Fundamental IT Engineer Examination (Afternoon)

Trial

Questions must be answered in accordance with the following:

Question Nos.	Q1 - Q5	Q6 - Q7	Q8 - Q9
Question Selection	Compulsory	Select 1 of 2	Select 1 of 2
Examination Time	150 minutes		

[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. [optional] If it is False, then Process 2 is executed.
■ Conditional expression ▪ Process ■	A repetition process with the 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

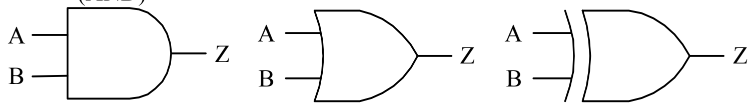
[Logical type constant]

```
true  false
```

Questions 1 through 5 are all compulsory.　Answer every question.

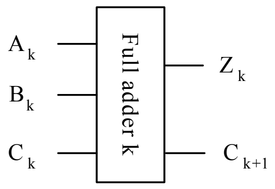
Q1. Read the following descriptions of logical operations and full adders, and then answer Subquestions 1 through 3.

(1) The logical circuit symbols for the main logical operations are as follows.

Logical operation name	Logical product (AND)	Logical sum (OR)	Exclusive logical sum (XOR)
Logical circuit symbols			

(A, B: Input, Z: Output)

(2) Below is a figure which shows a full adder that adds binary numbers digit by digit with considerations for carry.　The table shown below is the truth table for that full adder.



A_k, B_k: Input

Z_k: Output

C_k: Carry from k-1th digit

C_{k+1}: Carry to k+1th digit

Fig.　Full Adder (k-th digit)

Table　Truth Table of Full Adder

Input			Output	
C_k	A_k	B_k	C_{k+1}	Z_k
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1		
1	1	0	1	0
1	1	1	1	1

Subquestion 1

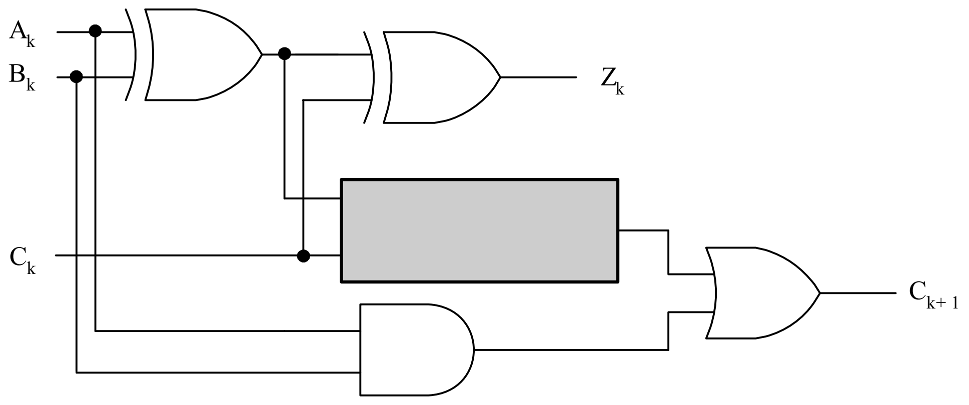
From the answer group below, select the correct answer to be inserted in the blank $\boxed{\qquad}$ in the truth table of the full adder.

Answer group:

a)	0	0	b)	0	1
c)	1	0	d)	1	1

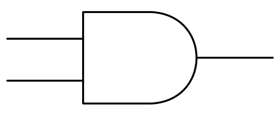
Subquestion 2

From the answer group below, select the correct answer to be inserted in the blank $\boxed{\qquad}$ in the logical circuit of the full adder.

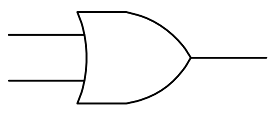


Answer group:

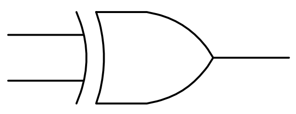
a)



b)

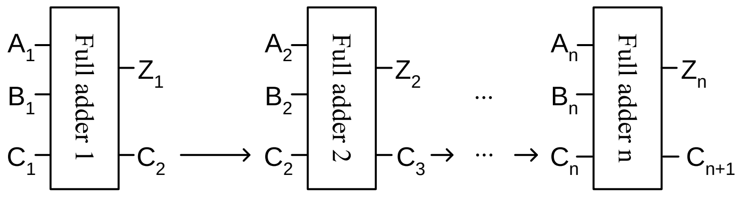


c)

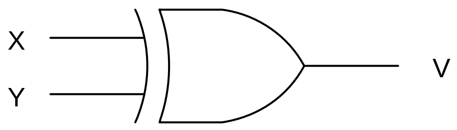


Subquestion 3

When a logical circuit is configured with full adders to add n-digit binary numbers represented as two's complement, the addition of the most significant digits (A_n, B_n and C_n) causes an overflow (the shaded part of the full adder truth table). A logical circuit for detecting this can be configured with one XOR. Select from the answer group below the correct combination of X and Y inputs to this logical circuit.



Note: C_1 is set to "0".



(Overflows when V = 1)

Answer group:

a) A_n, B_n b) A_n, Z_n c) B_n, Z_n

d) C_n, C_{n+1} e) C_n, Z_n f) C_{n+1}, Z_n

Q2. Read the following description about the relational database, and then answer Subquestions 1 through 3.

The following relational database consists of an employee table and an employee skill table.

Employee Table

Employee number	Employee name	Department
0001	Brown	A1
0002	Charles	A2
0003	Taylor	B1
0004	Williams	D3
0005	Parker	A1
0006	James	B1

Employee Skill Table

Employee number	Skill code	Date registered
0001	FE	19991201
0001	DB	20010701
0002	NW	19980701
0002	FE	19990701
0002	SW	20000701
0005	NW	19991201

Subquestion 1

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

True is returned for the EXISTS phrase when the sub-query result exists, and false is returned if it does not exist. When the following SQL statement is executed, the number of selected employee is $\boxed{\text{A}}$.

```
SELECT employee_number FROM employee_skill_table
    WHERE EXISTS (SELECT * FROM employee_skill_table
                    WHERE skill_code = 'FE')
```

The EXISTS phrase evaluates each respective row if the specified sub-query contains a reference to a table different from that in the main query. When the following SQL statement is executed, the number of selected employee is $\boxed{\text{B}}$.

```
SELECT employee_name FROM employee_table A
    WHERE EXISTS (SELECT * FROM employee_skill_table B
                    WHERE skill_code = 'FE'
                    AND A.employee_number = B.employee_number)
```

Answer group for A and B:

a) 0	b) 1	c) 2	d) 3
e) 4	f) 5	g) 6	

Subquestion 2

The following SQL statement outputs some employees' employee numbers. Who will be selected? From the answer group below, select the correct answer.

```
SELECT DISTINCT employee_number FROM employee_skill_table B1
    WHERE EXISTS (SELECT * FROM employee_skill_table B2
                WHERE B1.Employee_number = B2.Employee_number
                AND B1.skill_code <> B2.skill_code)
```

Answer group:

a) Employees who have at least one skill.
b) Employees who have only one skill.
c) Employees who have no skills.
d) Employees who have multiple skills.

Subquestion 3

You want to add the employee's name to the information obtained in Subquestion 2. From the answer group below, select the correct answer to be inserted in the blank $\boxed{}$ in the following SQL statement.

```
SELECT DISTINCT A.employee_number, employee_name
            FROM employee_table A, employee_skill_table B1
    WHERE EXISTS (SELECT * FROM employee_skill_table B2
                WHERE B1.employee_number = B2.employee_number
                AND B1.skill_code <> B2.skill_code)
        [                                    ]
```

Answer group:

a) AND A.employee_number = B1.employee_number
b) BETWEEN A.employee_number AND B1.employee_number
c) LIKE A.employee_number = B1.employee_number
d) OR A.employee_number = B1.employee_number

Q3. Read the following program description, and then answer Subquestions 1 through 3.

[Program description]

This program is an implementation of a text editor.

(1) This text editor can handle up to 80 characters of text per line and up to `MAX` lines.

(2) The following global variable is defined.

(i)	`CP`:	This variable indicates the line to be processed.

(3) The following sub-programs are defined.

(i)	`INSERT(x)`:	Inserts a new line `x` at the line indicated by `CP`. Insertion causes the current and subsequent lines to be moved downward by one line.
(ii)	`DELETE()`:	Deletes the line indicated by `CP`. Deletion causes the next and subsequent lines to be moved upward by one line.
(iii)	`LAST()`:	Returns the line number corresponding to the last line (the number of lines in the text). If the text is empty, then "0" is returned.
(iv)	`GET()`:	Returns the character string on the line indicated by `CP`.

Subquestion 1

A character string type array `LINE[i]` (`i` = 1, 2, ..., `MAX`) and integer type variable `TAIL` are defined in the program. The character string L_i on Line `i` is stored in `LINE[i]` and an index of the array element ("0" if empty), which always corresponds to the last line, is stored in the variable `TAIL`.

Variable `CP` contains an index of the array element of the line to be processed. From the answer group below, select the sub-program(s) for which the amount (order) of computation needed remain constant, regardless of the number of lines (select **all** applicable answers).

