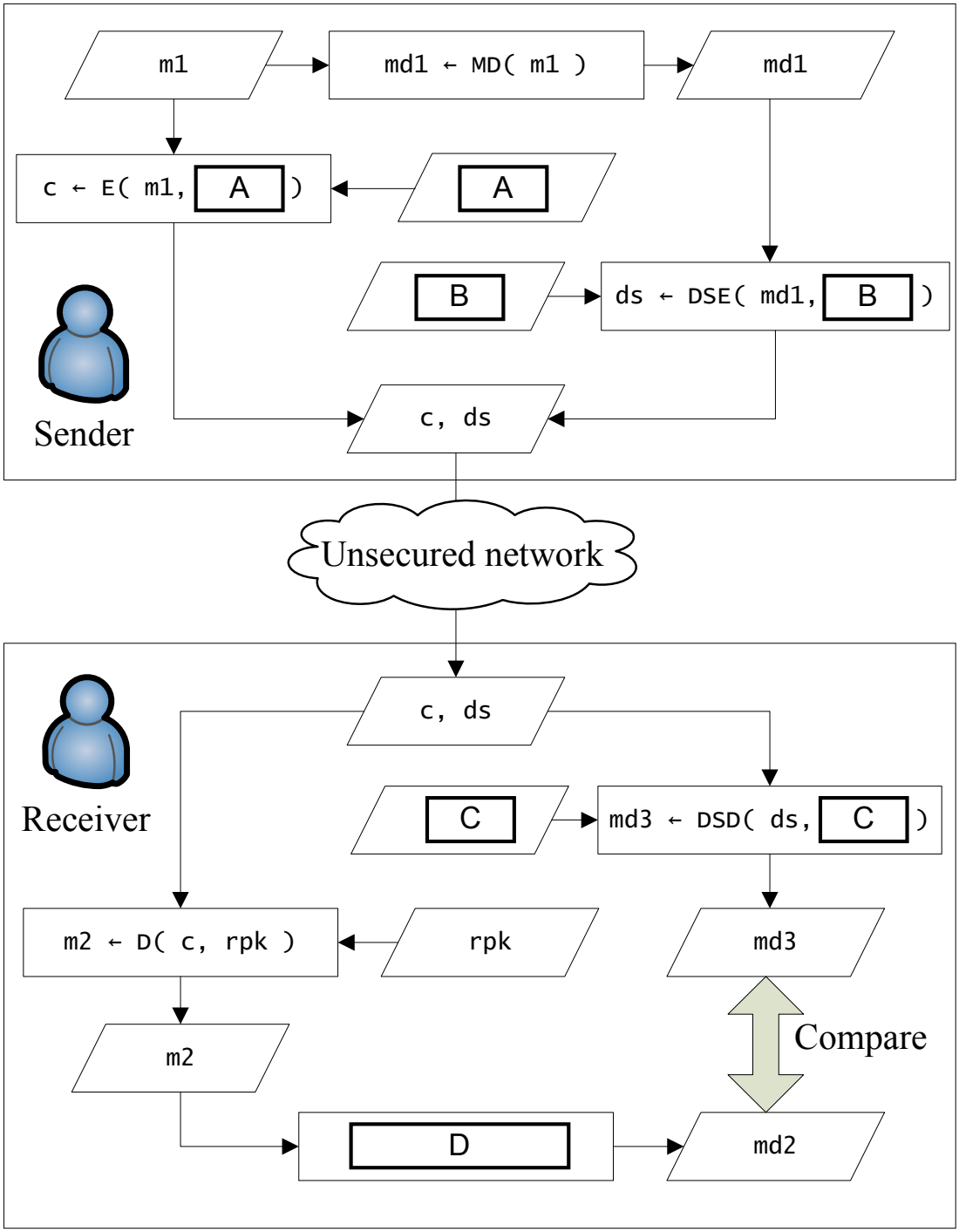


Figure 1 Scheme of the above processes

Table 1 Notations used in Figure 1.

