April, 2007

Fundamental IT Engineer Examination (Afternoon)

Questions must be answered in accordance with the following:

Question Nos.	Q1 – Q5	Q6 – Q9	Q10 – Q13
Question Selection	Compulsory	Select 1 of 4	Select 1 of 4
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 test 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** (**Q6-Q9** and **Q10-Q13**)

   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 next 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 Reply]

SQ	a	b	c	d
1	◯	⬤	◯	◯

3. "Assembly Language specifications" are provided as a reference at the end of this booklet.

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

Inquiries about the exam questions will not be answered.

[Explanation of the Pseudo-Code Description Format]

Pseudo-Language Syntax	Description
⌐	A continuous area where declarations and processes are described.
○	Declares names, types, etc. of procedures, variables, etc.
▪ Variable ← Expression	Assigns the value of an Expression to a Variable.
▲ Conditional expression 　▪ Process 1 　▪ Process 2 ▼	A selection process. If the Conditional expression is True, then Process 1 is executed. If it is False, then Process 2 is executed.
■ Conditional expression 　▪ Process ■	A repetition process with the termination condition at the top. The Process is executed while the Conditional expression is True.

[Operator]

Operation	Operator	Priority
Unary operation	+ − not	High
Multiplication and division operation	* /	
Addition and subtraction operation	+ −	
Relational operation	> < >= <= =	
Logical product	and	
Logical sum Exclusive logical sum	or xor	Low

[Logic type constant]

```
true  false
```

Q1.   Read the following description of lists, and then answer Subquestions 1, 2 and 3.

The structure of the lists is as shown in Figure 1.

ROOT

| → 0091 | → 0093 | → 0095 | → ⋯ | → 00B0 | 0000 |
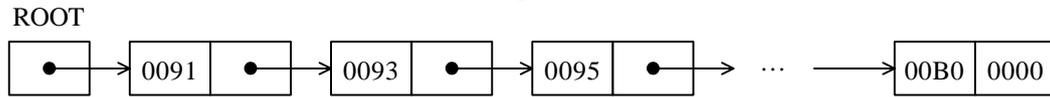
Fig. 1   List Structure

1)   ROOT means the start of the list.

2)   Elements of the list consist of two consecutive words.   The first word stores the value, and the second word stores a pointer to the next element.

3)   Each element of the list is linked to the next in ascending order of the value, and all values are unique.   In the second word of the last element, 0000 is stored as a pointer.

4)   A list with the structure shown in Fig. 1 is stored at addresses 00FF to 0117 in the main memory as shown in Figure 2.   Address 00FF is ROOT.

5)   One word consists of 16 bits, and addresses are assigned in word units.

Address	Contents	Address	Contents	Address	Contents	Address	Contents
⋮	⋮	0106	00A0	010E	00B0	0116	0099
00FF	0100	0107	010C	010F	0000	0117	0110
0100	0091	0108	00A9	0110	009B	0118	00A7
0101	010A	0109	0112	0111	0102	0119	0000
0102	009F	010A	0093	0112	00AB	011A	009C
0103	0106	010B	0104	0113	010E	011B	011C
0104	0095	010C		0114	00A2	011C	00A5
0105	0116	010D	0114	0115	0108	011D	0118

Fig. 2   State of the Main Memory

Subquestion 1

From the answer group below, select the correct answer for the contents of address 010C.

Answer group:

    a) 0099   b) 00A1   c) 00A3   d) 00A4
    e) 00A5

Subquestion 2

From the answer group below, select the correct answers to be inserted into the blanks ☐ in the following description.

In order to delete the elements at addresses 0110 and 0111, the contents of address ☐A need to be changed to ☐B.

Answer group:

    a) 0101   b) 0102   c) 0103   d) 0104
    e) 0105   f) 0113   g) 0114   h) 0115
    i) 0116   j) 0117

Subquestion 3

From the answer group below, select the correct answers to be inserted into the blanks ☐ in the following description.

In order to merge the sublist consisting of the three elements stored at addresses 0118 through 011D and the original list (before deleting the elements in Subquestion 2), the contents of address ☐C need to be changed to 011A, the contents of address ☐D to 0102, the contents of address ☐E to 011C, and the contents of address ☐F to 0108 respectively.

Answer group:

    a) 0109   b) 010B   c) 010D   d) 010F
    e) 0111   f) 0113   g) 0115   h) 0117
    i) 0119   j) 011B

Q2. Read the following description of a relational database, and then answer Subquestions 1, 2 and 3.

A certain database consists of the following employee and department table. An employee works for a department and may or may not have a manager.

emp (employee) table

empno	ename	job	mgr	hiredate	sal	comm	deptno

dept (department) table

deptno	dname	loc

Subquestion 1

A display of the highest earner of each job is required, wherein the employees of each job category would be compared to the highest salary within the category.

ename	job	Highest-Salary

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

```
SELECT ename, job,  A
FROM   emp
WHERE  sal IN (SELECT  B  FROM emp  GROUP BY job)
ORDER BY  C  descending;
```

Answer group:

a) Highest-Salary
b) MAX (sal)
c) MAX (sal) Highest-Salary
d) sal
e) sal Highest-Salary
f) salary

Subquestion 2

Provided that the empno is also used to denote mgr for another employee, what would be the appropriate select statement to determine the number of distinct managers without listing them.

No of managers

Answer group:

```
a)  select count (distinct (mgr)) "No of managers" from emp;

b)  select count (mgr (distinct)) "No of managers" from emp;

c)  select distinct (count (mgr)) "No of managers" from emp;

d)  select distinct (mgr) "No of managers" from emp;

e)  select mgr (distinct) "No of managers" from emp;
```

Subquestion 3

There is another table called `salgrade`, which has the salary grading and the lowest and highest salary within the grade.

`salgrade` table with data

grade	lowsal	highsal
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

A listing of employees including employee name, job, salary and salary grade is done to determine all employees in grade 2.

From the answer group below, select the correct answers to be inserted into the blanks ☐ in the following SQL statement.

```
select e.ename, e.job, e.sal, s.grade
from  [ D ] ,
where [ E ]  and s.grade = 2;
```

Answer group for D:

```
a)  e.emp, s.grade          b)  e.emp, s.salgrade

c)  emp e, grade s          d)  emp e, salgrade s

e)  emp, grade              f)  emp, salgrade
```

Answer group for E:

```
a)  e.sal <= s.highsal

b)  e.sal <= s.lowsal

c)  e.sal >= s.highsal

d)  e.sal >= s.lowsal

e)  e.sal between s.lowsal and s.highsal

f)  e.sal between s.lowsal or s.highsal
```

Q3. Read the following description of DNS (Domain Name System), and then answer the Subquestions 1 and 2.

In a certain enterprise, each computer in the in-house network is controlled by using a domain name.   The configuration of the network in this enterprise is shown below.
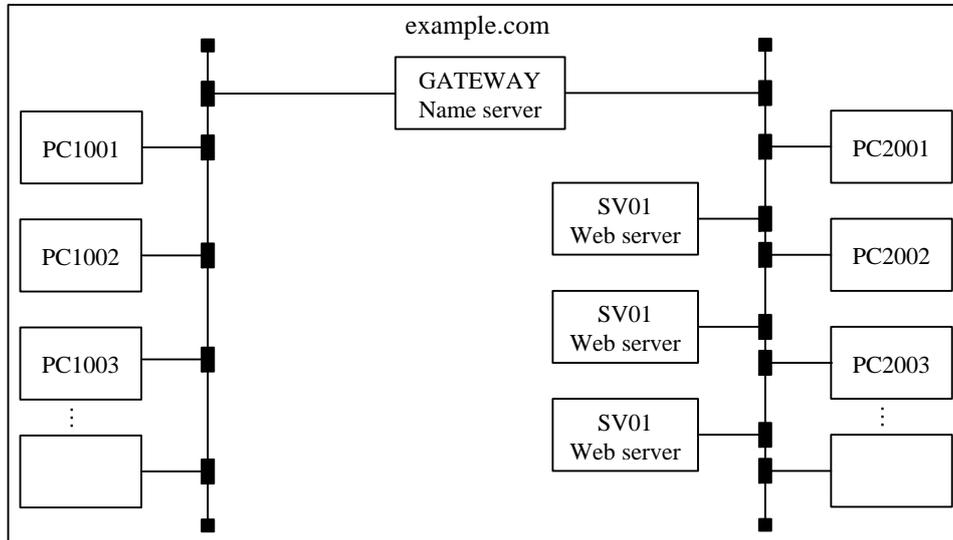
Correspondence between the domain name and the IP address of each computer is as shown in the table below.

Table   Correspondence between domain names and IP addresses

Domain name	IP address
PC1001.example.com	172.16.0.1
PC1002.example.com	172.16.0.2
PC1003.example.com	172.16.0.3
⋮	⋮
PC2001.example.com	172.31.0.1
PC2002.example.com	172.31.0.2
PC2003.example.com	172.31.0.3
⋮	⋮

Domain name	IP address
GATEWAY.example.com	172.16.0.101 172.31.0.101
SV01.example.com	172.31.0.91
SV01.example.com	172.31.0.92
SV01.example.com	172.31.0.93

Subquestion 1

From the answer group below, select the correct answers to be inserted into the blanks ☐ in the following description.

A name server has a definition file for searching for IP addresses from domain names.   In this file, a name server is defined as follows:

<Domain name to be defined >.   IN NS   <Domain name of name server>.

Moreover, correspondence between a domain name and an IP address is defined as follows:

<Domain name>.   IN A   <IP address>

In the case where GATEWAY.example.com is a name server, some of the statements in the definition file that this name server has are shown below.

```
example.com.        IN NS   [  A  ] .example.com.

localhost.example.com.  IN A  127.0.0.1
PC1001.example.com.     IN A  172.16.0.1
PC1002.example.com.     IN A  172.16.0.2
PC1003.example.com.     IN A  172.16.0.3
        ⋮
GATEWAY.example.com. IN A  172.16.0.101
GATEWAY.example.com. IN A  [  B  ]

PC2001.example.com.  IN A  172.31.0.1
PC2002.example.com.  IN A  172.31.0.2
PC2003.example.com.  IN A  172.31.0.3
        ⋮
GATEWAY-1.example.com.  IN A  172.16.0.101
GATEWAY-2.example.com.  IN A  172.31.0.101

SV01.example.com.   IN A  172.31.0.91
SV01.example.com.   IN A  172.31.0.92
SV01.example.com.   IN A  172.31.0.93
```

Answer group:

    a) 127.0.0.1   b) 172.31.0.91   c) 172.31.0.101

    d) GATEWAY   e) localhost   f) SV01

Subquestion 2

From the answer group below, select the correct answers to be inserted into the blanks ☐ in the following description.

In a DNS, a client which queries with a name server is called a resolver.
If IP addresses belonging to different subnetworks are defined for the same domain name, then the name server returns, on a priority basis, the IP addresses that belong to the same subnetwork to which the resolver belongs.   This mechanism is applied in the case of the definition of ☐ C ☐.example.com.
On the other hand, if different IP addresses belonging to the same subnetwork are defined for the same domain name, then the name server returns those IP addresses cyclically in the order of definition.   This mechanism is called round robin, and is used to distribute accesses to the server.   This mechanism is applied in the case of the definition of ☐ D ☐.example.com.

Answer group:

   a) GATEWAY-1   b) GATEWAY-2   c) GATEWAY

   d) localhost   e) SV01

Q4.　Read the following description of a program and the program itself, and then answer Subquestions 1, 2 and 3.