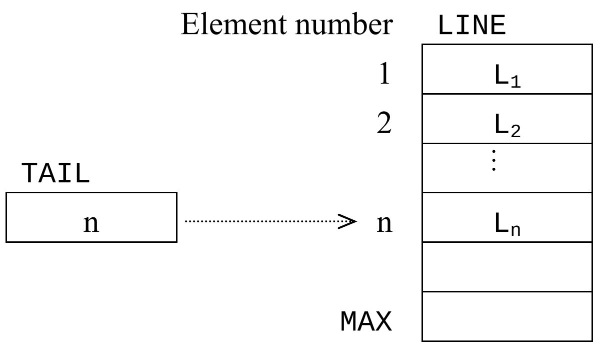


Fig. 1 Example: Implementation Using an Array

Answer group:

a) `DELETE ( )` b) `GET ( )` c) `INSERT (x)` d) `LAST ( )`

Subquestion 2

The program was changed so that it can handle a bi-directional list using pointers, as shown in Figure 2. Element `i` in the list consists of the pointer to the element storing Line `i-1`, the character string `Li` at Line `i`, and the pointer to the element storing Line `i+1`. If an applicable element does not exist, "0" is stored as the pointer value.
The variable `HEAD` is a pointer to the element which stores the first line. The variable `TAIL` is a pointer to the element which stores the last line.
The variable `CP` is a pointer to the element containing the line to be processed. From the answer group below, select the sub-program(s) for which the amount (order) of computation needed remain constant, regardless of the number of lines (select **all** applicable answers).

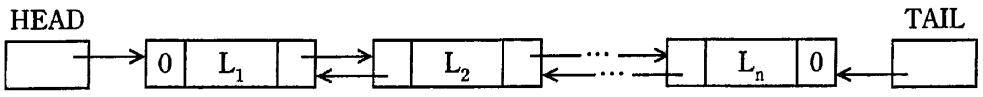


Fig. 2 Example: Implementation Using Pointers

Answer group:

a) `DELETE ( )` b) `GET ( )` c) `INSERT (x)` d) `LAST ( )`

Subquestion 3

From the answer group below, select the correct answers to be inserted in the blanks $\boxed{}$ in the following text.

A bi-directional list using pointers was implemented using three arrays. Figure 3 is an example of a case in which the maximum number of lines is 10. The index of the array element containing the line to be processed is stored in the variable `CP`. The index of the array element containing the first line is stored in the variable `HEAD`. The index of the array element containing the last line is stored in the variable `TAIL`. The index of the first array element in the empty list is stored in the variable `EMPTY`.
Assume that Lines L_1, L_2, L_3, L_4, and L_5 are stored as shown in Figure 3 and `CP` = 8. If `DELETE()` is executed to delete the array element containing L_3, then the values of `HEAD` and `TAIL` are not changed and `EMPTY` = 8, `PREV[9]` = $\boxed{A}$, `NEXT[2]` = $\boxed{B}$, and `NEXT[8]` = $\boxed{C}$. Assume that the deleted array element is added to the beginning of the empty list.

CP	HEAD	TAIL	EMPTY
8	4	6	3

Element number	PREV	LINE	NEXT
1			0
2	4	L_2	8
3			5
4	0	L_1	2
5			7
6	9	L_5	0
7			10
8	2	L_3	9
9	8	L_4	6
10			1

Fig. 3 Example: Implementation with a Bi-Directional List Using Arrays

Answer group:

a) 0	b) 1	c) 2	d) 3
e) 7	f) 8	g) 9	h) 10

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

[Program Description]

The subprogram `HeapSort` is a program to sort integer values that are stored in an array in ascending order by heapsorting.

(1) The `Num` items of integers (`Num` >= 2) to be sorted are stored in an array of global variables `A[1]`, `A[2]`, ... , `A[Num]`.

(2) The heap sort uses a binary tree to sort data. In order to represent a binary tree with an array, when a certain node corresponds to `A[i]`, the node for the left child corresponds to `A[2*i]` and the node for the right child corresponds to `A[2*i+1]`. In the figure below, the circles represent the nodes, the numbers inside the circles represent the node values, and the actual array elements that store the values are shown next to the circles.

(3) As shown in the figure, the heap is a binary tree in which the value of each node is greater than or equal to the values of its children.

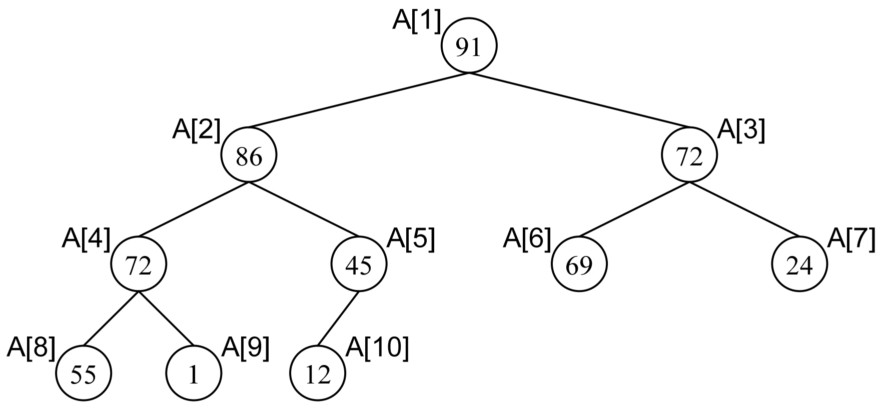


Fig. Example of a Heap

(4) The procedure for sorting is as follows.

(i) Elements `A[1]`, `A[2]`, ... , `A[Num]` of array `A` are the elements to be sorted.

(ii) Each element to be sorted represents a binary tree described in (2), and the heap is created by swapping the values of the elements. As a result, the largest value of the elements to be sorted is placed at the root of the tree (`A[1]`).

(iii) Swap the value at the root of the tree (`A[1]`) and the value at the last node of the tree (the node corresponding to the last element to be sorted).

(iv) The last node of the tree is removed from the binary tree. (The number of elements to be sorted is reduced by 1.)

(v) The processing of Steps (ii) through (iv) is repeated until only the root of the tree remains.

(5) The procedure for reconstructing the heap is as follows.

(i) Make the root of the tree the parent node.

(ii) Terminate if there is no child node.

(iii) Compare the larger of the values of the two child nodes to that of the parent node, and swap the corresponding values if the value of the parent node is smaller. Terminate if the value of the parent node is greater than or equal to the child node.

(iv) Repeat Steps (i) through (iii) for the subtree which has as its root the child node for which the value was swapped.

(6) The specifications of the subprogram arguments are shown in Tables 1 through 4.

Table 1 Specifications of the Argument for `HeapSort`

Argument Name	Data Type	Input/ Output	Meaning
`Num`	Integer type	Input	Index of the array element corresponding to the last node of the tree

Table 2 Specifications of the Argument for `InitHeap`

Argument Name	Data Type	Input/ Output	Meaning
`Last`	Integer type	Input	Index of the array element corresponding to the last node of the tree for which the heap is first created

Table 3 Specifications of the Arguments for `MakeHeap`

Argument Name	Data Type	Input/ Output	Meaning
`Top`	Integer type	Input	Index of the array element corresponding to the root of the subtree for which the heap is reconstructed
`Last`	Integer type	Input	Index of the array element corresponding to the last node of the subtree for which the heap is reconstructed

Table 4 Specifications of the Arguments for `Swap`

Argument Name	Data Type	Input/ Output	Meaning
`X`	Integer type	Input	Index of the array element to swap with `A[Y]`
`Y`	Integer type	Input	Index of the array element to swap with `A[X]`

[Program]

```
○Integer type: A[1000000]   /* Used as a global variable */

○HeapSort(Integer type: Num)
○Integer type: Idx
                                          /* First create the heap */
▪ InitHeap(Num)
                                          /* Sort */
■ Idx: Num, Idx > 1, -1
│   ▪ Swap(1, Idx)
│   ▪ MakeHeap(1, Idx-1)
■


○MakeHeap(Integer type: Top, Integer type: Last)
○Integer type: L, R
▪ [    A    ]
▪ R ← L + 1
▲ R <= Last                             /* Compare 3 elements */
│   ▲ A[L] < A[R]                       /* The right element is bigger */
│   │   ▲ A[Top] < A[R]
│   │   │   ▪ Swap(Top, R)
│   │   │   ▪ MakeHeap(R, Last)
│   │   ▼
│   ├─────                              /* The left element is bigger */
│   │   ▲ A[Top] < A[L]
│   │   │   ▪ Swap(Top, L)
│   │   │   ▪ MakeHeap(L, Last)
│   │   ▼
│   ▼
├─────
│   ▲   [    B    ]                     /* Compare 2 elements */
│   │   ▲ A[Top] < A[L]
│   │   │   ▪ Swap(Top, L)
│   │   │   ▪ MakeHeap(L, Last)
│   │   ▼
│   ▼
▼
○Swap(Integer type: X, Integer type: Y)
○Integer type: Tmp
▪ Tmp ← A[X]
▪ A[X] ← A[Y]
▪ A[Y] ← Tmp
```

Subquestion 1

From the answer groups below, select the correct answers to be inserted in the blanks $\boxed{}$ in the above program.

Answer group for A:

a) `L ← Top` b) `L ← Top + 1`
c) `L ← Top * 2` d) `L ← Top * 2 + 1`

Answer group for B:

a) `L <= Last` b) `L < Last`
c) `R <= Last - 1` d) `R <= Last - 2`

Subquestion 2

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

Using the heap in the figure, when Steps (iii) and (iv) of (4) in the section [Program Description] are executed just once and Step (ii) is completed, the index of the array element in which the value 12 (initially stored in `A[10]`) is stored is $\boxed{\text{C}}$. The number of times values in nodes were swapped up to this point is $\boxed{\text{D}}$.