Notation	Description
Functions	
DSE	Digital Signature Encryption function
DSD	Digital Signature Decryption function
E	Public-key Encryption function
D	Public-key Decryption function
MD	Message Digest function
Variables	
c	Cipher message (encrypted message)
ds	Digital signature
m1, m2	Plain messages
md1, md2, md3	Message digests
rpk	Receiver's Private Key
ruk	Receiver's Public Key
spk	Sender's Private Key
suk	Sender's Public Key

Answer group for A through C

a) rpk
b) ruk
c) spk
d) suk

Answer group for D

a) md2 ← DSE(m2, suk)
b) md2 ← E(m2, rpk)
c) md2 ← E(m2, suk)
d) md2 ← MD(m2)

Subquestion 2

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

Alice sent the message to Bob according to the above processes. In Bob's verification process, the signature could not be verified.
In this case, the security problem, such as [E] or [F], might have occurred.

Answer group

a) Arice's public key was stolen
b) Bob's public key was stolen
c) collision occurred in function MD
d) the message was not sent from Arice
e) third party changed the message
f) third party monitored the message

Q5. Read the following description concerning an address generation program, and then answer Subquestions 1 through 3.

Company *Z*, a mail-order company, manages its customers by using a customer master file based on a customer number assigned to each customer.

Company *Z* is planning to send sweepstakes (lottery) tickets and invitation tickets to an event to customers according to their spending during the sales promotion campaign period (hereinafter the period) from November 10, 2013 to December 20, 2013. Therefore, the company decides to develop an address generation program that selects customers from a sales slip file for sending sweepstakes tickets and invitation tickets to, and generates an address file.

The program is required to have the following functions.

(1) The number of sweepstakes tickets sent to each customer is to be the same as the number of campaign products the customer purchases during the period. The product codes of the campaign products covered in this sales promotion campaign are A001 through A199.

(2) One invitation ticket is sent to a customer whose total purchase amount for all the products the customer purchases during the period is 50,000 yen or more.

(3) For every customer to whom sweepstakes tickets or an invitation ticket are sent, one record is created in the address file. Even if both sweepstakes ticket and invitation ticket are sent, one record is created per customer in the address file.

(4) The total number of sweepstakes tickets, the total number of invitation tickets, and the number of records in the address file are printed in a summary table.

The customer master file, sales slip file, and address file are sequential files. Figure 1 shows the record format of each file.

Customer master file

Customer number	Customer address	Customer name

Sales slip file

Purchase date	Customer number	Product code	Purchased quantity	Purchase amount

Address file

Customer number	Customer address	Customer name	Number of sweepstakes tickets	Number of invitation tickets	Total purchase amount

Figure 1 Record format of each file

The customer master file is sorted in ascending order of customer number. A work file is created by sorting the sales slip file in ascending order of customer number, and is then passed to the program.

Figure 2 shows the input and output related to this program, Figure 3 shows the flow of the

program, and Table 1 shows the processing of the major modules. Furthermore, Table 2 shows the test data for the work file used in the testing of this program, and Figure 4 shows the summary table generated using this test data.