[Program Description]

This is a subprogram called `TowerHanoiGame` that move sequence of n disks in peg `FROM` to peg `TO` with the same rule.
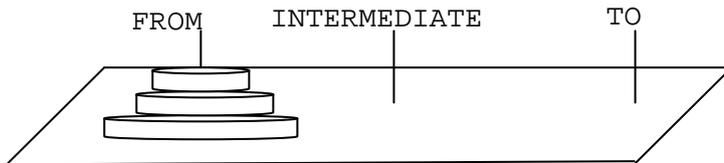
1) Procedures are as follows:

The Towers of Hanoi game is an example of problem whose solution demands recursion. The game consists of a board with three vertical pegs labeled `FROM`, `INTERMEDIATE`, and `TO`, and sequence of n disks with holes in their centers. The radius of the disks are in an arithmetic progression (e.g., 5cm, 6cm, 7cm…) and are mounted on the peg `FROM`. The rule is that no disk may be above a smaller disk on the same peg. The objective of the game is to move all the disks from peg `FROM` to peg `TO`, one disk at a time, without violating the rule.

The general solution to the Tower of Hanoi game is naturally recursive:

　Part 1: move the smaller n-1 disks from peg `FROM` to peg `INTERMEDIATE`

　Part 2: move the remaining disk from peg `FROM` to peg `TO`

　Part 3: move the smaller n-1 disks from peg `INTERMEDIATE` to peg `TO`

The first and third steps are recursive: apply the complete solution to n-1 disks. In the case n = 1, move this disk from peg `FROM` to peg `TO`.

2) Argument specification for the subprograms are given in the following tables.

Table 1　`TowerHanoiGame` **arguments**

Variable name	Input/ Output	Meaning
`N`	Input	Number of disks
`FROM`	Input	The name of peg where `N` disks are mounted before running this subprogram
`INTERMEDIATE`	Input	The name of peg where disks could be moved to or from during running this subprogram
`TO`	Input	The name of peg where `N` disks are finally mounted after running this subprogram

Table 2 `Println` **argument**

Variable	Input/Output	Meaning
`Str`	Input	Character string displayed in the screen in one line.

[Program]
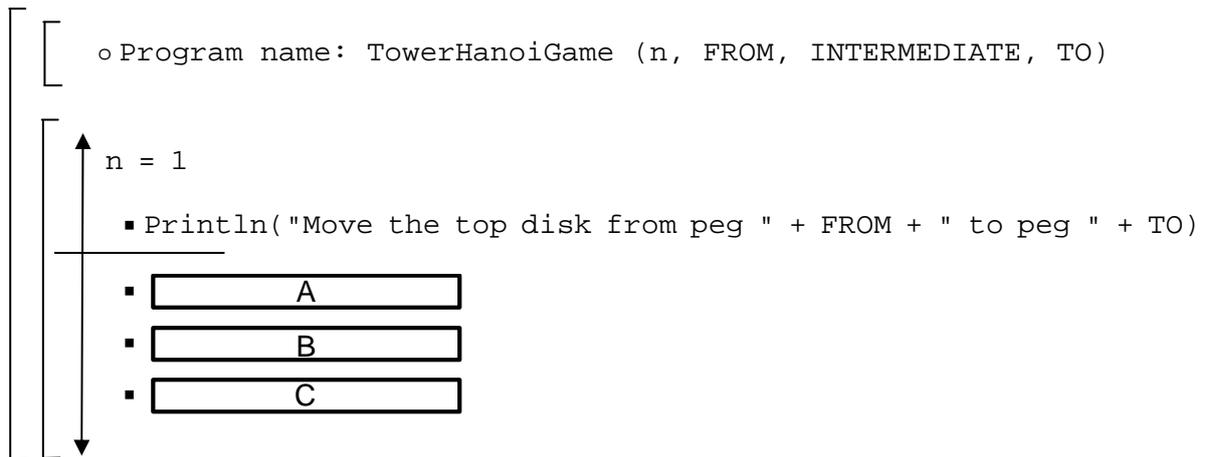
```
o Program name: TowerHanoiGame (n, FROM, INTERMEDIATE, TO)

  n = 1

    ▪ Println("Move the top disk from peg " + FROM + " to peg " + TO)
    ▪ ┌──────────────────────────┐
      │             A            │
      └──────────────────────────┘
    ▪ ┌──────────────────────────┐
      │             B            │
      └──────────────────────────┘
    ▪ ┌──────────────────────────┐
      │             C            │
      └──────────────────────────┘
```

Subquestion 1

From the answer group below, select the correct answers to be inserted into the blanks ☐ in the above program.

Answer group:

    a) `TowerHanoiGame (1, FROM, INTERMEDIATE, TO)`

    b) `TowerHanoiGame (1, FROM, TO, INTERMEDIATE)`

    c) `TowerHanoiGame (n-1, FROM, INTERMEDIATE, TO)`

    d) `TowerHanoiGame (n-1, FROM, TO, INTERMEDIATE)`

    e) `TowerHanoiGame (n-1, INTERMEDIATE, FROM, TO)`

    f) `TowerHanoiGame (n-1, INTERMEDIATE, TO, FROM)`

    g) `TowerHanoiGame (n-1, TO, INTERMEDIATE, FROM)`

    h) `TowerHanoiGame (n-1, TO, FROM, INTERMEDIATE)`

Subquestion 2

From the answer group below, select the correct answers to be inserted into the blanks
| D | through | I |.

Note: Answers could be the same.

The solution for 3 disks is produced by the call
```
TowerHanoiGame (3, "FROM ", "INTERMEDIATE ", "TO");
```
The output is as below:
```
Move the top disk from peg FROM to peg TO
```

D
E
F
G
H
I
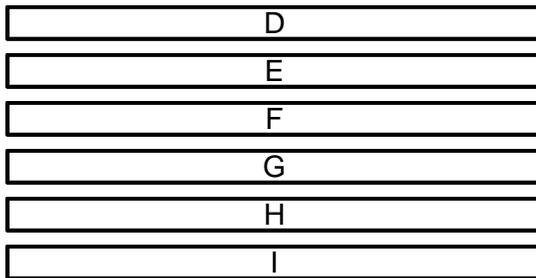
Answer group:

   a) `Move the top disk from peg FROM to peg INTERMEDIATE`

   b) `Move the top disk from peg FROM to peg TO`

   c) `Move the top disk from peg INTERMEDIATE to peg FROM`

   d) `Move the top disk from peg INTERMEDIATE to peg TO`

   e) `Move the top disk from peg TO to peg FROM`

   f) `Move the top disk from peg TO to peg INTERMEDIATE`

Subquestion 3

From the answer group below, select the correct answers to be inserted into the blanks
| J | through | L |.

The Towers of Hanoi game moves the disks 7 times for 3 disks (n=3).
This program moves the disks | J | times for 4 disks, | K | times for 5 disks, and | L | times for n disks.

Answer group:

   a) 11   b) 15   c) 31

   d) 32   e) $2n + 1$ f) $2^n - 1$

Q5. Read the following description of a program design, and then answer Subquestion.

[Program Description]

One program is developed to classify the students into two classes X and Y at a training center. Class X contains the more competent students and Class Y contains less competent students who didn't pass the tests that organized in the center. The center has the appropriate training plans for each class to get best result.

1) There are 80 students registered for the training.

2) Each test consists of 80 questions. The questions are classified by the topics. The number of questions and minimum number of correct answers (pass criteria) in each topic for each test is shown in the Table1.

Table1 Topic_Table (Number of questions for each topic and pass test criteria)

	0	1	← Column
Row ↓	Number of questions	Minimum number of correct answers	Topic's comment
0	6	4	Topic 1: Computer Science fundamentals
1	40	32	Topic 2: Computer system
2	10	6	Topic 3: System development and operation
3	6	4	Topic 4: Network Technology
4	6	4	Topic 5: Database Technology
5	6	4	Topic 6: Security and Standardization
6	6	4	Topic 7: Computerization and management

3) 80 questions are ordered by the topics as shown below:
   Questions 1 to 6 belong to Topic 1
   Questions 7 to 46 belong to Topic 2
   Questions 47 to 56 belong to Topic 3
   Questions 57 to 62 belong to Topic 4
   Questions 63 to 68 belong to Topic 5
   Questions 69 to 74 belong to Topic 6
   Questions 75 to 80 belong to Topic 7

4) StudentID and test results of 80 questions (1: correct, 0: incorrect) for all 80 students are stored in the **Master_Table** as shown in the Table 2.

Table2   Master_Table

| Row	StudentID 0	1	2	3	4	5	6	…	j	..	77	78	79	80	pass 81	Column
0	1	1	1	0	1	0	0	..		.	1	1	1	1		
1	2	1	1	1	0	0	1	..		..	1	1	0	1		
2	3	1	1	1	1	1	1	..		..	1	1	1	1		
3	4	1	1	1	0	1	0	..		..	0	1	1	1		
	..															
i	i															
79	80	1	1	0	1	0	0	..	..	..	1	1	1	1		

5) An element **Master_Table [i,j]**, where $(0 <= i <= 79, 1 <= j <= 80)$, corresponds to student "i" and the answer "j".

6) For clarity, an example is given by extraction of the last row from the Table2. After a test, the test answers made by the student whose StudentID is 80 are looked like this:

Row 79	StudentID 0	1	2	3	4	5	6	…	j	…	77	78	79	80	pass 81
	80	1	1	0	1	0	0	…		…	1	1	1	1	

Question 1, 2, 4, … : correct answer
Question 3, 5, 6, … : incorrect answer

7) After the test, the students that pass the test (is selected by the criteria, described in the Table1) are marked with pass = 1 in column **pass**, that means "correct answers" / "number of questions in the Topic" is greater than or equal to 4/6, 32/40, 6/10, 4/6, 4/6, 4/6, 4/6, corresponding to Topic 1 to Topic 7, respectively.

8) After the test, all the records of the passed students are copied to a table **class_X**, and rest of the records are copied to a table **class_Y**. This is done by the subroutine **copy**().

[Flowchart]

Start

$0 \rightarrow i$

Scan Row
i > 79

$0 \rightarrow$ topic
$0 \rightarrow$ OK
$0 \rightarrow$ last

Topics
topic > 6

last + 1 $\rightarrow$ first
A $\rightarrow$ last

$0 \rightarrow$ correct

α

Test pass criteria
j: first, last, 1

(See Note)

Master_Table[i,j] = 1   No

Yes

correct + 1 $\rightarrow$ correct

Test pass criteria

B   No

Yes

OK + 1 $\rightarrow$ OK

topic + 1 $\rightarrow$ topic

Topics

C   No

Yes

$1 \rightarrow$ Master_Table [i,81]

$0 \rightarrow$ Master_Table [i,81]

i + 1 $\rightarrow$ i

Scan Row

copy ()