Answer group:

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

Subquestion 3

The subprogram `InitHeap` that initially creates a heap can be created using `MakeHeap`. From the answer group below, select the correct answer to be inserted in the blank $\boxed{\quad\quad}$ in the following program.

```
○ InitHeap(Integer type: Last)
○ Integer type: Idx
■ $\boxed{\quad\quad}$
│  ▪ MakeHeap(Idx, Last)
■
```

Answer group:

a) `Idx: 1, Idx <= Last, 2`
b) `Idx: 1, Idx <= Last / 2, 1`
c) `Idx: Last, Idx >= 1, -2`
d) `Idx: Last / 2, Idx >= 1, -1`

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

You are going to design a program for a game that raises and lowers a flag. In the game, the player responds to instructions (raise/lower red flag and/or white flag) using the input device by operating buttons. A certain type of window is displayed when the correct response is made and another type is displayed when an incorrect response is made. This pattern is repeated only for a set number of times and the number of correct responses is displayed as the score. The game hardware configuration and the windows for correct responses and incorrect responses are shown in Figure 1 below.

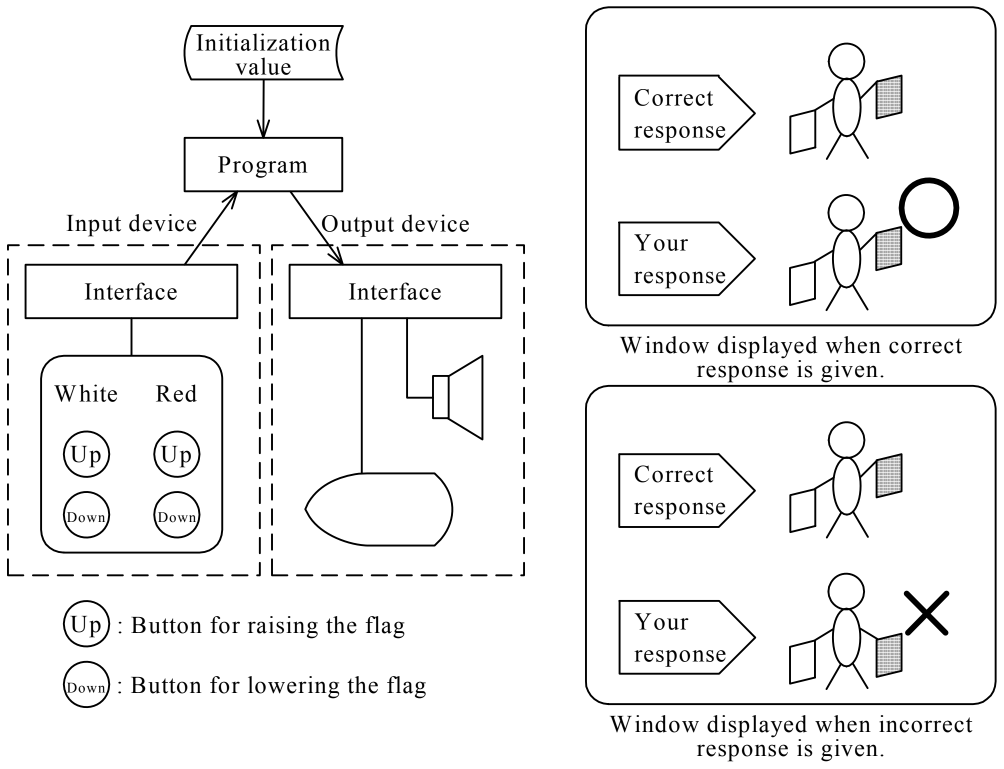

Initialization value
Program
Input device
Output device
Interface
Interface
White
Red
Up
Up
Down
Down
Up : Button for raising the flag
Down : Button for lowering the flag
Correct response
Your response
Window displayed when correct response is given.
Correct response
Your response
Window displayed when incorrect response is given.

Fig. 1 Game Hardware Configuration and Correct/Incorrect Response Display

- The output device gives an instruction by voice and displays the correct response and the player's response after the response detection time has elapsed. After the game ends, the score is displayed.
- The input device detects only the first button pressed after the instruction has been given and notifies the program of this button.

[Explanation of Program]

(1) The program internally holds the current status of the flag.

Red flag status:	Up or Down
White flag status:	Up or Down

(2) The program reads the initial value file and sets the initial status of both the red flag and white flag to “Down”.

Initial value file format

Response detection time	Incorrect response window display time	Number of repetitions

(3) The program randomly selects a flag for the raise/lower instruction and a selectable instruction, based on the status of the flags given in the table, and then sends that information to the output device.

Table: Flag Status and Selectable Instructions

Flag status	Selectable instructions	
Up	Lower	Do not lower
Down	Raise	Do not raise

(4) There are 5 types of operations which can be received from the input device: raise the red flag, lower the red flag, raise the white flag, lower the white flag, and move neither flag (when a response is not detected within the response detection time).

(5) The program detects the player’s response. If correct, it adds “1” to the number of correct responses and sends the correct response window to the output device. If incorrect, it sends the incorrect response window to the output device and, after the incorrect window display time in the initial value file has elapsed, it sets the flag in the window to the correct state.

(6) The programs repeats steps (3) through (5) the number of repetitions specified in the initial value file.

(7) The program sends the player’s score to the output device.

Subquestion 1

Figure 2 below is a state transition diagram of the flags. From the answer groups below, select the correct answers to be inserted in the blanks $\boxed{}$ in the following description about Figure 2.

S0 indicates the initial status and the "raise the red flag" response corresponds to state transition (i) from S0 to S1. In this case, S1 is the $\boxed{\text{A}}$ state. And, the response corresponding to state transition (ii) of S3 to S0 is $\boxed{\text{B}}$ and that for state transition (iii) of S3 to S2 is $\boxed{\text{C}}$.

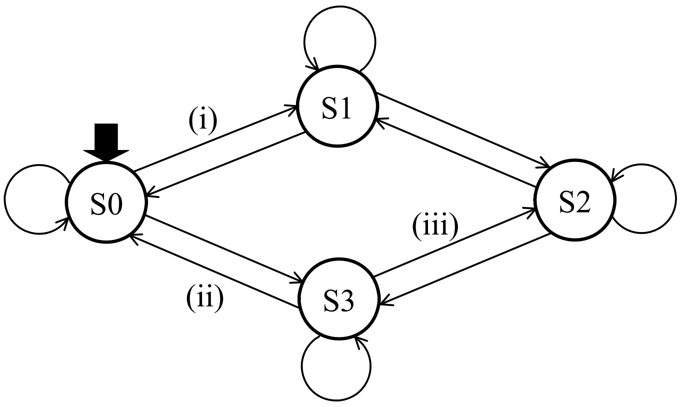


Fig. 2 State Transition Diagram of Flags

Answer group for A:

a) red flag up and white flag up
b) red flag up and white flag down
c) red flag down and white flag up
d) red flag down and white flag down

Answer group for B and C:

a) raise the red flag
b) lower the red flag
c) raise the white flag
d) lower the white flag

Subquestion 2

Figure 3 below is a flowchart of this program. The initial status is set in the initialization processing. Other than the initialization processing and the setting of the number of repetitions, what **two** processes reference the content of the initial value file?

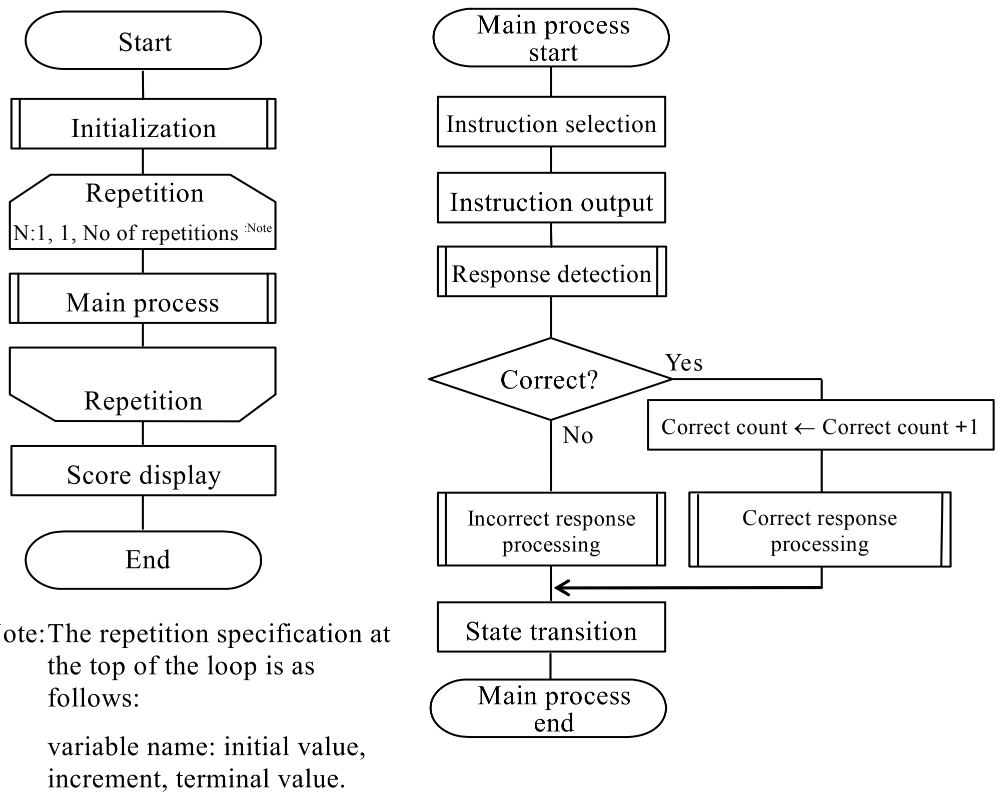

Start
Initialization
Repetition
N:1, 1, No of repetitions :Note
Main process
Repetition
Score display
End
Note: The repetition specification at the top of the loop is as follows:
variable name: initial value, increment, terminal value.
Main process start
Instruction selection
Instruction output
Response detection
Correct?
Yes
No
Correct count ← Correct count +1
Incorrect response processing
Correct response processing
State transition
Main process end

Fig. 3 Flowchart

Answer group:

a) Incorrect response processing
b) Instruction output
c) Instruction selection
d) State transition
e) Score display
f) Correct response processing
g) Response detection

Select one question from **Q6** and **Q7.** If **two questions** are selected, **only the first question** will be graded.