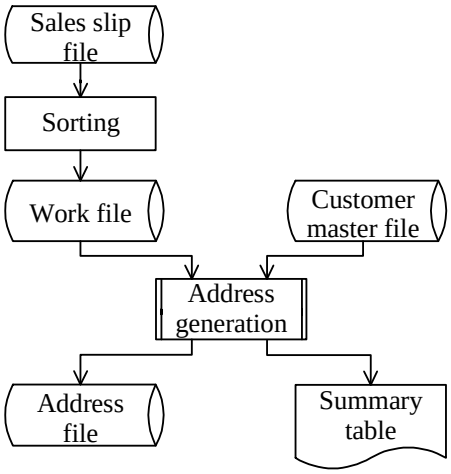
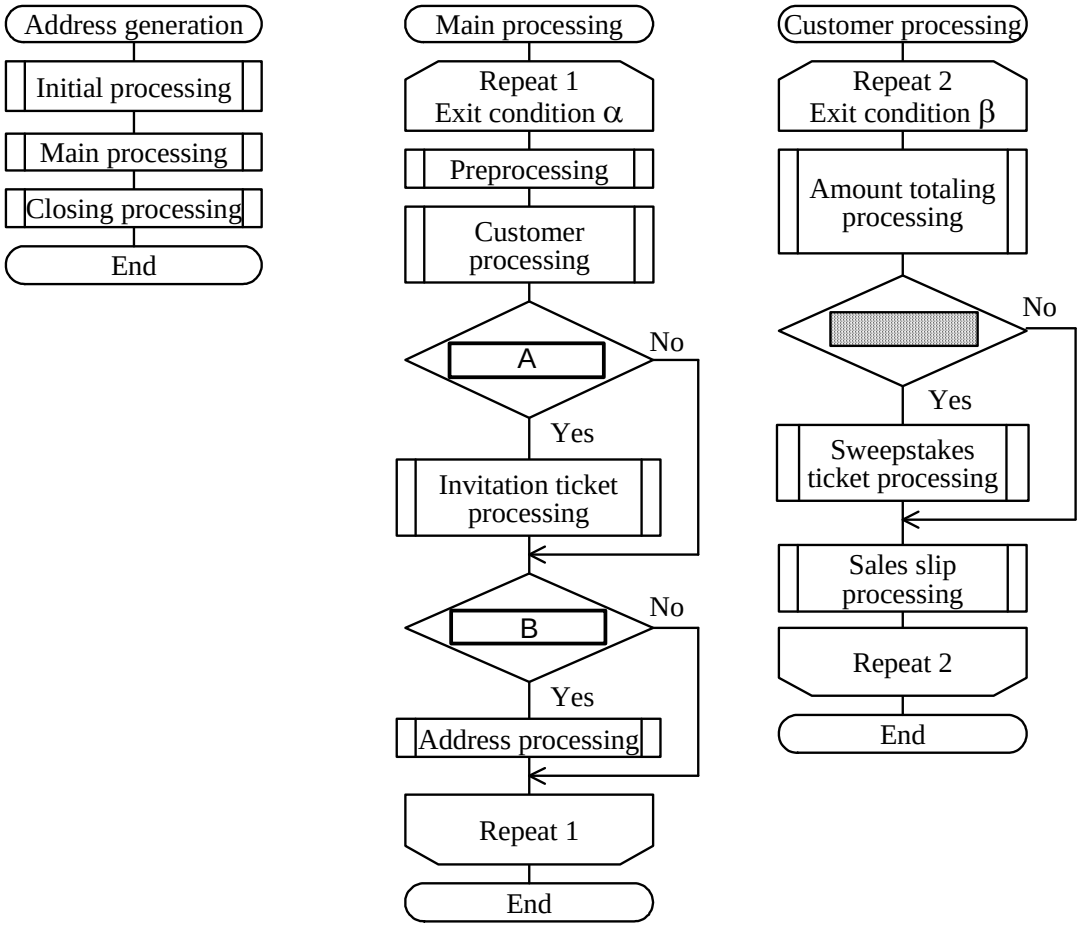


Figure 2 Input and output related to the address generation program



Note: The shaded portion is not displayed.

Figure 3 Flow of the address generation program

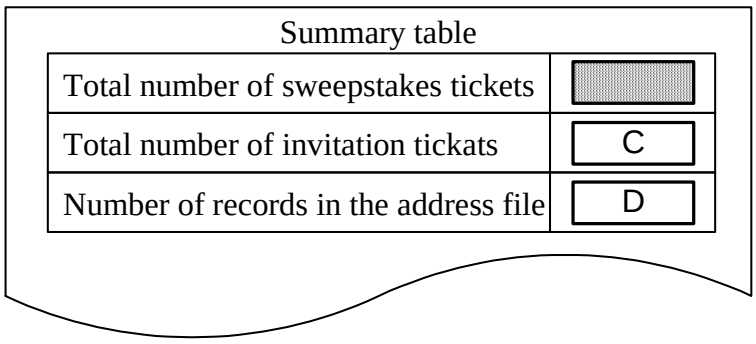
Table 1 Processing of major modules

Module	Description of processing
Initial processing	Opens each file. Total number of sweepstakes tickets ← 0, Total number of invitation tickets ← 0, Number of records in the address file ← 0 Reads the work file (Skips the records outside the period).
Preprocessing	Number of sweepstakes tickets← 0, Number of invitation tickets ← 0, Total purchase amount ← 0, W customer number ← Customer number in a record in the work file
Amount totaling processing	Calculates the total purchase amount.
Sweepstakes ticket processing	Number of sweepstakes tickets ← [ ]
Sales slip processing	Reads the work file (Skips the records outside the period).
Invitation ticket processing	Performs the calculation for determining the number of invitation tickets.
Address processing	Reads the customer master file (Skips if the value of the customer number is not equal to the value of W customer number). Edits a record of the address file, then writes it in the address file. Performs the calculation for determining the total number of sweepstakes tickets. Performs the calculation for determining the total number of invitation tickets. Performs the calculation for determining the number of records in the address file.
Closing processing	Writes the summary table. Closes each file.