End
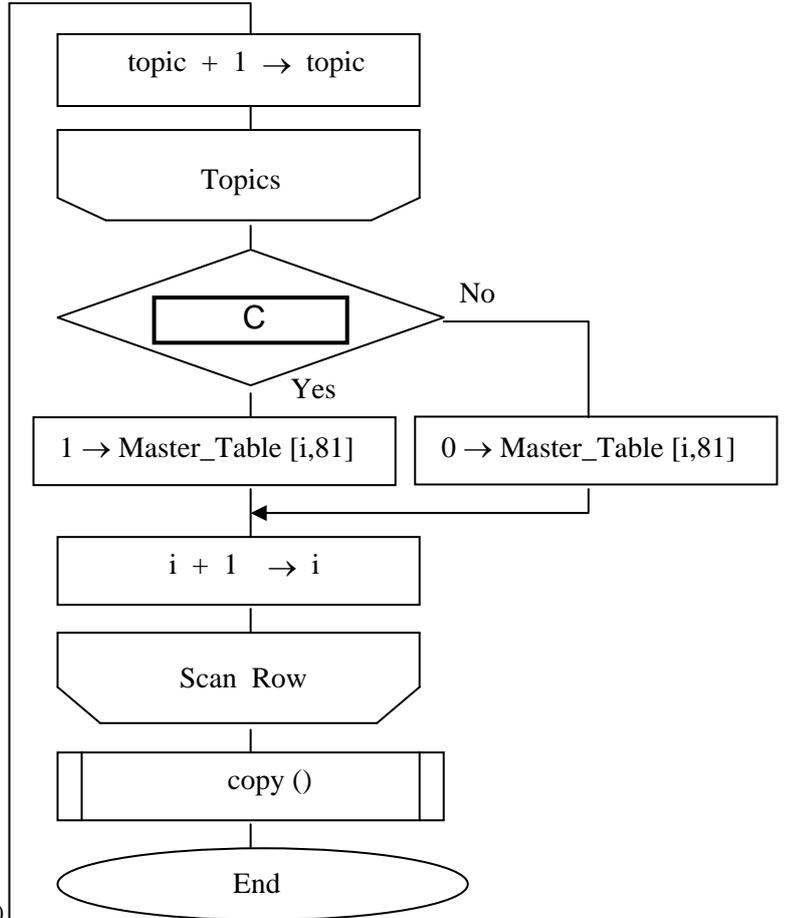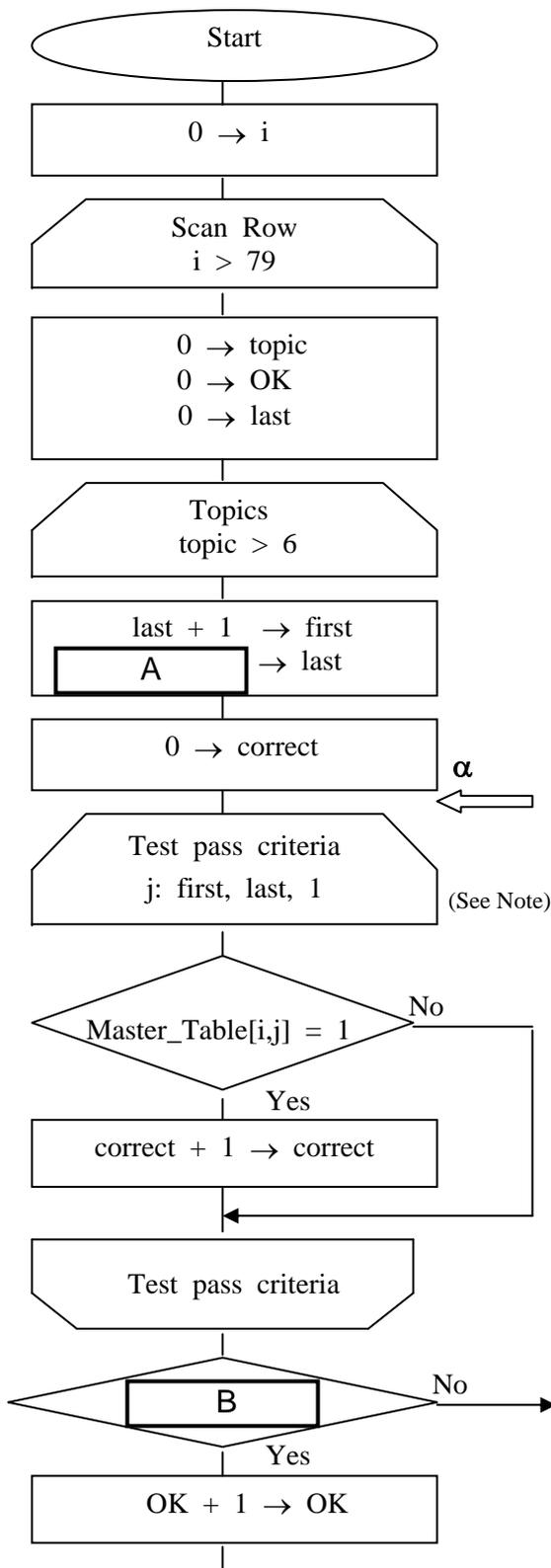
Note: The specification of the loop
conditions is as follows:
**variable name: initial value,
terminal value, increment**

Subquestion

From the answer groups below, select the correct answers to be inserted into the blanks [A] through [C] in the above flowchart.

And select the correct answer to be inserted into the blank [D] in the following description.

After execution of the process indicated by the arrow α, when the variable **Topic** = 2, the correct combination of values for the variables **first** and **last** are [D].

Answer group for A:

    a)   first + Topic_Table[topic, 0]   b)   first + Topic_Table [topic, 0] + 1

    c)   last + Topic_Table [topic, 0]   d)   last + Topic_Table [topic, 0] + 1

Answer group for B:

    a)   correct > 0   b)   correct = 1

    c)   correct <= Topic_Table [Topic, 1]   d)   correct = Topic_Table [Topic, 1]

    e)   correct >= Topic_Table [Topic, 1]

Answer group for C:

    a)   correct > 0   b)   correct = 1

    c)   correct = 6   d)   correct = 7

    e)   OK > 0   f)   OK = 1

    g)   OK = 6   h)   OK = 7

Answer group for D:

    a)   first = 1, last = 6   b)   first = 7, last = 45

    c)   first = 6, last = 46   d)   first = 7, last = 46

    e)   first = 47, last = 56   f)   first = 57, last = 62

Q6.   Read the following description of a C program and the program itself, and then answer Subquestion.

[Program Description]

To include a character string parameter in the URL during a request to CGI, the string parameter must be converted according to a predetermined transmission rule, and then transmitted.   The program `URLEncode` is used for the conversion.

1)   The string parameter to be converted consists of ASCII characters.   Assume that no null character (character code 0x00) is included in the string parameter.

2)   The conversion rule is as follows:

  (i)   Alphanumeric characters (0x30 - 0x39, 0x41 - 0x5A, 0x61 - 0x7A) and "@" (0x40), "*" (0x2A), "-" (0x2D), "." (0x2E), "_" (0x5F) are not converted. (These characters are called non-conversion characters hereinafter).

  (ii)   Characters not included in (i) are converted to 3 characters consisting of "%" followed by 2-digit hexadecimal character code.   For instance, the character with the character code 0x5E is converted to "%5E".

3)   The specification of function `URLEncode` is as follows:

```
Format:    void URLEncode( unsigned char *input,
                           unsigned char *output )
```

  Arguments: `input`   Pointer to the char type array that stores the character string to be converted

           `output`   Pointer to the char type array that stores the converted character string

4)   Assume that the array pointed by `output` has adequate space to store the converted character string.

5)   For example, the result of conversion of the string parameter "Hi!" is as shown in the figure below.

Fig.   Example of conversion of string parameter

6) The following function is available to the program:

Format: `int replaceChar(unsigned char c)`
Argument: `c`
Return value: If character `c` is a non-conversion character, 0 is returned.
             Otherwise, 1 is returned.

[Program]

```
int replaceChar( unsigned char );

void URLEncode( unsigned char *input, unsigned char *output ) {

    const unsigned char chars[] = "0123456789ABCDEF";

    while( *input != '\0' ) {
        if ( replaceChar(      A      ) ) {
            *output++ = '%';
            *output++ = chars[      B      ];
            *output++ = chars[      C      ];
        } else {
            *output++ = *input;
        }
            D      ;
    }
    *output = '\0';
}
```

Subquestion

From the answer groups below, select the correct answers to be inserted into the blanks ⬚ in the above program.

Answer group for A and D:

    a) `&input`   b) `(&input)++`   c) `(*input)++`

    d) `**input`   e) `**input++`   f) `*input`

    g) `input`   h) `input++`

Answer group for B and C:

    a) `*input << 4`   b) `*input >> 4`

    c) `*input & 0x0F`   d) `*input & 0xF0`

    e) `*input ^ 0x0F`   f) `*input ^ 0xF0`

    g) `*input | 0x0F`   h) `*input | 0xF0`

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

[Program Description]

This is a program that opens an input file containing the test results of all students in each grade in a particular institution and obtains the total scores in five subjects.   It sorts the results in descending order of the total scores and writes them onto output files so that the overall ranking in each grade can be determined.

1)   Input file `IN-FILE` is a sequential file which has the following record format, and stores the scores of all students in one grade.   Here, a perfect score for each subject is 100 points.

Class number	Student number	Name	Japanese language	Mathe-matics	English language	Science	Social studies
2 digits	2 digits	20 digits	3 digits	3 digits	3 digits	3 digits	3 digits

2)   Output file `OUT-FILE` is a sequential file which has the following record format, and stores data in descending order of the total scores.

Class number	Student number	Name	Total score
2 digits	2 digits	20 digits	3 digits

[Program]

(Line number)
```
 1   DATA DIVISION.
 2   FILE SECTION.
 3   FD IN-FILE.
 4   01 IN-REC          PIC X(39).
 5   FD OUT-FILE.
 6   01 OUT-REC         PIC X(27).
 7   SD SORT-FILE.
 8   01 SORT-REC.
 9      02 CLASS-NO     PIC 9(2).
10      02 STUDENT-NO   PIC 9(2).
11      02 STUDENT-NAME PIC X(20).
12      02 TOTAL        PIC 9(3).
13   WORKING-STORAGE SECTION.
14   01 W-IN-REC.
15      02 STUDENT-ID.
16         03 CLASS-NO     PIC 9(2).
17         03 STUDENT-NO   PIC 9(2).
18         03 STUDENT-NAME PIC X(20).
19      02 SCORE          PIC 9(3) OCCURS 5.
```

```
20   01 READ-STATUS  PIC X(1) VALUE SPACE.
21      88 AT-END    VALUE "E".
22   01 K            PIC 9(1).
23   PROCEDURE DIVISION.
24   SORT-PROCEDURE.
25      SORT SORT-FILE DESCENDING KEY TOTAL
26           INPUT  PROCEDURE IS READ-DATA
27           GIVING OUT-FILE.
28      STOP RUN.
29   READ-DATA.
30      OPEN INPUT IN-FILE.
31      PERFORM UNTIL AT-END
32         READ IN-FILE AT END
33   ┌──────────────────── A ────────────────────┐
34         NOT AT END
35            MOVE IN-REC TO W-IN-REC
36            PERFORM RELEASE-DATA
37         END-READ
38      END-PERFORM.
39      CLOSE IN-FILE.
40   RELEASE-DATA.
41   ┌──────────── B ────────────┐.
42      MOVE ZERO TO TOTAL.
43      PERFORM VARYING K FROM 1 BY 1 UNTIL K > 5
44         COMPUTE TOTAL = TOTAL + SCORE(K)
45      END-PERFORM.
46      RELEASE SORT-REC.
```

Subquestion 1

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

Answer group:

a) INITIALIZE SORT-REC

b) MOVE SPACE TO READ-STATUS

c) MOVE SPACE TO SORT-REC

d) MOVE STUDENT-ID TO SORT-REC

e) PERFORM RELEASE-DATA

f) SET AT-END TO TRUE

Subquestion 2

The program will be changed in such a way that the numbers of students and grade average points of all subjects will be displayed at the end of the program.   From the answer group below, select the correct answers to be inserted into the blanks ☐ in the following table showing description of changes in the program.

Here, the number of students is assumed to be in the 1 to 9,999 range, and the average scores are truncated to 2 decimal places.

Action	Description of changes in program

To be added between line numbers 22 and 23.	`01 STATISTICS.`
	`   02 STUDENT-TOTAL PIC 9(4) VALUE ZERO.`
	`   02 SUB-TOTAL     PIC 9(8) VALUE ZERO OCCURS 5.`
	`   02 AVERAGE      PIC 999.9.`

To be added between line numbers 27 and 28.	`DISPLAY "STUDENT:".`
	`DISPLAY STUDENT-TOTAL.`
	`DISPLAY "AVERAGE:".`
	`PERFORM VARYING K FROM 1 BY 1 UNTIL K > 5`
	`   COMPUTE AVERAGE = SUB-TOTAL(K) / STUDENT-TOTAL`
	`   DISPLAY AVERAGE`
	`END-PERFORM.`

To be added between line numbers 44 and 45.	**C**
To be added after line number 46.	**D** .