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

[Program Description]

This program parses a tagged character string, and picks up tag values into separate elements.

(1) The syntax for tagged character strings is defined as follows. Symbols used in syntax notation are defined as given in Table 1 below. In addition, <, >, and / are used as tokens.

Table 1 Meanings of Symbols Used in Syntax Notation

Symbol	Meaning
::=	Used for definitions
|	Or
{ }*	The syntax elements contained inside "{" and "}" are repeated 0 or more times.

tagged_character_string ::= {tagged_structure}*

tagged_structure ::= start_tag tag_value {tagged_structure}* end_tag

start_tag ::= <tag_name>

end_tag ::= </tag_name>

tag_name ::= character string that does not include <, >, / or "\0"

tag_value ::= character string that does not include <, >, / or "\0" | a null character string

character_string ::= {character}*

(2) Program specifications are as follows:

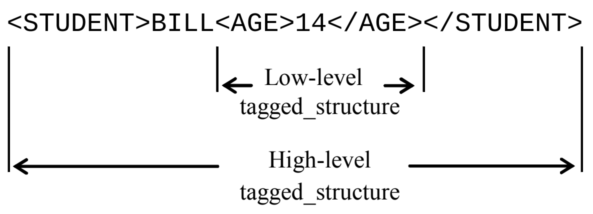
(i) The function `parse_ml_string` decomposes a given tagged_character_string `mlstr` into elements and stores them in the element array `elmtbl` in order of the appearance of their start_tags. The number of elements is stored in a variable `elmnum`. Here, an element is represented by the following data structure:

```
typedef struct {
    char *tag;    /* Tag_name */
    int  depth;   /* Depth of nesting (1, 2,...) */
    char *value;  /* Tag_value */
} ELEMENT;
```

(ii) The number of tag elements contained in a tagged_character_string does not exceed 256.

(iii) There are no syntax errors in a given tagged_character_string.

(3) The tagged_structure may include another tagged_structure. (See Figure 1 below.)



<STUDENT>BILL<AGE>14</AGE></STUDENT>
Low-level tagged_structure
High-level tagged_structure

Fig.1 Example of Nested Tagged Level Structures

(4) The depth of nesting is given by numeric values 1, 2, 3, ... etc., where "1" is used to indicate the highest level tagged_structure.

(5) The results of executing this program on the tagged_character_string shown in Figure 2 below are given in Table 2.

```
<STUDENT>BILL<AGE>14</AGE><SCHOOL>Junior
<PLACE>Tokyo</PLACE></SCHOOL></STUDENT>
```

Note: The character string does not contain CR/LF code.

Fig.2 Value of the Tagged Character String mlstr

Table 2 Element Array `elmtbl`

Tag name `tag`	Depth of nesting `depth`	Tag value `value`
Pointer to `STUDENT`	1	Pointer to `BILL`
Pointer to `AGE`	2	Pointer to `14`
Pointer to `SCHOOL`	2	Pointer to `Junior`
Pointer to `PLACE`	3	Pointer to `Tokyo`

[Program]

```
#define MAXELMNUM 256

typedef struct {
    char *tag;
    int  depth;
    char *value;
} ELEMENT;

char *parse_ml_data(char *, int);
ELEMENT elmtbl[MAXELMNUM];
int elmnum = 0;

void parse_ml_string(char *mlstr)
{
    while (*mlstr != '\0') {
        mlstr = parse_ml_data(mlstr + 1, 1);
    }
}

char *parse_ml_data(char *mlstr, int level)
{
    /* Start_tag processing */
    elmtbl[elmnum].tag = [    A    ] ;
    elmtbl[elmnum].depth = level;
    for (; *mlstr != '>'; mlstr++);
    *mlstr = '\0';
    /* Tag_value processing */
    elmtbl[elmnum].value = [    B    ] ;
    for (mlstr++; *mlstr != '<'; mlstr++);
    *mlstr = '\0';
    [    C    ] ;
    /* Low-level tag_structure processing */
    while ( [    D    ] )
        mlstr = parse_ml_data(mlstr + 1, level + 1);
    /* End_tag processing */
    for (mlstr += 2; *mlstr != '>'; mlstr++);
    return mlstr + 1;
}
```

Subquestion

From the answer groups below, select the correct answers to be inserted in the blanks $\boxed{\qquad}$ in the above program.

Answer group for A and B:

a) `mlstr`	b) `++mlstr`	c) `mlstr + 1`
d) `*mlstr`	e) `*(++mlstr)`	f) `*mlstr++`
g) `*(mlstr + 1)`		

Answer group for C:

a) `elmnum++`	b) `level++`	c) `mlstr++`
d) `*elmnum++`	e) `*level++`	f) `*mlstr++`

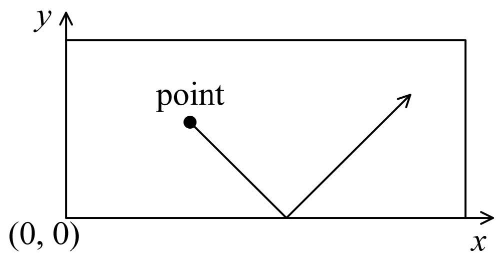
Answer group for D:

a) `*(mlstr + 1) == ‘<’`	b) `*(mlstr + 1) != ‘<’`
c) `*(mlstr + 1) == ‘/’`	d) `*(mlstr + 1) != ‘/’`
e) `*(mlstr + 1) == ‘>’`	f) `*(mlstr + 1) != ‘>’`

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

[Program Description]

This program draws a point that moves within a rectangular space as shown below.



The point is represented with class `Point`, and this class stores the coordinates (*x*, *y*) that represent the position of the point and the speed. The speed is a positive value and represents the distance moved in the *x*-axis direction and *y*-axis direction per unit time.
The rectangular space in the figure is given by class `Space`, and the following class methods can be called.

(1) `public static int getMaxX()`
This method returns the maximum value (a positive value) of the *x* coordinate in the rectangular space.

(2) `public static int getMaxY()`
This method returns the maximum value (a positive value) of the *y* coordinate in the rectangular space.

(3) `public static void draw(Point)`
This method draws the point specified by the argument at the coordinates of the point.

(4) `public static void erase(Point)`
This method erases the point specified by the argument.

The abstract class `Motion` uses the point given by `Point` as its initial value within the constructor, and repeatedly draws, moves and erases the point in this order to display the movement of the point. The coordinates of the point after moving is given by the method `update`.
`SimpleMotion` is the subclass of `Motion`, and it implements methods `update` and `main`. Method `update` expresses, within a rectangular space, the movement of a point that moves in a straight line at a fixed speed, and bounces off the sides of the rectangle. Method `main` tests the program.
Note that it is assumed that the initial values of the coordinates of `Point` generated by method `main` are within the rectangular space, and collisions between points do not need to be taken into account.

[Program 1]

```
public class Point {
    private int x, y, speed;
    public Point(int x, int y, int speed) {
        this.x = x; this.y = y;
        this.speed = speed;
    }
    public int getX() { return x; }
    public int getY() { return y; }
    public int getSpeed() { return speed; }
}
```

[Program 2]

```
public abstract class Motion implements Runnable {
    private Point point;

    public Motion(Point point) {
        this.point = point;
    }

    public void run() {
        while (true) {
            Space.draw(point);
            try {
                Thread.sleep(40);
            } catch (InterruptedException e) {}
            Point current = point;
            [           A           ];
            Space.erase(current);
        }
    }

    public abstract Point update(Point point);
}
```

[Program 3]

```
public class SimpleMotion extends Motion {
    private int directionX = 1, directionY = 1;

    public SimpleMotion(Point point) {
        super(point);
    }

    public Point update(Point point) {
        int speed = point.getSpeed();
        int x = point.getX() + directionX * speed;
        int y = point.getY() + directionY * speed;
        if (x <= 0) {
            x = -x;
            directionX *= -1;
        }
        x %= 2 * Space.getMaxX();
        if (x >= Space.getMaxX()) {
            x = 2 * Space.getMaxX() - x;
            directionX *= -1;
        }
        if (y <= 0) {
            y = -y;
            directionY *= -1;
        }
        y %= 2 * Space.getMaxY();
        if (y >= Space.getMaxY()) {
            y = 2 * Space.getMaxY() - y;
            directionY *= -1;
        }
        return new Point(x, y, speed);
    }

    public static void main(String[] args) {
        Point[] points = {
            new Point(10,  20, 3),
            new Point(50,  10, 5),
            new Point(150, 60, 2)
        };
        for (int i = 0; i < points.length; i++) {
            new Thread(            B            ).start();
        }
    }
}
```

Subquestion 1

From the answer groups below, select the correct answers to be inserted in the blanks $\boxed{}$ in the above programs.