Note: Number of sweepstakes tickets, number of invitation tickets, total purchase amount, W customer number, total number of sweepstakes tickets, total number of invitation tickets, and number of records in the address file are the work areas.

Table 2 Test data for the work file

Purchase date			Customer number	Product code	Purchased quantity	Purchase amount (Yen)
Year	Month	Day				
2013	11	8	001	A008	3	60000
2013	11	25	001	A180	8	44000
2013	12	3	002	B150	5	25000
2013	12	18	002	B125	5	20000
2013	11	25	005	A007	1	50000
2013	10	29	010	B001	1	45000
2013	11	25	010	B150	12	60000
2013	12	3	081	A201	2	52000
2013	12	12	081	B002	2	60000
2013	12	20	081	A006	2	20000
2014	1	16	081	A200	5	48000
2013	11	10	258	B003	1	2500
2013	11	18	386	A182	1	25000
2013	12	18	386	A198	2	20000
2013	12	21	386	A108	1	50000



Note: The shaded portion is not displayed.

Figure 4 Summary table generated using the test data shown in Table 2

Subquestion 1

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

Answer group for A

a) A001 ≤ Product code in a record in the work file ≤ A199
b) A001 ≤ Product code in a record in the work file ≤ A199 or
   Total purchase amount ≥ 50000
c) Number of invitation tickets > 0
d) Purchased quantity in a record in the work file > 0
e) Total purchase amount ≥ 50000

Answer group for B

a) A001 ≤ Product code in a record in the work file ≤ A199
b) A001 ≤ Product code in a record in the work file ≤ A199 and
   Purchase amount in a record in the work file ≥ 50000
c) A001 ≤ Product code in a record in the work file ≤ A199 or
   Purchase amount in a record in the work file ≥ 50000
d) Number of sweepstakes tickets > 0
e) Number of sweepstakes tickets > 0 and number of invitation tickets > 0
f) Number of sweepstakes tickets > 0 or number of invitation tickets > 0
g) Purchase amount in a record in the work file ≥ 50000
h) Purchased quantity in a record in the work file > 0

Subquestion 2

From the answer group below, select the correct answer to be inserted in the blank [ ] in Table 1. Here, [ ] in the answer group is a Gauss symbol, and [x] indicates the largest integer value not exceeding x.

Answer group

a) 1
b) Purchased quantity in a record in the work file
c) [Purchase amount in a record in the work file / 50000]
d) Number of sweepstakes tickets + 1
e) Number of sweepstakes tickets + Purchased quantity in a record in the work file
f) Number of sweepstakes tickets +
   [Purchase amount in a record in the work file / 50000]

Subquestion 3

From the answer groups below, select the correct answer to be inserted in each blank [ ] in Figure 4.

Answer group for C

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

Answer group for D

a) 3	b) 4	c) 5	d) 7	e) 9
f) 10	g) 11	h) 12	i) 13	j) 14

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

[Program Description]

Figure 1 shows an example of a weighted graph, and Figure 2 shows an example of a spanning tree of the graph shown Figure 1. A spanning tree is a sub-graph of a given graph. It is a tree, it has no cycles, and it connects all the vertices (nodes) together.

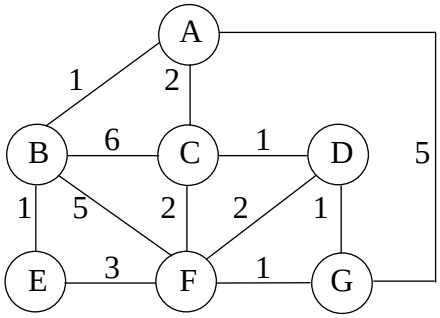


Figure 1 Example of a weighted graph

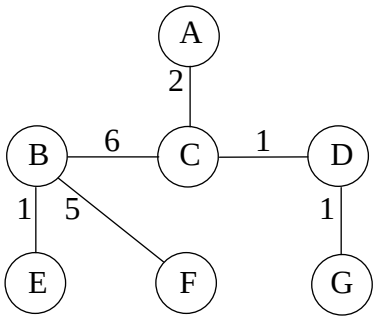


Figure 2 Example of a spanning tree

A minimum spanning tree (hereinafter MST) of a given graph is the spanning tree having the minimum weight for that graph. Kruskal's algorithm constructs an MST of a given graph by adding edges to the spanning tree one-by-one.