Answer group:

    a) `COMPUTE STUDENT-TOTAL = STUDENT-TOTAL + 1`

    b) `COMPUTE STUDENT-TOTAL = STUDENT-TOTAL + K`

    c) `COMPUTE SUB-TOTAL(K) = SUB-TOTAL(K) + SCORE(K)`

    d) `MOVE K TO STUDENT-TOTAL`

    e) `MOVE SCORE(K) TO SUB-TOTAL(K)`

    f) `MOVE TOTAL TO SUB-TOTAL(K)`

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

[Program Description]

The program will read in numbers from the user, then search the number in the predefined sorted array.

In the program, a binary search algorithm is used for finding the number.

The binary search begins by comparing the number to the one in the middle of the array. If not matched, it is obvious whether the number would belong before or after that middle number, because the numbers in the array are sorted.   The search then continues through the correct half in the same way.

[Program]

```java
public class BinarySearch {
    private long[] a;

    private int numberOfElements;

    public BinarySearch(int max) {
        a = new long[max];
        numberOfElements = 0;
}

public int size() {
    return numberOfElements;
}

public int search(long searchKey) {
    return BSearch(searchKey, 0, numberOfElements - 1);
}

private int BSearch(long searchKey, int lowerBound, int upperBound) {
    int currentPosition;

    currentPosition = (lowerBound + upperBound) / 2;
    if (a[currentPosition] == searchKey)
        return currentPosition;
    else if (lowerBound > upperBound)
        return numberOfElements;
    else
    {
    if (   A   )
        return BSearch(searchKey, currentPosition + 1, upperBound);
    else
        return BSearch(searchKey,    B   , currentPosition - 1);
    }
}
```

```java
public void insert(long value) {
    a[numberOfElements] = value;
    numberOfElements++;
}

public static void main(String[] args) {
    int maxSize = 100;
    BinarySearch br = new BinarySearch(maxSize);

    br.insert(121);
    br.insert(130);
    br.insert(150);
    br.insert(226);
    br.insert(314);
    br.insert(369);
    br.insert(444);
    br.insert(527);
    br.insert(695);
    br.insert(719);
    br.insert(723);
    br.insert(808);
    br.insert(944);
    br.insert(1017);
    br.insert(1053);
    br.insert(1296);

    int searchKey = Integer.parseInt(args[0]);
    if (     C     )
        System.out.println("Found " + searchKey);
    else
        System.out.println("Can't find " + searchKey);
    }
}
```

Subquestion

From the answer groups below, select the correct answers to be inserted into the blanks in the above program.

Answer group for A:

    a) `a[currentPosition] < searchKey`

    b) `a[currentPosition] > searchKey`

    c) `a[currentPosition] == searchKey`

    d) `a[currentPosition+1] > searchKey`

Answer group for B:

    a) `lowerBound`   b) `lowerBound + 1`

    c) `upperBound`   d) `upperBound + 1`

Answer group for C:

a) `br.search(searchKey)!=br.size()`

b) `br.search(searchKey)!=br.size()-1`

c) `br.search(searchKey)==br.size()`

d) `br.search(searchKey)==br.size()-1`

Q9. Read the following description of an assembler program and the program itself, and then answer Subquestions 1 and 2.

[Program 1 Description]

Program 1 (SFT1) is a subprogram that counts the number of "1" bits contained in one word as shifting a mask to the right, and sets the counted number in GR0.


	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 (Bit number)
Data to be processed	0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0

	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 (Bit number)
Mask	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Mask is shifted to the right. ▬▬▶
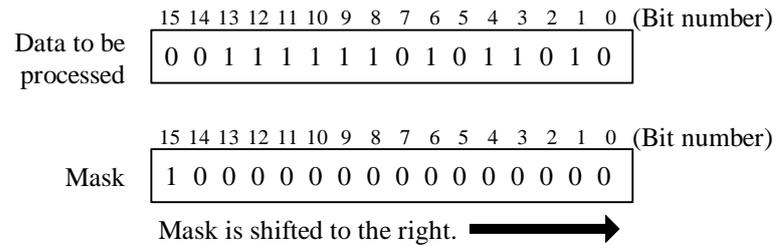
Fig. 1   Method of processing by Program 1

1) The main program loads the data to be processed into GR1 and calls the subprogram.
2) The 15th through 0th bits in the given data are compared with the mask in sequence. The number of bits whose value is 1 is counted, and the counted number is then set in GR0.
3) When control is returned from the subprogram, the contents of general-purpose registers GR1 through GR7 are restored to the original values.

[Program 1]

```
SFT1   START
       RPUSH
       LD    GR2,MASK
       LAD   GR0,0
LOOP   LD    GR3,GR2      ← α
       AND   GR3,GR1
       JZE   SKIP
       ADDA  GR0,=1
SKIP   SRL   GR2,1
       [      A      ]
       RPOP
       RET
MASK   DC    #8000
       END
```

[Program 2 Description]

Program 2 (SFT2) is a subprogram that counts the number of bits whose value is 1 by always comparing the lowest-order bit with the mask as the given data is shifted to the right so that the number of instruction executions can be smaller than in Program 1, and sets the counted number in GR0.

```
                15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0  (Bit number)
Data to be     ┌─────────────────────────────────────────────┐
processed      │ 0  0  1  1  1  1  1  1  0  1  0  1  1  0  1  0 │
               └─────────────────────────────────────────────┘
               Data is shifted to the right.        ──────────▶

                15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0  (Bit number)
               ┌─────────────────────────────────────────────┐
Mask           │ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 │
               └─────────────────────────────────────────────┘
```
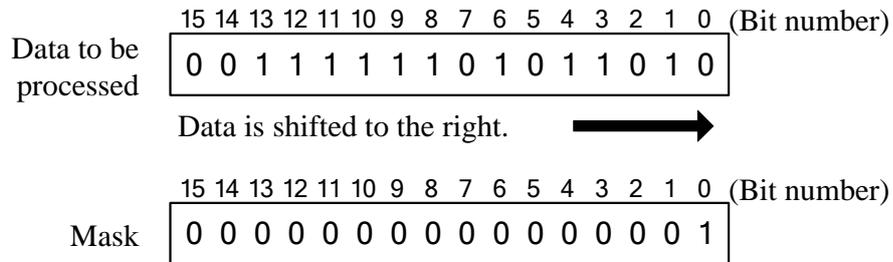
Fig. 2   Method of processing by Program 2

[Program 2]

```
SFT2   START
       RPUSH
       LAD    GR0,0
LOOP   LD     GR3,GR1      ◀─ β
       ┌─────────────────┐
       │        B        │
       └─────────────────┘
       ADDA   GR0,GR3
       SRL    GR1,1
       ┌─────────────────┐
       │        A        │
       └─────────────────┘
       RPOP
       RET
       END
```

Subquestion 1

From the answer groups below, select the correct answers to be inserted into the blanks ┌───────┐ in Programs 1 and 2.

Answer group for A:

a) JMI   LOOP   b) JNZ   LOOP

c) JUMP   LOOP   d) JZE   LOOP

Answer group for B:

a) AND   GR3,=#0001   b) OR   GR3,=#0001

c) AND   GR3,=#1000   d) OR   GR3,=#1000

e) AND   GR3,=#FFFF   f) OR   GR3,=#FFFE

Subquestion 2

From the answer group below, select the correct answers to be inserted into the blanks ⬚ in the following description.

If the main program loads #0555 into GR1 and calls Programs 1 and 2, then the number of executions of instruction β in Program 2 is smaller by ‖ C ‖ than the number of executions of instruction α in Program 1.

The number of executions of instruction α in Program 1 is equal to the number of executions of instruction β in Program 2 when the bit value of bit number ‖ D ‖ in the data that the main program loads into GR1 is ‖ E ‖.

Answer group:

a) 0   b) 1   c) 5   d) 6   e) 7

f) 8   g) 9   h) 10   i) 14   j) 15

Q10. Read the following description of a C program and the program itself, and then answer Subquestions 1 and 2.

[Program 1 Description]

This program reads a nonempty source program written in C language from a standard input, removes comments, and then outputs it to a standard output.

1) Description on the notation of source programs
   (i) "Comments" handled by this program are character strings that start with "/*" and end with "*/", excluding those included in character constants, character string literals, and comments.
   (ii) The type of usable characters is as follows.

Space	0	@	P	`	p
!	1	A	Q	a	q
"	2	B	R	b	r
#	3	C	S	c	s
$	4	D	T	d	t
%	5	E	U	e	u
&	6	F	V	f	v
'	7	G	W	g	w
(	8	H	X	h	x
)	9	I	Y	i	y
*	:	J	Z	j	z
+	;	K	[	k	{
,	<	L	\	l	|
-	=	M	]	m	}
.	>	N	^	n	~
/	?	O	_	o	

   (iii) Description as shown below is not used.
      ▪ Nested comments
        Example `/* aaaaa /* bbbbbb */ ccccc */`
      ▪ Three consecutive graphic characters
        `??= ??( ??/ ??' ??< ??> ??) ??! ??-`

   (iv) There are no grammatical errors.

2) Program 1 removes comments in accordance with the following procedure. Since the program simply processes the analyses of character constants, character string literals and comments, they may not be recognized correctly depending on the coding in source programs, resulting in a malfunction.

   (i) When detecting a single quote or double quote, the program interprets it as the beginning of a character constant or character string literal, and then uses function `quote` to read and output the character string as it is until the program detects the corresponding single quote or double quote.

   (ii) When detecting "/*", the program interprets it as the beginning of a comment and skips characters before the first appearance of "*/".

3) An execution example of the comment removal by Program 1 is shown below.

Input source program

```
/* This program uses fgets to display
 * a line from a file on the screen. */
#include <stdio.h>
int main( void )
{
  FILE *stream;  /* file pointer */
  char line[100]; /* input stream */

  if( (stream = fopen( "crt_fgets.txt", "r" )) != NULL )
  {
    if( fgets( line, 100, stream ) == NULL)
      printf( "fgets error\n" ); /* error message */
    else
      printf( "%s", line);
    fclose( stream );
  }
}
```

Output results after the removal of comments

```
#include <stdio.h>
int main( void )
{
  FILE *stream;
  char line[100];

  if( (stream = fopen( "crt_fgets.txt", "r" )) != NULL )
  {
    if( fgets( line, 100, stream ) == NULL)
      printf( "fgets error\n" );
    else
      printf( "%s", line);
    fclose( stream );
  }
}
```

Fig.   Execution Example of the Comment Removal

[Program 1]

```c
#include <stdio.h>
void quote( char );

main()
{
    int c1, c2;

    while ( (c1 = getchar()) != EOF ) {
        /* detection of single quote   */
        if ( c1 == '\'' ) quote( '\'' );
        /* detection of double quote   */
        else if ( c1 == '\"' ) quote( '\"' );
        /* detection of slash          */
        else if ( c1 == '/' ) {
            c2 = getchar();
            /* when the next character is an asterisk */
            if ( c2 == '*' ) {
                /* removal of comment character string */
                while ( 1 ) {
                    while ( (c1 = getchar()) != '*' );
                    c2 = getchar();
                    if ( c2 == '/' ) break;
                }
            }
            /* other cases   */
            else {
                putchar(c1);
                putchar(c2);
            }
        }
        else putchar(c1); /* one character read is outputted as it is */
    }
}

void quote( char c )
{   /* extraction of character constant and character string literal  */
    char cc;

    putchar(c);
    while ( (cc = getchar()) != c ) putchar(cc);
    putchar(cc);
}
```

Subquestion 1

From the answer group below, select the coding that causes wrong operation when it is entered into program 1.