Answer group for A:

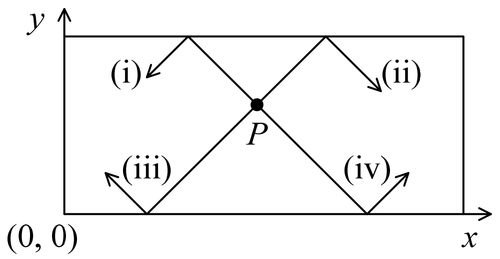
a) `current = update(current)` b) `current = update(point)`
c) `point = update(point)` d) `update(current)`
e) `update(point)`

Answer group for B:

a) `new Motion(points[i])` b) `new Point(points[i])`
c) `new Runnable(points[i])` d) `new SimpleMotion(points[i])`
e) `points[i]`

Subquestion 2

From the answer group below, select the correct answer for the movement of point *P* in the following figure when method `run` is executed. Assume that point *P* is what is given as `Point` to the constructor of class `SimpleMotion`, and that the value of `speed` is 1.
Also assume that all the $\boxed{}$ in the program have the correct answers.



Answer group:

a) The point moves as shown by arrow (i).
b) The point moves as shown by arrow (ii).
c) The point moves as shown by arrow (iii).
d) The point moves as shown by arrow (iv).

Select one question from **Q8** and **Q9**. If **two questions** are selected, **only the first question** will be graded.

Q8. Read the description of the following C program and the program itself, and then answer Subquestions 1 through 4.

[Program Description]

This program finds the path through a maze.

(1) Data for a maze represented on an 8-row by 8-column square, such as shown in Figure 1 below, is stored in a two-dimensional array `M`. Squares which are colored gray represent walls, while squares which are white represent the path.

(2) Array `M` stores a code to indicate the start point (0xf0), assigned to the entrance square, a code to indicate the end point (0xf1), assigned to the exit square, a code to indicate walls (0x00), assigned to the wall squares, and a code to indicate the path (0xff), assigned to the path squares.

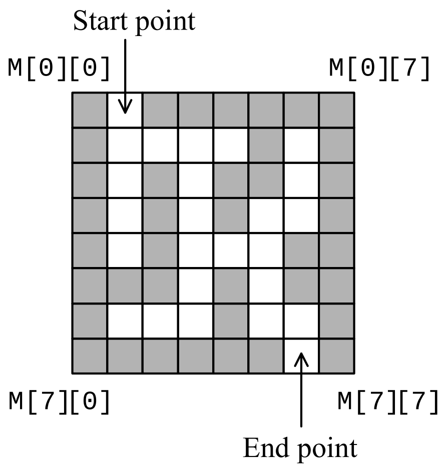


Fig. 1 Example of Maze

(3) Except for the start point and end point, all squares on the outside edge of the maze (squares whose row or column value within `M` is either "0" or "7") are walls.

(4) Global variables are used by this program as follows.

Variable name	Use
`M`	Stores the maze data (two-dimensional array)
`x`	Row of square inside maze
`y`	Column of square inside maze
`dir`	Direction of move while going through maze

Assume that initial values are loaded for these global variables.

The function `maze` searches through the maze to find the path from the entrance to the exit.

[Program]

(Line No.)

```
 1 #define   UP          0
 2 #define   RIGHT       1
 3 #define   DOWN        2
 4 #define   LEFT        3
 5 #define   ROAD        0x00        /*  Path code  */
 6 #define   WALL        0xff        /*  Wall code  */
 7 #define   SMAX        8
 8 #define   ENTRANCE    0xf0        /*  Start point code  */
 9 #define   EXIT        0xf1        /*  End point code  */
10
11 int rcheck(void);
12 int fcheck(void);
13 void go(void);
14 void maze(void);
15
16 int M[SMAX][SMAX], x, y, dir;
17
18 void maze()
19 {
20      while ( M[y][x] != EXIT ) {
21          if ( ( rcheck() == ROAD ) ||
22              ( rcheck() == EXIT ) ) {
23              dir = ( dir+1 ) % 4;
24              go();
25          }
26          else if ( ( fcheck() == ROAD ) ||
27                   ( fcheck() == EXIT ) ) go();
28          else dir = ( dir+3 ) % 4;
29      }
30      return;
31 }
32
33 int rcheck()
34 {
35      if      ( dir == UP )      return M[y][x+1];
36      else if ( dir == RIGHT )   return M[y+1][x];
37      else if ( dir == DOWN )    return M[y][x-1];
38      else                       return M[y-1][x];
39 }
40
41 int fcheck()
42 {
43      if      ( dir == UP )      return M[y-1][x];
44      else if ( dir == RIGHT )   return M[y][x+1];
45      else if ( dir == DOWN )    return M[y+1][x];
46      else                       return M[y][x-1];
47 }
48
```

```
void go()
{
     if      ( dir == UP )      y--;
     else if ( dir == RIGHT )   x++;
     else if ( dir == DOWN )    y++;
     else                       x--;
}
```

Subquestion 1

Given the maze shown in Figure 2 below, what is the path output by this program? Select the correct answer from the answer group below. *1* through *8* shown in the figure are used to represent the location of squares in the maze. In this program, the value of the global variables `x`, `y`, and `dir` are as follows at the time the function maze is called:

```
x = 1
y = 0
dir = DOWN
```

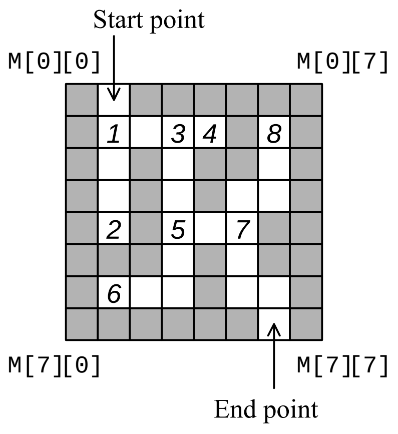


Fig. 2 Locations of Squares in Maze

Answer group:

a) Start point→*1*→*2*→*1*→*3*→*5*→*6*→*5*→*7*→End point
b) Start point→*1*→*2*→*1*→*3*→*4*→*3*→*5*→*6*→*5*→*7*→*8*→*7*→End point
c) Start point→*1*→*3*→*5*→*7*→End point
d) Start point→*1*→*3*→*4*→*3*→*5*→*6*→*5*→*7*→*8*→*7*→End point

Subquestion 2

Given the maze shown in Figure 3 below, what is the path output by this program? Select the correct answer from the answer group below. In this program, the value of the global variables `x`, `y`, and `dir` are as follows at the time the function `maze` is called.

```
x = 1
y = 0
dir = DOWN
```

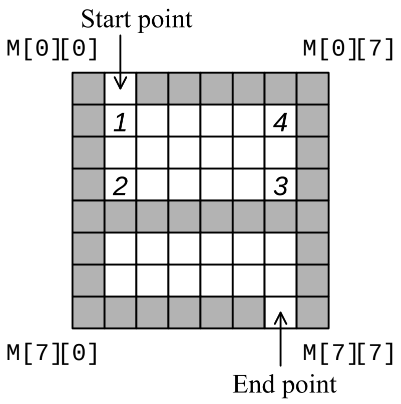


Fig. 3 Unsolvable Maze

Answer group:

a) Start point→*1*→*4*→*3*→*2*→*1*→*4*→*3*→*2*→... are repeated indefinitely.
b) Start point→*1*→*2*→*3*→*4*→*1*→*2*→*3*→*4*→… are repeated indefinitely.
c) Start point→*1*→*2*→*2*→*2*→... are repeated indefinitely.
d) Start point→*1*→*2*→*3*→*4*→*1*→Start point (Program ends)
e) Start point→*1*→*2*→*3*→*4*→*1*→*1*→*1*→… are repeated indefinitely.

Subquestion 3

The following statement was added to the program to output the positions of the squares, including the start points and end points, being passed through and the directions of the moves. What is the best place to add this statement? Select the correct answer from the answer group below.
Assume that `#include <stdio.h>` is placed at the top of the program.

```
printf("dir=%d y=%d x=%d\n", dir, y, x);
```

Answer group:

a) Immediately after Line 20 and immediately after Line 29
b) Immediately after Line 23 and immediately after Line 29
c) Immediately after Line 28 and immediately after Line 29

Subquestion 4

Assume that the following function `lcheck` is used in place of `rcheck` to find maze solutions, and that the function `lcheck` replaces `rcheck` on Lines 11, 21, and 22.
If this is done, what other changes must be made to the program? Select the correct answer from the answer group below.

```
int lcheck()
{
    if      ( dir == UP )       return M[y][x-1];
    else if ( dir == RIGHT )return M[y-1][x];
    else if ( dir == DOWN ) return M[y][x+1];
    else                    return M[y+1][x];
}
```

Answer group:

a) `== ROAD` on Line 21 must be changed to `== WALL`.
b) `+1` on Line 23 must be changed to `+3` and `+3` on Line 28 must be changed to `+1`.
c) `UP` on Line 35 and 43 must be changed to `RIGHT`, `RIGHT` on Line 36 and 44 must be changed to `DOWN`, and `DOWN` on Line 37 and 45 must be changed to `LEFT`.
d) All instances of `--` from Line 51 to Line 54 must be changed to `++` and all instances of `++` found on the same lines must be changed to `--`.

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

[Program Description]

This is a program for an electronic calculator that performs addition, subtraction, multiplication, and division operations on integers. The I/O component is supplied by a test program, and can be used to test the main calculator component program.

(1) Class `CalculatorEvent` is an event that is generated when a calculator key is pressed. The value of the field `type` represents events. Types are either `DIGIT`, `OPERATOR`, or `CLEAR`, and each represent either the number keys (0 through 9) on the calculator, operation (e.g., +) or the equal (=) keys, or the Clear key (C) respectively. When the type is `DIGIT`, the numerical value corresponding to the number key is stored in the field `value`. When the type is `OPERATOR`, the character representing the type of operation or '=' is stored in the field `value`. When the type is `CLEAR`, `value` is not used.