Initially, each vertex is in its own tree in a forest, and all the edges stay unresolved. In the case of Figure 1, there are 7 trees composed of only one vertex, and 12 unresolved edges.

Then, the edges are selected and processed one by one, as follows:

(1) Select an edge (v_1, v_2), which connects the vertices v_1 and v_2, with the minimum weight among the unresolved edges.

(2) If the vertices v_1 and v_2 belong to two different trees, then add this edge to the forest, and combine these two trees into one tree.

(3) Otherwise, simply discard this edge.

When all the edges are processed, an MST of a given graph is obtained.

The function `Kruskal(V, E, F)` has 3 parameters.

`V` is a set whose elements are vertices. When `Kruskal` is called, `V` contains all the vertices in the given graph.

`E` is a set whose elements are edges. When `Kruskal` is called, `E` contains all the edges in the given graph. Each edge has 3 values: v_1, v_2, and w. Here, v_1 and v_2 are the vertices at both ends of the edge, and w is the weight of the edge. In the program, specific edge is expressed as `e(v1,v2,w)`.

`F` is a set whose elements are trees. During the process, some trees may have only one vertex, while other trees may have two or more vertices connected by one or more edges.

The program uses the following functions:

`AddTree(`v`)`: Adds a new element to `F`. The element to be added is a tree that has only one vertex v.

`FindTree(`v`)`: Scans `F`, and returns an ID of the element that has the vertex v in its tree. Assuming that a unique ID is assigned to each element in `F`.

`Connect(`v_1`, `v_2`, `w`)`: Connect the vertices v_1 and v_2 with the edge `e(v1,v2,w)`. As a result, two elements (trees) are combined into one element (tree) in `F`.

In the program, an empty (null) set is expressed as `{}`.

When the program finishes, `F` contains an element of an MST.

[Program]

```
○ Function Kruskal(V, E, F)
• F ← {}                                  /* make the set F empty */
■ V ≠ {}                                  /* while V is not empty */
  • v ← next element in V
  • AddTree(v)
  • remove v from V
■
• sort E in the ascending order of w
■ A
  • e(v1,v2,w) ← next element in E
  ▲ B
  ▼ • Connect(v1,v2,w)
• remove e(v1,v2,w) from E
■
• Return
```

Subquestion 1

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

Answer group for A

a) `E = {}`
b) `E ≠ {}`
c) `F = {}`
d) `F ≠ {}`

Answer group for B

a) `(FindTree(v1) = FindTree(v2)) and (E ≠ {})`
b) `(FindTree(v1) ≠ FindTree(v2)) and (E = {})`
c) `FindTree(v1) = FindTree(v2)`
d) `FindTree(v1) ≠ FindTree(v2)`

Subquestion 2

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

The function `Kruskal(V, E, F)` is executed for the weighted graph shown in Figure 3.

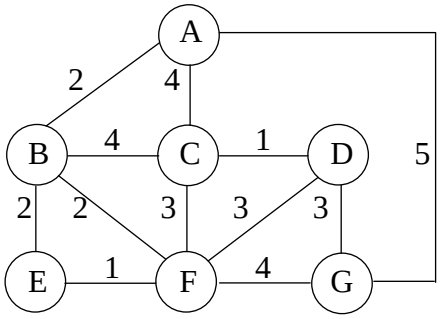


Figure 3 Weighted graph

Assuming that, after the statement "• sort `E` in the ascending order of `w`" is executed, the sets `E` and `F` have the following elements:

Set `E` : {e(C, D, 1), e(E, F, 1), e(A, B, 2), e(B, E, 2), e(B, F, 2), e(C, F, 3), e(D, F, 3), e(D, G, 3), e(A, C, 4), e(B, C, 4), e(F, G, 4), e(A, G, 5)}

Set `F` : {A, B, C, D, E, F, G}

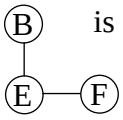
Also, assuming that the elements in `E` are picked up one by one from the top.

Because the set `E` have 12 elements, the statement "• `e(v1,v2,w)` ← next element in `E`" is executed 12 times. Table 1 shows the status of the set `F` in the iteration process.