Answer group:

```
a)  /*   "aaaaaaa"  */
b)  /* aaa  'a'    */
c)  if ( c == '\'' ) {
d)  printf( "  \'    " );
e)  printf( "aaa  /*  comment  */ \n" );
```

[Program 2 Description]

To solve the problem pointed out in 2) of **[Program 1 Description]**, Program 2 is then written as follows.

1) The process to be implemented is divided into the three modes: character constant, character string literal, and comment.

2) An appearance of a single quote switches the "character constant mode" between ON and OFF.   However, this does not apply to a piece of coding which is an expanded representation using a "\", to a piece of coding within a character string literal, or to a piece of coding within a comment.

3) An appearance of double quotes switches the "character string literal mode" between ON and OFF.   However, this does not apply to a piece of coding which is an expanded representation using a "\", to a piece of coding within a character constant, or to a piece of coding within a comment.

4) An appearance of "/*" and "*/" switches the "comment mode" between ON and OFF.   However, this does not apply to a piece of coding within a character constant or to a piece of coding within a character string literal.

[Program 2]

```c
#include <stdio.h>
main()
{
    int c1, c2;
    int c_mode = 0; /* set comment mode to off               */
    int quote1 = 0; /* set character constant mode to off     */
    int quote2 = 0; /* set character string literal mode to off */

    for ( c1 = getchar(); ( c2 = getchar()) != EOF; c1 = c2 ) {

        if ( !c_mode ) { /* when comment mode is off           */
            /* '\' in character constant or character string literal? */
            if ( [  A  ] && c1 == '\\' ) {
                putchar(c1);
                putchar(c2);
                c2 = getchar();
                continue;
            }
            /* single quote which is not inside a character string literal? */
            else if ( !quote2 && c1 == '\'' )
                [  B  ];
            /* double quote which is not inside a character constant? */
            else if ( !quote1 && c1 == '\"' )
                [  C  ];
            /* '/' and '*' which is not inside a character constant
                                  and character string literal? */
            else if ( [  D  ] && c1 == '/' && c2 == '*' ) {
                [  E  ];
                c2 = getchar();
                continue;
            }
            putchar(c1);
        }

        else {
            if ( c1 == '*' && c2 == '/' ) {  /* end of comment? */
                [  E  ];
                c2 = getchar();
            }
        }
    }
    putchar(c1);
}
```

- 33 -

Subquestion 2

From the answer groups below, select the correct answers to be inserted into the blanks [   ] in Program 2.

Answer group for A and D:

a) `!quote1`

b) `!quote2`

c) `(!quote1 || !quote2)`

d) `(!quote1 && !quote2)`

e) `(quote1 || quote2)`

f) `(quote1 && quote2)`

Answer group for B, C and E:

a) `c_mode = !c_mode`

b) `c_mode = quote1 && quote2`

c) `quote1 = !quote1`

d) `quote1 = !quote2`

e) `quote1 = quote2`

f) `quote2 = !quote1`

g) `quote2 = !quote2`

h) `quote2 = quote1`

Q11. Read the following description of a COBOL program and the program itself, and then answer Subquestion.

[Program Description]

Five television channels can be received in a certain geographical area. This program reads the view data file that records TV programs viewed by surveyed households in that area on a given day, and the TV program data file that records the TV program data on the same day. It then calculates the average audience rating and prints it. The program rounds down the time in units of one minute. For instance, the time 10:00 is the figure obtained by rounding down the time between 10:00:00 and 10:00:59.

1) The view data file (`VIEW-FILE`) is a sequential file with the following record format:

Channel No. 2 digits	Detection start time		Detection end time	
	Hour 2 digits	Minute 2 digits	Hour 2 digits	Minute 2 digits

(i) The view data file stores the audience data for a given day collected from the surveyed 600 sample households.

(ii) Assume that there is one television set per household.

(iii) Assume that the channel numbers range from 01 to 05.

(iv) A record shows the channel viewed by a given household, indicating the "channel no.", the "detection start time" and the "detection end time."

(v) Audience data is obtained by detecting the channels viewed by surveyed households at 00 second of every minute.
For instance, suppose that a certain household viewed TV from 10:00:30 to 10:05:30, during which it changed channels as follows:
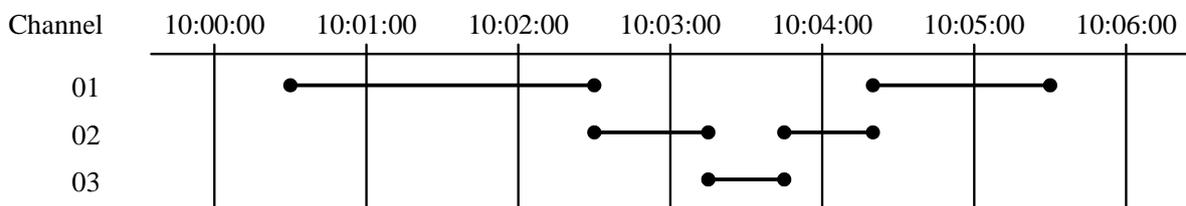
Since channels are detected at 00 second of every minute, the data is recorded in the view file, rounded down in minute units as follows. Channel 03 was viewed but not recorded.

```
0110011002
0210031004
0110051005
```

   (vi) The range of the detection start time and the detection end time is from 00:00 to 23:59. No data extends to the second day.

   (vii) The detection start time is equal to or less than the detection end time.

2) The program data file (PROGRAM-FILE) is a sequential file with the following record format:

Channel No. 2 digits	Program start time		Program end time		Program title 50 digits
	Hour 2 digits	Minute 2 digits	Hour 2 digits	Minute 2 digits	

   (i) The program data file stores program data for 5 channels, on the same day as in the view data file.

   (ii) The range of the program start time and the program end time is from 00:00 to 23:59. No data extends to the second day.

   (iii) The program start time is equal to or less than the program end time.

   (iv) A record shows that a program is broadcast from the "program start time" to the "program end time."

3) The program executes as follows:

   (i) The view count table (VIEW-COUNT-TABLE) is a two-dimensional table consisting of 5 (channels) × 1,440 (minutes). It summarizes the number of households by channel viewed in minute units and enters the data in each category of the table.

   (ii) The average audience rating of a TV program is obtained using the following equation:

Average audience rating of program (%)

    = (Sum of the time during which each household viewed that program)

     / (Number of sample households × broadcast time of the program) × 100

(iii) The channel, program start time, program end time, average audience rating (in percent, to the first decimal place), and the program name are printed in the order in which programs were read from the program data file, in the following format:

```
channel from - to    rating(%) program-title
   02   2320 - 2359    8.3     Sports News
   04   0430 - 0439    1.7     Good Morning
    ⋮      ⋮     ⋮       ⋮           ⋮
```

[Program]

```
DATA DIVISION.
FILE SECTION.
FD  VIEW-FILE.
01  VIEW-REC.
  05  VIEW-CHANNEL        PIC 99.
  05  VIEW-START-HHMM.
    10  VIEW-START-HH     PIC 99.
    10  VIEW-START-MM     PIC 99.
  05  VIEW-END-HHMM.
    10  VIEW-END-HH       PIC 99.
    10  VIEW-END-MM       PIC 99.
FD  PROGRAM-FILE.
01  PROGRAM-REC.
  05  PROG-CHANNEL        PIC 99.
  05  PROG-START-HHMM.
    10  PROG-START-HH     PIC 99.
    10  PROG-START-MM     PIC 99.
  05  PROG-END-HHMM.
    10  PROG-END-HH       PIC 99.
    10  PROG-END-MM       PIC 99.
  05  PROGRAM-TITLE       PIC X(50).
FD  OUT-FILE.
01  OUT-REC              PIC X(100).
WORKING-STORAGE SECTION.
01  SAMPLE-SIZE          PIC 9(4)  VALUE 600.
01  M                    PIC 9(4).
01  END-OF-FILE          PIC X.
01  START-MMMM           PIC 9(4).
01  END-MMMM             PIC 9(4).
01  SUMMATION            PIC 9(9)  BINARY.
01  PROG-RATING          PIC ZZ9.9.
01  VIEW-COUNT-TABLE.
  05  CHANNEL            OCCURS 5.
    10  COUNT-OF-MINUTE OCCURS 1440  PIC 9(4)  BINARY.
```

```cobol
01  O-DATA.
  05  FILLER                PIC X(3) VALUE SPACE.
  05  O-PROG-CHANNEL        PIC 99.
  05  FILLER                PIC X(3) VALUE SPACE.
  05  O-PROG-START-HHMM.
   10  O-PROG-START-HH      PIC 99.
   10  O-PROG-START-MM      PIC 99.
  05  FILLER                PIC X(3) VALUE " - ".
  05  O-PROG-END-HHMM.
   10  O-PROG-END-HH        PIC 99.
   10  O-PROG-END-MM        PIC 99.
  05  FILLER                PIC X(2) VALUE SPACE.
  05  O-PROG-RATING         PIC ZZ9.9.
  05  FILLER                PIC X(6) VALUE SPACE.
  05  O-PROGRAM-TITLE       PIC X(50).
01  TITLE-LINE             PIC X(100)
        VALUE "channel from - to   rating(%) program-title".
PROCEDURE DIVISION.
MAIN-PARAGRAPH.
    PERFORM EXPAND-VIEW-COUNT.
    PERFORM CALCULATE-RATING-AND-PRINT.
    STOP RUN.
EXPAND-VIEW-COUNT.
    [          A          ].
    OPEN INPUT VIEW-FILE.
    MOVE "N" TO END-OF-FILE.
    PERFORM UNTIL END-OF-FILE = "Y"
        READ VIEW-FILE AT END
            MOVE "Y" TO END-OF-FILE
        NOT AT END
            PERFORM SET-VIEW-COUNT
        END-READ
    END-PERFORM.
    CLOSE VIEW-FILE.
CALCULATE-RATING-AND-PRINT.
    OPEN INPUT PROGRAM-FILE  OUTPUT OUT-FILE.
    WRITE OUT-REC FROM TITLE-LINE AFTER ADVANCING 1.
    [          B          ].
    PERFORM UNTIL END-OF-FILE = "Y"
        READ PROGRAM-FILE AT END
            MOVE "Y" TO END-OF-FILE
        NOT AT END
            PERFORM CALCULATE-RATING
            MOVE PROG-CHANNEL    TO O-PROG-CHANNEL
            MOVE PROG-START-HHMM TO O-PROG-START-HHMM
            MOVE PROG-END-HHMM   TO O-PROG-END-HHMM
            MOVE PROG-RATING     TO O-PROG-RATING
            MOVE PROGRAM-TITLE   TO O-PROGRAM-TITLE
            WRITE OUT-REC FROM O-DATA AFTER ADVANCING 1
        END-READ
    END-PERFORM.
    CLOSE PROGRAM-FILE OUT-FILE.
```

```
SET-VIEW-COUNT.
    COMPUTE START-MMMM = VIEW-START-HH * 60 + VIEW-START-MM + 1.
    COMPUTE END-MMMM   = VIEW-END-HH   * 60 + VIEW-END-MM + 1.
    PERFORM VARYING M FROM START-MMMM BY 1 UNTIL M > END-MMMM
        [              C              ]
    END-PERFORM.
CALCULATE-RATING.
    COMPUTE START-MMMM = PROG-START-HH * 60 + PROG-START-MM + 1.
    COMPUTE END-MMMM   = PROG-END-HH   * 60 + PROG-END-MM + 1.
    [              D              ].
    PERFORM VARYING M FROM START-MMMM BY 1 UNTIL M > END-MMMM
        [              E              ]
    END-PERFORM.
    COMPUTE PROG-RATING ROUNDED
        = [              F              ].
```

Subquestion

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

Answer group for A, B and D:

   a)   COMPUTE SUMMATION = END-MMMM - START-MMMM

   b)   INITIALIZE VIEW-COUNT-TABLE

   c)   MOVE "N" TO END-OF-FILE

   d)   MOVE "Y" TO END-OF-FILE

   e)   MOVE SAMPLE-SIZE TO SUMMATION

   f)   MOVE ZERO TO SUMMATION

   g)   MOVE ZERO TO VIEW-COUNT-TABLE

Answer group for C and E:

   a)   ADD 1 TO COUNT-OF-MINUTE(VIEW-CHANNEL M)

   b)   ADD 1 TO COUNT-OF-MINUTE(VIEW-CHANNEL M + 1)

   c)   ADD 1 TO SUMMATION

   d)   ADD COUNT-OF-MINUTE(PROG-CHANNEL M) TO SUMMATION

   e)   COMPUTE SUMMATION = COUNT-OF-MINUTE(PROG-CHANNEL M + 1)

   f)   COMPUTE SUMMATION = COUNT-OF-MINUTE(VIEW-CHANNEL M) + 1

   g)   MOVE 1 TO COUNT-OF-MINUTE(VIEW-CHANNEL M)

Answer group for F:

a) `100 * SUMMATION / ((END-MMMM - START-MMMM) * (SAMPLE-SIZE + 1))`

b) `100 * SUMMATION / ((END-MMMM - START-MMMM + 1) * SAMPLE-SIZE)`

c) `100 * SUMMATION / (END-MMMM - START-MMMM) / SAMPLE-SIZE`

d) `100 * SUMMATION / (END-MMMM - START-MMMM + 1) / (SAMPLE-SIZE + 1)`

Q12. Read the following description of a Java program and the program itself, and then answer Subquestions 1 and 2.