(2) The interface `CalculatorOutput` declares the method `display` that displays numerical values and errors on the calculator.

(3) The class `Calculator` is the main calculator itself.
Method `eventDispatched` receives events, and performs operations, etc., in accordance with the event type.
Note that the results of the operations—addition, subtraction, multiplication, and division—of two numerical values match the results of the same operations on Java's `int` type.

(4) Class `CalculatorTest` is a program to test `Calculator`.
`CalculatorOutput` is implemented as an anonymous class. In this implementation, method `display` outputs the numerical value or character string specified by `System.out`. Method `main` generates `CalculatorEvent` from the character string given by the argument `args[0]`, and calls method `eventDispatched` of `Calculator`.
The correspondences between the characters and calculator keys are as shown in the following table.

Character	Calculator key
'0' to '9'	Number key (0 to 9)
'+'	Addition key (+)
'-'	Subtraction key (-)
'*'	Multiplication key (×)
'/'	Division key (÷)
'='	Equal key (=)
'C'	Clear key (C)

For example, the character string “2+7=” represents the calculator keys 2, +, 7, and = being pressed, in that order. When the character string is passed to the method `main` as argument `args[0]`, the program outputs the following:

```
2
2
7
9
```

[Program 1]

```
public class CalculatorEvent {
    public static final int DIGIT = 1;
    public static final int OPERATOR = 2;
    public static final int CLEAR = 3;

    private int type, value;

    public CalculatorEvent(int type) {
        [           A            ];
    }
    public CalculatorEvent(int type, int value) {
        if (type < DIGIT || type > CLEAR)
            throw new IllegalArgumentException();
        this.type = type; this.value = value;
    }
    public int getType() { return type; }
    public int getValue() { return value; }
}
```

[Program 2]

```
public interface CalculatorOutput {
    public void display(int value);
    public void display(String value);
}
```

[Program 3]

```
public class Calculator {
    private int accumulator = 0, register = 0;
    private int operator = 0;
    private CalculatorOutput output;

    public Calculator(CalculatorOutput output) {
        this.output = output;
    }

    public void eventDispatched(CalculatorEvent event) {
        switch (event.getType()) {
        case CalculatorEvent.DIGIT:
            if (operator == '=') {
                register = 0; operator = 0;
            }
            register = register * 10 + event.getValue();
            output.display(register);
            break;
        case CalculatorEvent.OPERATOR:
            try {
                register = calculate();
                output.display(register);
                accumulator = register;
                operator = event.getValue();
            } catch (ArithmeticException e) {
                output.display("Error");
                accumulator = 0; operator = 0;
            }
            if (operator != '=')
                register = 0;
            break;
        case CalculatorEvent.CLEAR:
            register = 0;
            accumulator = 0;
            operator = 0;
            output.display(register);
            break;
        }
    }
    private int calculate() {
        switch (operator) {
        case '+':
            return accumulator + register;
        case '-':
            return accumulator - register;
        case '*':
            return accumulator * register;
        case '/':
            return accumulator / register;
        }
        return register;
    }
}
```

[Program 4]

```
public class CalculatorTest {
    public static void main(String[] args) {
        Calculator calc = new Calculator(
            [          B          ] {
                public void display(int value) {
                    System.out.println(value);
                }
                public void display(String value) {
                    System.out.println(value);
                }
            });
        String keys = args[0];
        for (int i = 0; i < keys.length(); i++) {
            char c = keys.charAt(i);
            CalculatorEvent event = null;
            if (c ≥ '0' && c ≤ '9') {
                event = new CalculatorEvent(
                                [          C          ]);
            } else if (c == '=' || c == '+' || c == '-'
                        || c == '*' || c == '/') {
                event = new CalculatorEvent(
                        CalculatorEvent.OPERATOR, c);
            } else if (c == 'C') {
                event = new CalculatorEvent(
                        CalculatorEvent.CLEAR);
            }
            if (event != null)
                calc.eventDispatched(event);
        }
    }
}
```

Subquestion 1

From the answer groups below, select the correct answers to be inserted in the blanks $\boxed{}$ in the above programs.

Answer group for A:

a) `CalculatorEvent(type, 0)`
b) `new CalculatorEvent(type, 0)`
c) `return new CalculatorEvent(type, 0)`
d) `super(type, 0)`
e) `this(type, 0)`

Answer group for B:

a) `implements CalculatorOutput()`
b) `interface CalculatorOutput()`
c) `new CalculatorOutput()`
d) `new Temp() implements CalculatorOutput`
e) `public class Temp implements CalculatorOutput`

Answer group for C:

a) `c - '0', CalculatorEvent.DIGIT`
b) `c, CalculatorEvent.DIGIT`
c) `CalculatorEvent.DIGIT`
d) `CalculatorEvent.DIGIT, c`
e) `CalculatorEvent.DIGIT, c - '0'`

Subquestion 2

The following table shows the final output results when method `main` is executed with the character strings below as the arguments.

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

Assume that all the blanks $\boxed{}$ in the program have the correct answers.

Character string	Output
`3+4*5=`	35
`3*4***=`	$\boxed{\text{D}}$
`3*4=+5`	$\boxed{\text{E}}$
`3+4/0=`	$\boxed{\text{F}}$

Answer group:

a) `0`	b) `3`	c) `4`	d) `5`
e) `7`	f) `12`	g) `17`	h) `53`
i) `/ by zero`	j) `Error`		

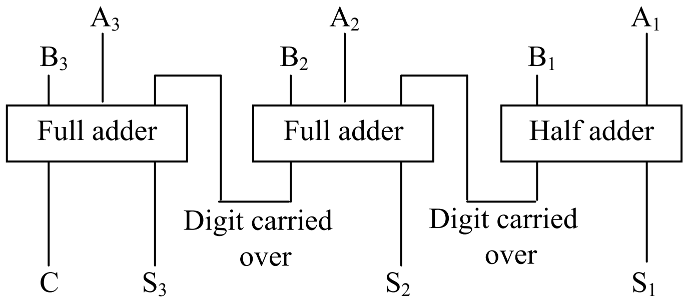
Trial Exam Answers & Comments on Afternoon Questions

Q1:

Points	➢ **A half adder has 2 input values while a full adder has 3.** ➢ **Exclusive logical sum is an operation that produces 0 when the two input values are equal.**

An adder is a circuit that adds binary bits and can be a half adder or a full adder. A half adder is a circuit that does not consider carrying from lower (less significant) digits and is used in the operation of the lowest order (least significant) digit. A full adder is a circuit that does take into account carrying from lower digits and is used in addition of digits other than the lowest digit.

Below, we show an example of a circuit that performs addition in 3 digits. Ai (i = 1, 2, 3) and Bi (i = 1, 2, 3) are binary numerals that are addends. Si (i = 1, 2, 3) are the result of addition in each digit, and C is the binary digit carried over as the highest binary digit (bit).



$$\begin{array}{r} A_3A_2A_1 \\ +)\ B_3B_2B_1 \\ \hline CS_3S_2S_1 \end{array}$$

Subquestion 1: [Correct Answer] c

C_k (= 1), A_k (= 0), and B_k (= 1) are added together, and the result of this addition is Z_k with a digit C_{k+1} carried over. The addition is done as shown below:

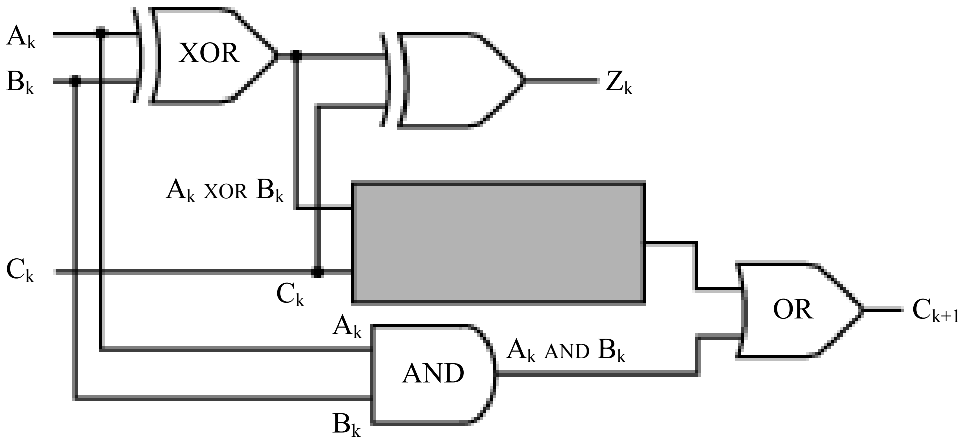
$$\begin{array}{r} 1 = C_k \\ \underline{+)\ 0\ } = A_k \\ 1 \end{array} \quad \longrightarrow \quad \begin{array}{r} 1 = C_k + A_k \\ \underline{+)\ 1\ } = B_k \\ \underset{C_{k+1}\ \ Z_k}{10} \end{array}$$

Because C_{k+1} = 1 and Z_k = 0, we have the following:

Input			Output	
C_k	A_k	B_k	C_{k+1}	Z_k
1	0	1	1	0

Subquestion 2: [Correct Answer] a

The operation for obtaining C_{k+1} is the following:



As this figure shows, if we let P be the result of the logical operation in the shaded box, the result of the logical sum (OR) of P and "A_k AND B_k" is C_{k+1}. In addition, P is the result of some binary operation involving "A_k XOR B_k" and C_k. From the answer group, it is clear that this operation is the logical product (AND), logical sum, or exclusive logical sum (XOR). So we can organize all this information as shown below.