Table 1 Status of the set F in the iteration process

Iteration	Edge used	• Connect(v1,v2,w)	Resulted elements in set F
1	e(C, D, 1)	Executed	{ A, B, C-D, E, F, G }
2	e(E, F, 1)	Executed	{ A, B, C-D, E-F, G }
3	e(A, B, 2)	Executed	{ A-B, C-D, E-F, G }
…	…	…	…
[shaded]	C	Executed	{ [shaded], [shaded] }
…	…	…	…
[shaded]	D	Executed	{ E }
…	…	…	…
10	e(B, C, 4)	Not executed	(Not changed)
11	e(F, G, 4)	Not executed	(Not changed)
12	e(A, G, 5)	Not executed	(Not changed)

Note: (1) Shaded parts are not shown.

(2) F is the set whose elements are trees. In Table 1, a tree is shown in a simplified form. For example, a tree 

is shown as "B-E-F".

As in Table 1, every time the statement " • Connect(v1,v2,w) " is executed, the number of elements in the set F decreases by 1. When the edge [C] is picked up from the set E and the function Connect is executed, the number of elements in the set F becomes 2. After that, when the edge [D] is picked up from the set E and the function Connect is executed, the number of elements in the set F becomes 1.

At this point, an MST [E] is obtained in the set F. The total weight value of the 6 edges in the MST is [F]. This is the minimum total weight value among any spanning trees of the given graph.

Answer group for C and D

a) e(A, C, 4) b) e(B, E, 2) c) e(B, F, 2)
d) e(C, F, 3) e) e(D, F, 3) f) e(D, G, 3),

Answer group for E

a) A-B-C-D-G-E-F
b) A-B-E-F-C-D-G
c) A-B-C-D-G
E-F
d) A-B-E-F-G
C-D

Answer group for F

a) 11 b) 12 c) 13 d) 14 e) 15

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.

A program was created for drawing one or more playing cards from a shuffled deck of playing cards.
A set of playing cards consists of 52 cards. There are 4 suits: Clubs, Diamonds, Hearts, and Spades. Each suit has 13 ranks: 2 to 9, 10, Jack, Queen, King, and Ace.

[Program Description]
The program prompts the user to enter the number of cards to draw. Then, the program selects the specified number of cards randomly, and displays them on the screen.

(1) The cards are numbered from 0 to 51, and contained in the array `cards[]`. The first 13 values are used for Clubs, the second for Diamonds, the third for Hearts, and the fourth for Spades suit. The program uses this internal numbering system throughout, and only converts to card ranks and suits at the output stage by `display()` subprogram.
(2) The variable `nDraw` contains the number of cards to draw. The value of `nDraw` ranges from 1 to 52.
(3) `lShuffle()` subprogram shuffles a deck of cards. In the `for` loop, the process which sets aside one card and shuffles the remaining cards is repeated. The same card will never be drawn twice or more.
(4) `draw()` subprogram determines which card will be drawn, by using an array `drawn[]` that corresponds to `cards[]`. At the beginning, the elements of the array `drawn[]` are set to 0. Each time a card is drawn, the corresponding element of `drawn[]` is set to 1.
(5) When the cards are displayed on the screen, each card is expressed as 2-character code *rs*. Here, *r* is one of the codes 2 to 9, T, J, Q, K, and A, that expresses the rank 2 to 9, 10, Jack, Queen, King, and Ace respectively. *s* is one of the codes C, D, H, and S, that expresses the suit Club, Diamond, Heart, and Spade. For example, 2C expresses 2 of Club, and AD expresses Ace of Diamond.
(6) Figure 1 shows a sample output of the program.

```
Number of Cards to Draw: 4
2C
AD
5H
2S
```

Figure 1 Sample output of the program

[Program]

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define CARDS 52
#define SUITS 4
void initialize(int []);
void lShuffle(int []);
void draw(int [], int);
void display(int [], int []);

int main(void) {
    int card[CARDS], drawn[CARDS] = {0}, nDraw;

    srand(time(NULL));
    initialize(card);
    lShuffle(card);

    printf("Number of Cards to Draw: ");
    scanf("%d", &nDraw);
    draw(drawn, nDraw);
    display(card, drawn);
    return 0;
}

/* Initialize cards */
void initialize(int card[]) {
    int i;

    for (i = 0; i < CARDS; i++)
        card[i] = i;
}

/* Shuffle a deck of cards in linear time */
void lShuffle(int card[]) {
    int i, cardsLeft, j, temp;

    cardsLeft = CARDS;
    for (i = 0; i < CARDS; i++) {
        j = [      A      ];
        temp = card[i];
        card[i] = card[j];
        card[j] = temp;
        cardsLeft--;
    }
}
```