[Program Description]

This program simulates the processing by an automated ticket gate installed at each station of a railroad line comprising four stations A, B, C, and D, with A serving as the starting point and D as the terminal point. On this line, the following two types of fare certificates can be used: one-way ticket (hereinafter, "ticket"); and prepaid card (hereinafter, "card").

A fare on the line is decided based on the distance between stations. The fare for up to 4 kilometers is 120 yen (this fare is called the base fare). An amount of 30 yen is added for each additional 2 kilometers. Any additional distance less than 2 kilometers is rounded up to 2 kilometers. For example, if the distance is 7 kilometers, the fare is 180 yen.

The station of embarkation is recorded on a ticket when a passenger passes through the automated ticket gate in that station to enter the platform area. When he/she passes through the automated ticket gate to leave the platform area in the station of disembarkation, the fare is calculated, and if the amount paid for the ticket is insufficient, the gate is closed to prevent him/her from leaving the platform area. Once a ticket is used, it becomes invalid. On this line, a passenger can enter the platform area through the automated ticket gate in any station, thus being able to embark, regardless of the station that issued the ticket. For instance, with a ticket issued at Station A, a passenger can enter the platform area through the automated ticket gate in Station B.

The station of embarkation is recorded on a card when a passenger enters the platform area through the automated ticket gate in that station. At this time, if the balance on the card is zero, the gate is closed to prevent him/her from entering the platform area. When he/she leaves the platform area through the automated ticket gate in the station of disembarkation, a fare adjustment is processed. Namely, the fare is calculated and is subtracted from the balance on the card. At this time, if this balance is less than the amount of the fare, the gate is closed to prevent him/her from leaving the platform area.

Class `Line` represents the railroad line. Method `getFare` calculates the fare on the basis of a given distance, and returns this information.

Class `Gate` signifies an automated ticket gate. Fields `A`, `B`, `C` and `D` in class `Line` are instances of `Gate`, and represent the automated ticket gates installed at stations A, B, C and D, respectively. The constructor and each method perform the following processing.

1) The constructor generates an instance of `Gate`. A station name is specified as the first argument, and the distance from station A, which is the starting point of the line,

is specified as the second argument.

2) Method `enter` performs processing when a passenger enters the platform area through an automated ticket gate.   If the fare certificate is not valid, the gate is closed.   If entrance processing is carried out normally, information on the station of embarkation is recorded on this fare certificate.

3) Method `exit` performs processing when a passenger leaves the platform area through an automated ticket gate.   If the fare certificate is invalid in that, for example, the amount (balance) on the fare certificate is insufficient, then the gate is closed.

4) Methods `open` and `close` output messages for opening and closing a gate, respectively.

Abstract class `Ticket`, which represents a fare certificate for this line, is inherited in defining a ticket and a card.   The constructor and each method perform the following processing.

1) The constructor retains a purchase amount on the fare certificate as the initial value.

2) Method `getValue` returns the fare certificate amount (balance) as it is at the point in time when it is called.

3) Method `adjustValue` performs fare adjustment processing if necessary.

4) Method `deduct` deducts the amount specified as an argument from the amount (balance) of the fare certificate, and updates the amount (balance).

5) Method `setOrigin` records a specified Gate as the station of embarkation.   If `null` is specified, the record of the station of embarkation is deleted.

6) Method `getOrigin` returns the station of embarkation that is recorded.   If this station is not recorded, `null` is returned.

Class `OneWayTicket` represents a ticket, and class `PrepaidCard` represents a card.   In the processing of either fare certificate, an abstract method is implemented, and the method in class `Ticket` is overridden, as necessary.

[Program 1]

```
public final class Line {
   public static final Gate A = new Gate("A", 0);
   public static final Gate B = new Gate("B", 5);
   public static final Gate C = new Gate("C", 8);
   public static final Gate D = new Gate("D", 14);

   public static int getFare(int distance) {
      return 120 + (Math.max(distance - 3, 0) / 2) * 30;
   }
}
```

[Program 2]

```java
public class Gate {
    private final String name;
    private final int distance;

    public Gate(String name, int distance) {
        this.name = name;
        this.distance = distance;
    }

    public void enter(Ticket ticket) {
        if (ticket.isValid() && ticket.getOrigin() == null) {
            [        A        ];
            open();
        } else {
            close();
        }
    }

    public void exit(Ticket ticket) {
        Gate origin = ticket.getOrigin();
        if (origin != null) {
            int d = Math.abs(origin.distance - distance);
            int fare = Line.getFare(d);
            if ([        B        ]) {
                ticket.adjustValue(fare);
                ticket.setOrigin(null);
                open();
                return;
            }
        }
        close();
    }

    private void open() { System.out.println(name + ": open"); }
    private void close() {
        System.out.println(name + ": closed");
    }
}
```

[Program 3]

```java
public abstract class Ticket {
    private Gate origin;
    private int value;

    public Ticket(int value) {
        this.value = value;
    }
```

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

    public void deduct(int amount) { value -= amount; }

    public void setOrigin(Gate gate) { origin = gate; }

    public Gate getOrigin() { return origin; }

    public abstract void adjustValue(int amount);

    public abstract boolean isValid();
}
```

[Program 4]
```
public class OneWayTicket extends Ticket {
    private boolean valid = true;

    public OneWayTicket(int value) {
        ┌─────────────────────┐
        │          C          │ ;
        └─────────────────────┘
    }

    public void setOrigin(Gate gate) {
        super.setOrigin(gate);
        if (gate == null)
            valid = false;
    }

    public void adjustValue(int amount) { }

    public boolean isValid() { return valid; }
}
```

[Program 5]
```
public class PrepaidCard extends Ticket {
    public PrepaidCard(int value) {
        ┌─────────────────────┐
        │          C          │ ;
        └─────────────────────┘
    }

    public void adjustValue(int amount) { deduct(amount); }

    public boolean isValid() {
        return getValue() > 0;
    }
}
```

Subquestion 1

From the answer groups below, select the correct answers to be inserted into the blanks
[＿＿＿＿] in the above programs.

Answer group for A:

    a) `ticket.setOrigin(Line.A)`   b) `ticket.setOrigin(Line.D)`

    c) `ticket.setOrigin(null)`   d) `ticket.setOrigin(this)`

    e) `ticket.setOrigin(ticket)`

Answer group for B:

    a) `ticket.getValue() < fare`   b) `ticket.getValue() <= fare`

    c) `ticket.getValue() == fare`   d) `ticket.getValue() > fare`

    e) `ticket.getValue() >= fare`

Answer group for C:

    a) `super()`   b) `super(this)`

    c) `super(value)`   d) `super(); this.value = value`

    e) `this(value)`

Subquestion 2

It was decided that a new type of fare certificate be sold.   A fare certificate of this type
permits a passenger to freely embark or disembark at all stations within 24 hours from the
time that the fare certificate is issued.   Twenty-four hours after such a fare certificate is
issued, the passenger can leave the platform area in any station but cannot enter any such
area.   For this, it is desired that a new class that inherits abstract class `Ticket` be defined
to permit the use of this type of certificate without making any modification to class `Gate`.
Processing by the constructor and methods of this class are shown in the following table.
From the answer group below, select the correct answers to be inserted into the blanks
in the table.   Here, `value` in the answer group is a field `value` of class `Ticket`, and it is
assumed that the initial value is set by the constructor and that the value is obtained by
method `getValue`.

Constructor and methods	Processing
Constructor	D
Method getValue	E
Method setOrigin	The same as is defined in the superclass
Method getOrigin	The same as is defined in the superclass
Method adjustValue	No processing is performed (methods proper do not contain any statements).
Method isValid	F

Answer group:

a) 0 is returned at all times.

b) An amount that is theoretically too large to be spent in 24 hours is set as the initial value of value.

c) Method deduct is called, with amount as an argument.

d) No processing is performed (methods proper do not contain any statements).

e) The base fare on the line is returned at all times.

f) The highest fare on the line (the maximum value among all fares) is set as the initial value of value, and the fare certificate issuance time is recorded.

g) The same as is defined in the superclass

h) true is returned only when the call time is within 24 hours from the fare certificate issuance time stored in the instance.   In other cases, false is returned.

i) true is returned only when value is not less than the base fare on the line.   In other cases, false is returned.

j) true is returned only when value is not less than the highest fare on the line (the maximum value among all fares).   In other cases, false is returned.

Q13.   Read the following description of an assembler program and the program itself, and then answer Subquestions 1 through 3.

[Program Description]

This is a subprogram `BPSRH` in which one word is searched for a specified bit pattern.

1)   The main program sets the first address of the parameter area in `GR1`, and calls `BPSRH`.   The format of the parameter area is as follows:

(GR1)→ 0	Word to be searched
→ 1	n
→ 2	Bit pattern (right-justified)

n: Number of bits in bit pattern
（ 1<=n <=16）
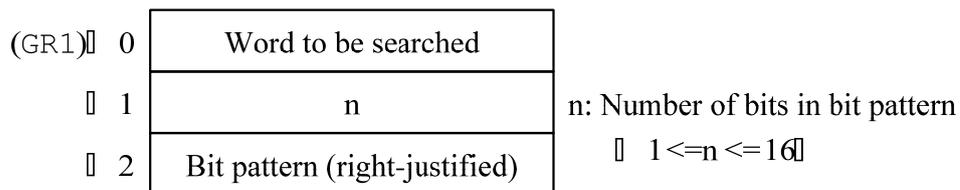
2)   In `BPSRH`, bits in the word to be searched are compared with the specified bit pattern in descending order of bit number, and the highest bit number in the portion that first matched is set in `GR0`; `BPSRH` then returns to the main program.   If no portion is matched, –1 is set in `GR0`, and `BPSRH` returns to the main program.   In the following example, the highest bit number in the portion that matched with the pattern is 12, which is set in `GR0`.
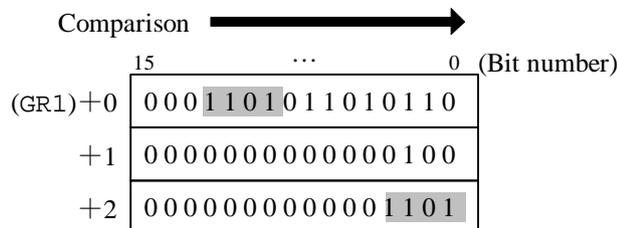
3)   When returning from the subprogram, the contents of general-purpose registers `GR1` through `GR7` are restored to the original values.

[Program]

(Line number)