C_k	A_k	B_k	P	A_k AND B_k	(P OR A_k AND B_k) = C_{k+1}
0	0	0	?	0	0
0	0	1	?	0	0
0	1	0	?	0	0
0	1	1	?	1	1
1	0	0	?	0	0
1	0	1	?	0	1
1	1	0	?	0	1
1	1	1	?	1	1

Here, the logical sum of P and "A_k AND B_k" is C_{k+1}. So we can "estimate" what the value of P is as follows. Since the operation here is the logical sum, if C_{k+1} is "0," both arguments (inputs) would have to be "0." But if C_{k+1} is 1, then either both arguments are "1" or one of the arguments is "1."

A_k AND B_k	$C_k + 1$	Estimated value of P
0	0	0
0	0	0
0	0	0
1	1	1, 0
0	0	0
0	1	1
0	1	1
1	1	1, 0

Now, from the estimated values of P, we can estimate the operation of "A_k XOR B_k" and C_k. Again, the answer group offers only the logical product, logical sum, and exclusive logical sum, so we limit our consideration to these.

A_k XOR B_k	C_k	P	Possible logical operation(s)
0	0	0	Logical product, Logical sum, Exclusive logical sum
1	0	0	Logical product
1	0	0	Logical product
0	0	1, 0	Logical product if this is 0. No operation produces 1.
0	1	0	Logical product
1	1	1	Logical product, Logical sum
1	1	1	Logical product, Logical sum
0	1	1, 0	Logical product if this is 0. Logical sum or exclusive logical sum if this is 1

We can therefore see that the only operation that is possible in all cases is the logical product.

Subquestion 3: [Correct Answer] d

The shaded part of the truth table has values as shown below. In the question, the index is k, but since we are only considering the highest (most significant) bit, we use the index n. The fact that the highest bit is C_n suggests that the register has n bits, causing C_{n+1} to be the overflow bit, not contained in the register.

	Input			Output	
	C_k	A_k	B_k	C_{k+1}	Z_k
Case 1	0	1	1	1	0
Case 2	1	0	0	0	1

Below, we add C_n, A_n, and B_n for each of these two cases.

(1) Case 1

$$\begin{array}{r} 0 = C_n \\ 1 = A_n \\ \underline{+)\ 1\ } = B_n \end{array}$$

$$\begin{tikzcd}[column sep=tiny] & 1 & 0 & \\ C_{n+1} \arrow[ur] & & & Z_n \arrow[ul] \end{tikzcd}$$

In this case, A_n and B_n are highest bits, both of which are 1, implying that we are adding two negative numbers. However, Z_n, the highest bit of the result of the operation, is 0, indicating a positive number. Hence, the operation is not performed correctly. This is because an overflow has occurred, pushing a “1,” which indicates a negative number, over to C_{n+1} (which has disappeared).

(2) Case 2

$$\begin{array}{r} 1 = C_n \\ 0 = A_n \\ \underline{+)\ 0\ } = B_n \end{array}$$

$$\begin{tikzcd}[column sep=tiny] & 0 & 1 & \\ C_{n+1} \arrow[ur] & & & Z_n \arrow[ul] \end{tikzcd}$$

Here, A_n and B_n are highest bits, both of which are 0, so we are adding two positive numbers. However, Z_n, the highest order bit of the result of the operation, is “1,” indicating a negative number. Hence, this operation is not performed correctly either. This is because an overflow has occurred, pushing a “0,” which indicates a positive number, over to C_{n+1} (which has disappeared).

Let us summarize the results thus far:

A_n	B_n	C_n	Z_n	C_{n+1}
0	0	1	1	0
1	1	0	0	1

Because an overflow is occurring due to digit-carrying, we focus on C_n and C_{n+1}. Then we notice that an overflow occurs whenever these two values are not equal. In the subquestion, it is stated that exclusive logical sum is used, and the exclusive logical sum of these two values C_n and C_{n+1} is 1 when they are not equal to each other. Hence, an overflow can be detected by taking the exclusive logical sum of C_n and C_{n+1}.

Q2:

Points	➢ **EXISTS returns “true” when the result of a sub-query has at least one line.** ➢ **DISTINCT eliminates duplicates.**

EXISTS in an SQL statement is executed from a sub-query (outside the SELECT statement). For one line of the result of the main query, if the result of the sub-query (the SELECT statement within parentheses) has at least one line, the value “true” is returned; if it has no lines, the value “false” is returned. If the value is “true,” the contents of the column designated by the SELECT statement in the main query are extracted.

Subquestion 1: [Correct Answer] A – g, B – c

A:
In the SELECT statement in the sub-query, the extracting condition is “skill_code = ‘FE’.” In the employee skill table, there are two lines where the skill code is “FE,” so `EXISTS` returns “true.” Hence, the SQL statement can be interpreted as follows:

`SELECT` employee_number **`FROM`** employee_skill_table **`WHERE`** (condition is true)

We thus conclude that all lines of the employee skill table do satisfy the condition, so all the employee numbers from the employee skill table will be extracted. This table has six rows.

B:
The SELECT statement in the main query picks one line from the employee table, and its result is delivered to the sub-query. In the sub-query, the contents of the delivered line are evaluated, and the result is returned to the main query. The main query then evaluates the result of the sub-query concerning the extracted row and determines whether or not to extract it.
In the sub-query, the employee table (A) and the employee skill table (B) are joined, and the rows where the skill code is “FE” get extracted. For example, the first line in the employee table (0001, Brown, A1) is joined to the employee skill table by the employee number “0001” as shown below:

Employee Table

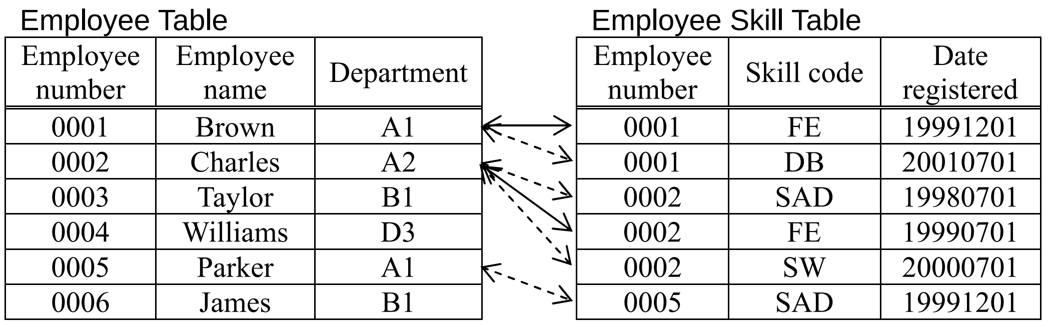
Employee number	Employee name	Department
0001	Brown	A1
0002	Charles	A2
0003	Taylor	B1
0004	Williams	D3
0005	Parker	A1
0006	James	B1

Employee Skill Table

Employee number	Skill code	Date registered
0001	FE	19991201
0001	DB	20010701
0002	SAD	19980701
0002	FE	19990701
0002	SW	20000701
0005	SAD	19991201

Now, there are two rows in the employee skill table with the same employee number “0001”, but only one of these lines has the skill code “FE”, the first row. Here, the result of the sub-query is “true” (there is a row), so the “employee name” designated in the main query gets extracted from the employee table. Similarly, the second row of the employee table (0002, Charles, A2) shares the same employee number with Rows 3, 4, and 5 of the employee skill table, but only Row 4 has the skill code “FE”. Hence, the employee name is extracted. In contrast, there are no rows in the employee skill table that share the same employee number as Rows 3, 4, or 6 of the employee table.

So these employee names are not extracted. As for the fifth row (0005, Parker, A1) of the employee table, there is a row in the employee skill table with the same employee number; however, the skill code is not "FE," so the employee name is not extracted.
Summarizing all of this, we see that two rows (Brown and Charles) are extracted as shown below. In this figure, solid lines indicate relations that are objects of extraction; dotted lines indicate that the same employee number exists but the name is not extracted because the skill code is not "FE."



Employee Table

Employee number	Employee name	Department
0001	Brown	A1
0002	Charles	A2
0003	Taylor	B1
0004	Williams	D3
0005	Parker	A1
0006	James	B1

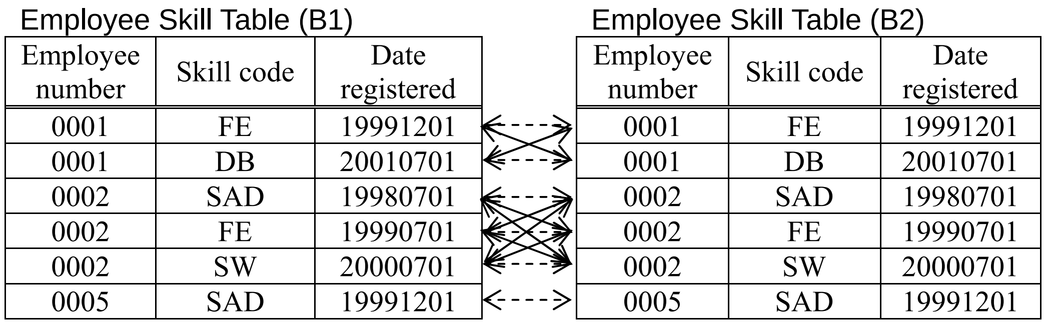
Employee Skill Table

Employee number	Skill code	Date registered
0001	FE	19991201
0001	DB	20010701
0002	SAD	19980701
0002	FE	19990701
0002	SW	20000701
0005	SAD	19991201

The idea of `EXISTS` is covered above. However, the end result here is that the employee names of the rows satisfying the sub-query condition (skill_code = 'FE') are extracted.