```
/* Draw n cards from a deck */
void draw(int drawn[], int nDraw) {
    int i, j, notDrawn, card, skipped, keeplooking;

    notDrawn = CARDS;
    for (i = 0; i < nDraw; i++) {
        card = rand() % notDrawn;
        skipped = 0;
        keeplooking = 1;
        for (j = 0; j < CARDS && keeplooking == 1; j++) {
            if (drawn[j] == 0 && [ B ]) {
                drawn[j] = 1;
                keeplooking = 0;
            } else if (drawn[j] == 0)
                [ C ];
        }
        notDrawn--;
    }
}

/* Display cards on standard output */
void display(int card[], int drawn[]) {
    char suit, face;
    int rank, i;
    char FACES[20] = "23456789TJQKA";

    for (i = 0; i < CARDS; i++) {
        if (drawn[i] == 1) {
            switch ([ D ]) {
            case 0:
                suit = 'S';
                break;
            case 1:
                suit = 'H';
                break;
            case 2:
                suit = 'D';
                break;
            case 3:
                suit = 'C';
                break;
            }
            rank = card[i] % (CARDS/SUITS);
            face = [ E ];
            printf("%c%c\n", face, suit);
        }
    }
}
```

Subquestion 1

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

Answer group for A

a) `(i + rand()) % cardsLeft`
b) `(j + rand()) % cardsLeft`
c) `i + rand() % cardsLeft`
d) `j + rand() % cardsLeft`

Answer group for B

a) `skipped != card`
b) `skipped != card - 1`
c) `skipped == card`
d) `skipped == card - 1`

Answer group for C

a) `card = 0`
b) `card++`
c) `skipped++`
d) `skipped--`

Answer group for D

a) `(card[i] + 1) / (CARDS / SUITS)`
b) `(card[i] + 1) / (CARDS / SUITS) + 1`
c) `(card[i] + 1) / (CARDS / SUITS) – 1`
d) `card[i] / (CARDS / SUITS)`
e) `card[i] / (CARDS / SUITS) + 1`
f) `card[i] / (CARDS / SUITS) – 1`

Answer group for E

a) `FACES[rank]`
b) `FACES[rank + 1]`
c) `FACES[rank - 1]`
d) `FACES[rank / SUITS]`

Subquestion 2

From the answer group below, select the correct rank-and-suit of cards to be displayed, when 5 cards are drawn randomly and their values are 16, 37, 28, 30, and 40.

Answer group

a) 4D, QH, 3H, 5H, and 2S
b) 4H, QD, 3D, 5D, and 2C
c) 5D, KH, 4H, 6H, and 3S
d) 5H, KD, 4D, 6D, and 3C

Q8. Read the following description of Java programs and the programs themselves, and then answer Subquestion.

[Program Description]

In many interesting situations, separate, concurrently running threads do share data and must consider the state and activities of other threads. In one such set of programming situations, called producer-consumer scenarios, the producer generates a stream of data that the consumer uses.

For example, imagine an application in which one thread (the producer) writes data to a file while a second thread (the consumer) reads data from the same file. Or, on a PC, the producer thread places mouse events in an event queue while the consumer thread reads the events from the same queue. Both use concurrent threads that share a common resource: the first shares a file; the second shares an event queue. Because the threads share a common resource, they must be synchronized.

If the two threads make no arrangements for synchronization, some problems might arise.

One problem might arise when the producer is quicker than the consumer, as the producer puts two values before the consumer has a chance to get the first one. In this situation, the consumer misses a value, as shown in the following example.

```
  ...
Consumer got: 3
Producer put: 4     ← Consumer missed 4
Producer put: 5
Consumer got: 5
  ...
```

Another problem might arise when the consumer is quicker than the producer. In this situation, the consumer gets the same value twice, as shown in the following example.