```
 1  BPSRH  START
 2         RPUSH
 3         LD    GR6,1,GR1
 4         LAD   GR7,16
 5         SUBA  GR7,GR6   ; GR7 <- (16 - n)
 6         LD    GR2,2,GR1
 7         SLL   GR2,0,GR7 ; With bit pattern left-justified
 8         LAD   GR4,-1
 9  ┌──────────A──────────┐ ;  Generation of mask pattern
10         LAD   GR0,-1    ; Initialization of return value
11         LAD   GR3,0     ; Initialization of comparison location pointer
12         LD    GR5,0,GR1
13  LOOP   LD    GR6,GR5   ; GR6 is for temporary use
14         AND   GR6,GR4
15         CPL   GR6,GR2   ; Comparison with bit pattern
16         JZE   FIND
17         LAD   GR3,1,GR3 ; Next comparison location is set.
18         CPA   GR3,GR7   ; Does uncompared portion contain n bits or more?
19         JPL   EXIT
20  ┌──────────B──────────┐
21         JUMP  LOOP
22  FIND   LAD   GR0,15    ; Calculation of bit number
23         SUBA  GR0,GR3
24  EXIT   RPOP
25         RET
26         END
```

Subquestion 1

From the answer groups below, select the correct answers to be inserted into the blanks ☐ in the above program.

Answer group for A:

    a) SLL   GR4,0,GR6   b) SLL   GR4,0,GR7

    c) SRA   GR4,0,GR6   d) SRA   GR4,0,GR7

    e) SRL   GR4,0,GR6   f) SRL   GR4,0,GR7

Answer group for B:

    a) LD   GR5,1,GR2   b) SLL   GR3,1

    c) SLL   GR5,1   d) SRL   GR3,1

    e) SRL   GR5,0,GR2   f) SRL   GR5,0,GR3

    g) SRL   GR5,1

Subquestion 2

From the answer group below, select the correct answer as the hexadecimal notation of the value of GR5 when the following parameters are passed and control is just transferred to the instruction labeled FIND.

(GR1)＋0	0 0 0 1 1 0 1 0 1 1 0 1 0 1 1 0
＋1	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
＋2	0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1

Answer group:

a) 000D
b) 1AD6
c) AD60
d) D000
e) D6B0

Subquestion 3

A subprogram RESERVE was created in which movie theater seats are reserved using a subprogram BP1SRH for performing searches specializing in bit patterns comprising consecutive 1's.

From the answer groups below, select the correct answers to be inserted into the blanks [   ] in RESERVE.

1) The number of seats for reservation in the movie theater is 1,024.   Seat numbers are 0 through 1023.   Moreover, seats for reservation are divided in such a way that every 16 seats with consecutive seat numbers constitute a group.   A table for seat control consists of 64 consecutive words.   Bit number 15 of the first word represents the state of seat number 0, and bit number 0 of the last word represents the state of seat number 1023.   When the corresponding bit in the table for seat control is 1, the relevant seat is vacant.   If the bit is 0, the relevant seat is booked.

Bit number   15 ··· 0 15 ··· 0   15 ··· 0

Seat number   0   16   1023

Bit values. 1: Vacant   0: Booked
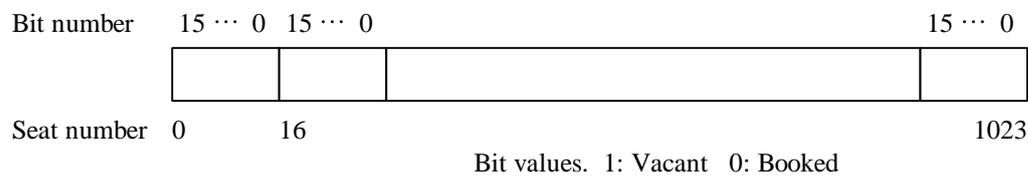
Fig.　Format of table for seat control (64 words)

2) In the main program, the number of seats to be booked, n (1 <= n <= 16), is set in GR1, the beginning address of the table for controlling seats for reservation is set in GR2, and RESERVE is called.

3) RESERVE secures a specified number of seats in such a way that all relevant seats are consecutive and that no group is skipped in between.   Vacant seats are searched for

in sequence starting with seat number 0.   If consecutive vacant seats are found, these seats are placed in the "booked" state, the lowest seat number secured is set in GR0, and control is returned to the main program.   If no vacant seat is secured, –1 is set in GR0, and control is returned to the main program.

4)   When returning from the subprogram RESERVE, the contents of general-purpose registers GR1 through GR7 are restored to the original values.

5)   The format of the parameters given to BP1SRH are as follows.

(GR1) 　 0	Word to be searched	
1	n	n: Number of bits comprising consecutive 1's 　 $1 <= n <= 16$
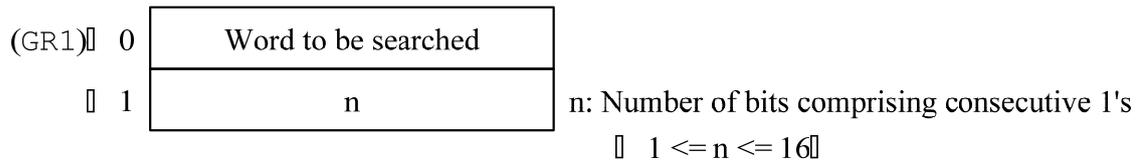
BP1SRH is a program in which line numbers 6 through 9 of BPSRH are replaced with the following three sets of instructions.

```
LAD    GR2,#8000
SRA    GR2,-1,GR6
LD     GR4,GR2
```

[Program]

```
RESERVE  START
         RPUSH
         LD     GR6,GR1       ; Save n
         LAD    GR1,PARAM
         ST     GR6,1,GR1     ; Preparation of parameters for calling BP1SRH (1)
         ST     GR2,TBLADD    ; Save beginning address of table for seat control
         LAD    GR4,64,GR2
         LAD    GR0,-1        ; Initialization of return value
LOOP     CPL    GR2,GR4       ; End of search?
         JZE    EXIT
         LD     GR5,0,GR2     ; Load one word from table for seat control
         ST     GR5,0,GR1     ; Preparation of parameters for calling BP1SRH (2)
         CALL   BP1SRH        ; Searching for vacant seats in one word
         CPA    GR0,=-1
         JNZ    FIND
         LAD    GR2,1,GR2     ; Going to search next word
         JUMP   LOOP
FIND     LAD    GR3,15
         SUBA   GR3,GR0       ; GR3 <- (15 - GR0)
         LAD    GR7,#8000
         SRA    GR7,-1,GR6
                C
         XOR    GR7,=#FFFF    ; GR7 <- 1110000111111111 (In case GR0 = 12, n = 4)
         AND    GR5,GR7       ; Set at booked state.
```

```
        ST      GR5,0,GR2
        SUBL    GR2,TBLADD      ;  Calculation of seat number
        ┌─────────────────┐
        │       D         │
        └─────────────────┘
        ADDA    GR2,GR3
        LD      GR0,GR2
EXIT    RPOP
        RET
TBLADD  DS      1
PARAM   DS      2               ;  Parameter areas for calling BP1SRH
        END
```

Answer group for C:

    a) AND   GR5,GR3   b) AND   GR5,GR7

    c) OR   GR5,GR3   d) OR   GR5,GR7

    e) SRA   GR7,0,GR3   f) SRL   GR7,0,GR3

Answer group for D:

    a) SLL   GR2,1   b) SLL   GR2,2

    c) SLL   GR2,4   d) SLL   GR3,1

    e) SLL   GR3,2   f) SLL   GR3,4

Assembly Language Specifications

1. COMET II Hardware Specifications
1.1 Hardware Specifications

(1) One word is 16 bits, and the bit format is as follows:

	Upper 8 bits								Lower 8 bits							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	(Bit No.)

Sign (Negative:1, Positive:0)
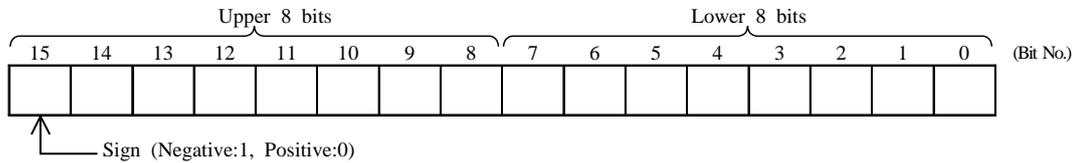
(2) Main storage capacity is 65,536 words with address numbers 0 through 65,535.
(3) Numeric values are expressed as 16-bit binary numbers. Negative numbers are expressed in complements of two.
(4) Control is sequential. COMET II utilizes a one-word or two-word instruction word.
(5) The COMET II has four types of registers: GR (16 bits), SP (16 bits), PR (16 bits) and FR (3 bits).
There are eight GR (General Register) registers, GR0 through GR7. These eight registers are used for arithmetic, logical, compare and shift operations. Of these, GR1 through GR7 are also used as index registers to modify addresses.
The stack pointer stores the address currently at the top of the stack.
The PR (Program Register) stores the first address of the next instruction.
The FR (Flag Register) consists of three bits: OF (Overflow Flag), SF (Sign Flag) and ZF (Zero Flag). The following values are set, depending on the result generated by certain operation instructions. These values are referenced by conditional branch instructions.

OF	When the result of an arithmetic operation instruction is out of the range of –32,768 to 32,767, the value is 1, and in other cases, the value is 0. When the result of a logical operation instruction is out of the range of 0 to 65,535, the value is 1, and in other cases, the value is 0.
SF	When the sign of the operation result is negative (bit number 15 = 1), the value is 1, and in other cases, the value is 0.
ZF	When the operation result is 0 (all bits are 0), the value is 1, and in other cases, the value is 0.

(6) Logical addition or logical subtraction: Treats the data to be added or subtracted as unsigned data, and performs addition or subtraction.

1.2 Instructions

Formats and functions of instructions are described in the following chart. When an instruction code has two types of operands, the upper operand shows the instruction between registers and the lower operand shows the instruction between register and main storage.

Instruction	Format		Description of instructions	FR setting
	Opcode	Operand		

(1) Load, store, load address instruction

Instruction	Opcode	Operand	Description	FR
LoaD	LD	r1,r2	r1 ← (r2)	○*1
		r,adr [,x]	r ← (effective address)	
STore	ST	r,adr [,x]	Effective address ← (r)	–
Load ADdress	LAD	r,adr [,x]	r ← effective address	

(2) Arithmetic and logical operation instructions

Instruction	Opcode	Operand	Description	FR
ADD Arithmetic	ADDA	r1,r2	r1 ← (r1) + (r2)	○
		r,adr [,x]	r ← (r) + (effective address)	
ADD Logical	ADDL	r1,r2	r1 ← (r1) +$_L$ (r2)	
		r,adr [,x]	r ← (r) +$_L$ (effective address)	
SUBtract Arithmetic	SUBA	r1,r2	r1 ← (r1) – (r2)	
		r,adr [,x]	r ← (r) – (effective address)	
SUBtract Logical	SUBL	r1,r2	r1 ← (r1) –$_L$ (r2)	
		r,adr [,x]	r ← (r) –$_L$ (effective address)	
AND	AND	r1,r2	r1 ← (r1) AND (r2)	○*1
		r,adr [,x]	r ← (r) AND (effective address)	
OR	OR	r1,r2	r1 ← (r1) OR (r2)	
		r,adr [,x]	r ← (r) OR (effective address)	
eXclusive OR	XOR	r1,r2	r1 ← (r1) XOR (r2)	
		r,adr [,x]	r ← (r) XOR (effective address)	

(3) Compare operation instructions

Instruction	Opcode	Operand	Description of instructions	FR
ComPare Arithmetic	CPA	r1,r2	Performs an arithmetic compare or logical compare operation on (r1) and (r2) or (r) and (effective address), and sets FR as follows, according to the result of the compare operation.	○*1
		r,adr [,x]		
ComPare Logical	CPL	r1,r2		
		r,adr [,x]		

Compare result	FR value	
	SF	ZF
(r1) > (r2) / (r) > (effective address)	0	0
(r1) = (r2) / (r) = (effective address)	0	1
(r1) < (r2) / (r) < (effective address)	1	0

(4) Shift operation instructions

Shift Left Arithmetic	SLA r,adr [,x]	Shifts (r) (excluding the sign bit) left or right by the number of bits specified by the effective address. When a left shift is performed, those bits that are left vacant by the shift operation are filled with zeroes. When a right shift is performed, those bits that are left vacant by the shift operation are filled with the same value as the sign bit.	○*2
Shift Right Arithmetic	SRA r,adr [,x]		
Shift Left Logical	SLL r,adr [,x]	Shifts (r) (including the sign bit) left or right by the number of bits specified by the effective address. Those bits that are left vacant by the shift operation are filled with zeroes.	
Shift Right Logical	SRL r,adr [,x]		

(5) Branch instructions

Jump on PLus	JPL adr [,x]	Branches to the effective address, depending on the value of FR. If control does not branch to a new address, execution continues with the next instruction.	–
Jump on MInus	JMI adr [,x]		
Jump on Non Zero	JNZ adr [,x]		
Jump on ZEro	JZE adr [,x]		
Jump on OVerflow	JOV adr [,x]		
unconditional JUMP	JUMP adr [,x]	Branches unconditionally to the effective address.	

Instruc-tion	Value of FR in order to branch		
	OF	SF	ZF
JPL		0	0
JMI		1	
JNZ			0
JZE			1
JOV	1		

(6) Stack operation instructions

PUSH	PUSH adr [,x]	SP ← (SP) $-_L$ 1, (SP) ← effective address	–
POP	POP r	r ← ((SP)), SP ← (SP) $+_L$ 1	

(7) Call and return instructions

CALL subroutine	CALL adr [,x]	SP ← (SP) $-_L$ 1, (SP) ← (PR), PR ← effective address	–
RETurn from subroutine	RET	PR ← ((SP)), SP ← (SP) $+_L$ 1	

(8) Other

SuperVisor Call	SVC adr [,x]	Determine based on the effective address as the argument. After the execution, GR and FR are undefined.	–
No OPeration	NOP	N/A	

Note) r, r1, r2   All of these represent GR. Values from GR0 to GR7 can be specified.

adr   This represents the address. A value from 0 to 65,535 can be specified.

x   This represents GR used as the index register. A value from GR1 to GR7 can be specified.

[   ]   Square brackets ([   ]) indicate that the specification contained in the brackets may be omitted.

(   )   The contents of the register or address contained in the parentheses (   ).

Effective   A value produced by adding, through "logical addition," adr and the
address   contents of x, or the address pointed at by that value.

←   This means that the operation result is stored in the left part register or address.

+L, −L   Logical addition and logical subtraction.

Effective   O   : Setting is performed.
address for O*1 : Setting is performed, but 0 is set to OF.
FR setting   O*2 : Setting is performed, but the bit value sent from the register is set to OF.
             −   : The value before execution is stored.

1.3 Character Set

(1) A JIS X0201 Romaji/katakana character set that uses 8-bit codes is used.

(2) Part of the character set is shown in the right table. Eight bits are used to represent one character; the upper four bits indicate the column in the table, and the lower four bits indicate the row. For example, the hexadecimal codes for the space character, "4," "H," and "\" are 20, 34, 48 and 5C, respectively. The characters that correspond to the hexadecimal codes 21 to 7E (and A1 to DF omitted in this table) are called "graphics characters." Graphics characters can be displayed (printed) as characters on an output device.

(3) If any characters not listed in this table and the bit configuration for those characters is needed, they are given in the problem.

Column / Row	02	03	04	05	06	07
0	Space	0	@	P	`	p
1	!	1	A	Q	a	q
2	"	2	B	R	b	r
3	#	3	C	S	c	s
4	$	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	'	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
10	*	:	J	Z	j	z
11	+	;	K	[	k	{
12	,	<	L	\	l		
13	-	=	M	]	m	}
14	.	>	N	^	n	~
15	/	?	O	_	o	

2   Specifications of the CASL II Assembly Language

2.1   Specifications of the language

(1)   CASL II is an assembly language for the COMET II.
(2)   A program consists of instruction lines and comment lines.
(3)   One instruction is described in one instruction line, and cannot continue to the next line.
(4)   Instruction lines and comment lines are written from the first character of the line in the following description formats:

Line type		Description format
Instruction line	With operand	[label]{blank}{instruction code}{blank}{operand}[{blank}[comment]]
	Without operand	[label]{blank}{instruction code}[{blank}[{;}[comment]]]
Comment line		[blank]{;}[comment]

(Note)   [   ]   Square brackets ([   ]) indicate that the specification contained in the brackets may be omitted.

{   }   Braces ({   }) indicate that the specification contained in the braces is mandatory.

Label   Label is the name used to refer to the address of (the first word of) the instruction from other instructions and programs. A label must be 1 to 8 characters in length, and the leading character must be an uppercase alphabetic letter. Either uppercase alphabetic letters or numeric characters can be used for the subsequent characters. Reserved words, GR0 through GR7, are not available.

Blank   One or more space characters.

Instruction code   The description format is defined by instruction.

Operand   The description format is defined by instruction.

Comment   Optional information such as memorandums that can be written in any characters allowed by the processing system.

2.2   Instruction Types

CASL II consists of four assembler instructions (START, END, DS and DC), four macro instructions (IN, OUT, RPUSH and RPOP) and machine language instructions (COMET II instructions). The specifications are as follows:

Instruction type	Label	Instruction code	Operand	Function
Assembler instruction	Label	START	[Execution start address]	Defines the top of a program. Defines the starting address for execution of a program. Defines the entry name for reference in other programs.
		END		Defines the end of a program.
	[label]	DS	Word length	Allocates an area.
	[label]	DC	Constant[, constant]···	Defines a constant.
Macro instruction	[label]	IN	Input area, input character length area	Input character data from input devices.
	[label]	OUT	Output area, output character length area	Output character data from output devices.
	[label]	RPUSH		Stores the contents of GR in the stack
	[label]	RPOP		Stores the contents of stack in GR
Machine language instruction	[label]		(See "1.2 Instructions")	

2.3 Assembler Instructions

Assembler instructions are used for assembler control, etc.

(1) | START | [Execution start address] |

The START instruction defines the top of a program.

The label name that is defined within this program specifies the execution start address. If the label is specified, execution begins from the address, and if the label is omitted, execution begins from the next instruction of the START instruction.

The label for this instruction can be referred to from other programs as the entry name.

(2) | END |

The END instruction defines the end of a program.

(3) | DS | Word length |

The DS instruction allocates an area of the specified word length.

The word length is specified by a decimal constant (≥ 0). If "0" is specified for the word length of an area, the area is not allocated, but the label is valid.

(4) | DC | Constant[, constant] ⋯ |

The DC instruction stores the data that has been specified as a constant in (consecutive) words.

There are four types of constants: decimal constants, hexadecimal constants, character constants and address constants.

Type of constant	Format	Description of instruction
Decimal constant	n	This instruction stores the decimal value specified by "n" as one word of binary data. If "n" is outside of the range of –32,768 to 32,767, only the lower 16 bits of n are stored.
Hexadecimal constant	#h	Assume "h" is a four-digit hexadecimal number. (Hexadecimal notation uses 0 through 9 and A through F.) This instruction stores the hexadecimal value specified by "h" as one word of binary data. ($0000 \leq h \leq FFFF$)
Character constant	'character string'	This instruction allocates a continuous area for the number of characters (> 0) in the character string. The first character is stored in bits 8 through 15 of the first word, the second character is stored in bits 8 through 15 of the second word, and so on, so that the character data is stored sequentially in memory. Bits 0 through 7 of each word are filled with zeroes. Spaces and any of the graphics characters can be written in a character string. Apostrophes (') must be written twice consecutively.
Address constant	Label	This instruction stores an address corresponding to the label name as one word of binary data.

2.4 Macro Instructions

Macro instructions use a pre-defined group of instructions and operand data to generate a group of instructions that performs a desired function (the word length is undefined).

(1) | IN | Input area, input character length area |

The IN instruction reads one record of character data from a previously assigned input device.

The input area operand should be the label of a 256-word work area, and the input data is input in this area beginning at the starting address, one character per word. No record delimiter code (such as a line return code, when using a keyboard) is stored. The storage format is the same as character constants with the DC instruction. If the input data is less than 256 characters long, the previous data is left as is in the remaining portion of the input area. If the input data exceeds 256 characters, the excess characters are ignored.

The input character length area should be the label of the one-word work area, and the character length that was input ($>= 0$) is stored as binary data. If the end-of-file indicator is encountered, -1 is stored.

When the IN instruction is executed, the contents of GR registers are saved but the contents of FR are undefined.

(2) | OUT | Output area, output character length area |

The OUT instruction writes character data as one record of data to the previously assigned output device.

The output area operand should be the label of the area where the data to be output is stored, one character per word. The storage format is the same as character constants with the DC instruction. Bits 0 through 7 do not have to be zeroes because the OS ignores them.

The output character length area should be the label of the one-word work area, and the character length that is to be output (>= 0) is stored as binary data.

When the OUT instruction is executed, the contents of the GR registers are saved but the contents of FR are undefined.

(3) | RPUSH | |

The RPUSH instruction stores the contents of GR in the stack in order of GR1, GR2, … and GR7.

(4) | RPOP | |

The RPOP instruction takes out of the contents of the stack sequentially, and stores in GR in order of GR7, GR6, … and GR1.

2.5 Machine Language Instructions

Operands of machine language instructions are described in the following formats:

r, r1, r2	GR is specified using a symbol from GR0 to GR7.
x	GR used as the index register can be specified by a symbol from GR1 to GR7.
adr	The address is specified by a decimal constant, a hexadecimal constant, an address constant or a literal. A literal can be described by attaching the equal sign (=) before a decimal constant, a hexadecimal constant or a character constant. CASL II generates a DC instruction by specifying the constant after the equal sign as the operand, and sets the address to the adr value.

2.6 Other

(1) The relative positions of the instruction words and areas generated by the assembler conform to the order of the descriptions in the assembly language program. All DC instructions generated from literals are located just before the END instruction.

(2) The instruction words and areas that are generated occupy a continuous area in the main memory.

3. Guide to Program Execution

3.1 OS

The following arrangements exist regarding program execution.

(1) The assembler interprets undefined labels (of those labels written in the operand column, any that are not defined within the program) as entry names (START instruction labels) for other programs. In this case, the assembler refrains from determining the address and entrusts that task to the OS. Before executing the program, the OS performs link processing with entry names for other programs and determines the addresses (program linking).

(2) The program is started up by the OS. Although the area in the main memory where a program is loaded is undefined, the address value corresponding to the label in the program is corrected to the actual address by the OS.

(3) During program startup, the OS allocates enough stack area for the program, then adds one to the last address and sets that value in the SP.

(4) The OS passes control to the program by the CALL instruction. When returning control to the OS after executing the program, the RET instruction is used.

(5) The assignment of an input device to the IN instruction or of an output device to the OUT instruction is made by the user before executing the program.

(6) The OS handles the differences that may arise in input and output procedures due to the different I/O devices and media involved; I/O is performed using the system's standard format and procedures (including error handling).   Therefore, the user of these IN and OUT instructions does not need to be concerned with differences among I/O devices.

3.2 Undefined Items

Ensure that any items concerning program execution that are not defined in these specifications are handled by the processing system.