```
  ...
Producer put: 4
Consumer got: 4
Consumer got: 4     ← Consumer got 4 twice
Producer put: 5
  ...
```

Either way, the result is wrong. The consumer should get each value placed by the producer exactly once. A problem like this is called a race condition. A race condition is a situation in which two or more threads or processes are reading or writing some shared data, and the final result depends on the timing of how the threads are scheduled. Race conditions can lead to unpredictable results and subtle program bugs. Race conditions in the producer-consumer example are prevented by having storage of a new integer in the `HoldInteger` by the producer synchronized with retrieval of an integer from the `HoldInteger` by the consumer.

The activities of the producer and the consumer must be synchronized in two ways.
First, the two threads must not simultaneously access the `HoldInteger`. A thread can prevent this from happening by locking an object. When an object is locked by one thread and another thread tries to call a synchronized method on the same object, the second thread will be blocked until the object is unlocked.
Second, the two threads must do some simple coordination. That is, the producer must have a way to indicate to the consumer that the value is ready, and the consumer must have a way to indicate that the value has been retrieved. The `Object` class provides a collection of methods — `wait`, `notify`, and `notifyAll` — to help threads wait for a condition and to notify other threads when that condition changes.
When the programs are executed, the following list will be printed out.

```
Producer put: 1
Consumer got: 1
Producer put: 2
Consumer got: 2
Producer put: 3
Consumer got: 3
Producer put: 4
Consumer got: 4
Producer put: 5
Consumer got: 5
Producer put: 6
Consumer got: 6
Producer put: 7
Consumer got: 7
Producer put: 8
Consumer got: 8
Producer put: 9
Consumer got: 9
Producer put: 10
Producer finished producing values
Terminating Producer
Consumer got: 10
Consumer retrieved values totaling: 55
Terminating Consumer
```

[Program 1]

```
public class ProducerConsumerTest {
    public static void main(String args[]) {
        HoldInteger sharedObject = new HoldInteger();
        ProduceInteger producer = new ProduceInteger(sharedObject);
        ConsumeInteger consumer = new ConsumeInteger(sharedObject);
```

A

```
    }
}
```

[Program 2]

```
public class ProduceInteger extends Thread {
    private HoldInteger sharedObject;

    public ProduceInteger(HoldInteger shared) {
        super("Producer");
        sharedObject = shared;
    }
    public void run() {
        for (int count = 1; count <= 10; count++) {
            try {
                Thread.sleep((int)(Math.random() * 1000));
            } catch(      B      exception) {
                System.err.println(exception.toString());
            }
            sharedObject.setSharedInt(count);
        }
        System.err.println(getName() + " finished producing values"
                           + "\nTerminating " + getName());
    }
}
```

[Program 3]

```
public class ConsumeInteger extends Thread {
    private HoldInteger sharedObject;

    public ConsumeInteger(HoldInteger shared) {
        super("Consumer");
        sharedObject = shared;
    }
    public void run() {
        int value, sum = 0;
        do {
            try {
                Thread.sleep((int)(Math.random() * 1000));
            } catch(      B      exception) {
                System.err.println(exception.toString());
            }
            value = sharedObject.getSharedInt(); // Accesses shared data
            sum += value;
        } while (value != 10);
        System.err.println(getName() + " retrieved values totaling: "
                           + sum + "\nTerminating " + getName());
    }
}
```

[Program 4]

```
public class HoldInteger {
    private int sharedInt = -1;
    private boolean writeable = true;

    [ C ] void setSharedInt(int value) {
        while (!writeable) {
            try {
                [ D ];
            } catch ([ B ] exception) {
                exception.printStackTrace();
            }
        }
        System.err.println(Thread.currentThread().getName()
                           + " put: " + value);
        sharedInt = value;
        writeable = false;
        [ E ];
    }
    [ C ] int  getSharedInt() {
        while (writeable) {
            try {
                [ D ];
            } catch ([ B ] exception) {
                exception.printStackTrace();
            }
        }
        writeable = true;
        [ E ];
        System.err.println(Thread.currentThread().getName()
                           + " got: " + sharedInt);
        return sharedInt;
    }
}
```

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)
```
consumer.run();
producer.run();
```

b)
```
consumer.start();
producer.start();
```

c)
```
(new Thread(consumer)).run();
(new Thread(producer)).run();
```

d)
```
(new Thread(consumer)).start();
(new Thread(producer)).start();
```

e)
```
(new Thread(new ConsumerInteger())).start();
(new Thread(new ProducerInteger())).start();
```

Answer group for B

a) `IllegalMonitorStateException`
b) `IllegalStateException`
c) `IllegalThreadStateException`
d) `InterruptedException`

Answer group for C

a) `private`
b) `private static`
c) `private synchronized`
d) `public`
e) `public static`
f) `public synchronized`

Answer group for D and E

a) `interrupt()`
b) `join()`
c) `notify()`
d) `sleep()`
e) `stop()`
f) `wait()`
g) `yield()`