Subquestion 2: [Correct Answer] d

In this SQL statement, the employee skill table is defined by two names, B1 and B2. As the result, the condition of the sub-query "B1.Employee_number = B2.Employee_number" finds rows with the same employee number in the employee skill table. Further, among these rows, those with different skill codes (B1.skill_code < > B2.skill_code) are objects of extraction.
Summarizing this, we can draw a diagram as shown below. Solid lines indicate that these are extracted because the skill codes are different; dotted lines indicate that they are not extracted because the skill code is the same.



Employee Skill Table (B1)

Employee number	Skill code	Date registered
0001	FE	19991201
0001	DB	20010701
0002	SAD	19980701
0002	FE	19990701
0002	SW	20000701
0005	SAD	19991201

Employee Skill Table (B2)

Employee number	Skill code	Date registered
0001	FE	19991201
0001	DB	20010701
0002	SAD	19980701
0002	FE	19990701
0002	SW	20000701
0005	SAD	19991201

If we do not consider `DISTINCT`, because the employee numbers of rows having the solid-line correspondence are taken out of B1, what is extracted is (0001, 0001, 0002, 0002, 0002, 0002, 0002, 0002). However, the SELECT statement in the main query designates `DISTINCT`, duplicate values are eliminated, and consequently (0001, 0002) will be extracted. Therefore, the employee numbers of those with multiple skills are extracted. Incidentally, if an employee number has only one line (employee number 0005), the skill code always matches, so it will not be extracted.

Subquestion 3: [Correct Answer] a

To extract employee names, the employee table is necessary, and so it is also necessary to join the employee table and the employee skill table. The common column found in both the employee table and the employee skill table is the employee number. Hence, we join the employee numbers of the employee table (A) and the employee skill table (B1) using an AND condition. Therefore, what is to be inserted is "`AND A.`employee_number `= B1.`employee_number."

Q3:

Points	➢ **Constant amount of computation means that the amount of processing is the same in all cases.** ➢ **A bidirectional list has a forward pointer and a backward pointer.**

A global variable is a variable that can be referenced from every subprogram. A variable used in a program can be a global variable or a local variable. A local variable is a variable valid only within the program that defines it.

Subquestion 1: [Correct Answer] b, d

The requirement that the "amount (order) of computation needed" be "constant, regardless of the number of lines" means that the amount of processing does not change regardless of the order of procedures and operations.

a) `DELETE()` deletes the line designated by `CP`. By deletion, the next line and lines below that all move up by one line, so for instance the removal of Line 1 requires shifts Line 2 → Line 1, Line 3 → Line 2, … , Line n → Line (n – 1), which is (n – 1) moves. If Line 2 is removed, Line 1 stays the same, but Line 3 → Line 2, Line 4 → Line 3, … , Line n → Line (n – 1), which is (n – 2) moves. Hence, the number of moves depends on the value of n, suggesting that the amount of computation is not constant.

b) `GET()` returns the character string of the line designated by `CP`. This simply takes out the character string from the position designated by `CP`, so the amount of computation is constant.

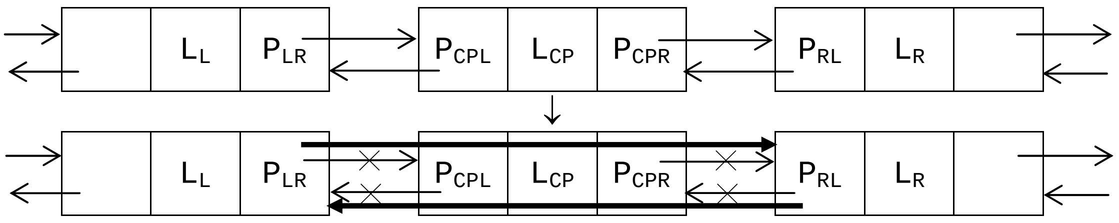
c) `INSERT()` inserts a new line x in the line designated by `CP`. By insertion, all lines below the inserted line move down by one line. For example, if a line is to be inserted before the first line, Line n → Line (n + 1), Line (n – 1) → Line n, … , Line 1 → Line 2, which is n moves, and then the insertion into Line 1 occurs. If a line is to be inserted before the second line, Line 1 stays the same, but Line n → Line (n + 1), Line (n – 1) → Line n, … , Line 2 → Line 3, which is (n – 1) moves, and then the insertion into Line 2 occurs. Thus, the number of moves depends on the value of n, suggesting that the amount of computation is not constant.

d) `LAST()` returns the line number n of the last line (the number of lines in the text). If the text is empty, this returns 0. As shown in Figure 1, this is stored in `TAIL`, so this simply extracts the value of `TAIL`. Therefore, the amount of computation is constant.

Subquestion 2: [Correct Answer] a, b, c

Given a list, deletion and addition simply change the pointer, so no manipulation of line change is necessary. In explanations below, L_{CP} indicates the line subject to manipulation (the line designated by the `CP`). Further, L_L is the line immediately before it, and L_R is the line immediately following.

a) `DELETE()` changes the pointer connection as shown below. An "×" shows a pointer that is cut, and a thick arrow indicates a pointer that is connected.



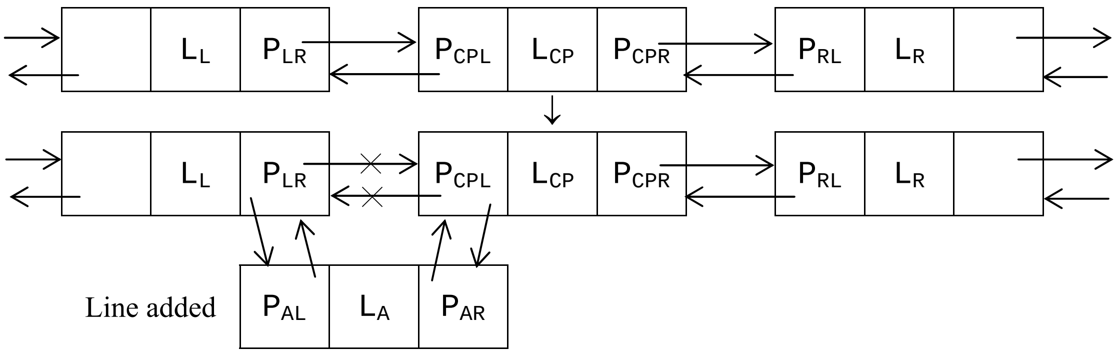
Pointer switching is carried out as follows:

(1) P_{CPR} is stored in P_{LR}. This is so that L_L points to L_R because L_{CP} is being deleted.
(2) P_{CPL} is stored in P_{RL}. This is so that L_R points to L_L because L_{CP} is being deleted.

This process only involves operations (1) and (2), so the amount of computation is constant.

b) `GET( )` simply reads LCP at the position designated by `CP`, so the amount of computation is constant.

c) `INSERT(x)` changes the pointer connection as shown below. Assume that LA is the line to be added and that the pointer of this added line is known.



Pointer switching is carried out as follows:

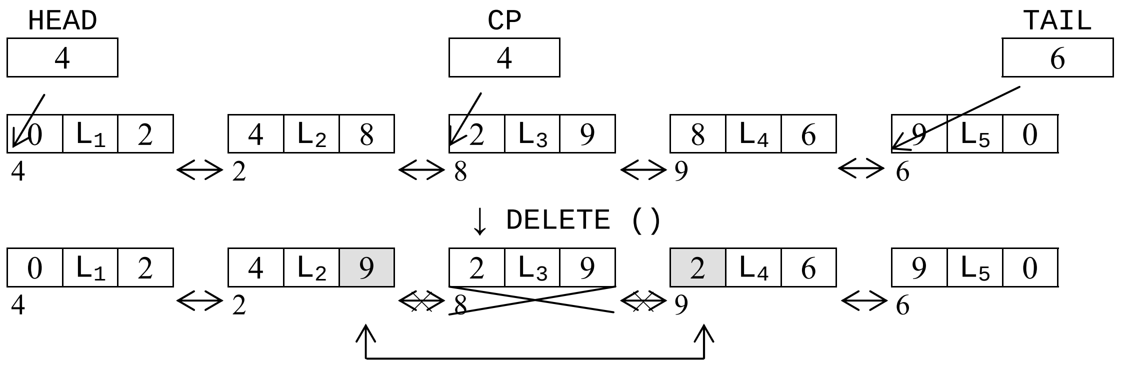
(1) P_{CPL} is stored in P_{AL}. This is so that L_A points to L_L since L_A is being added.
(2) P_{LR} is stored in P_{AR}. This is so that P_{AR} points to L_{CP} since L_A is being added.
(3) The pointer value of the line added is stored in P_{CPL}. This is so that L_{CP} points to L_A since L_A is being added.
(4) The pointer value of the line added is stored in P_{LR}. This is so that L_L points to L_A since L_A is being added.

This process only involves operations (1) through (4), so the amount of computation is constant.

d) There is no information on the number of lines anywhere, so to count the number of lines, it is necessary to actually count, beginning at `HEAD` and following pointers. Or, the number of lines can be counted from `TAIL`. Hence, `LAST()` takes longer to process when there are more lines. Therefore, the amount of computation is not constant.

Subquestion 3: [Correct Answer] A – c, B – g, C – d

The figure below shows how lines go from HEAD and TAIL. Deletion of L_3 switches the pointers as shown.



Hence, “8 is changed to 9” so that NEXT[2] points to L4, and “8 is changed to 2” so that PREV[9] points to L2.

Next, we switch the pointer of the empty list. Since the element number 8 is deleted, this becomes the head of the empty list. So EMPTY changes “from 3 to 8.” Further, we make sure that NEXT[8] of the element number 8 deleted will point to “3,” which is the head of the empty list prior to the deletion.

Q4:

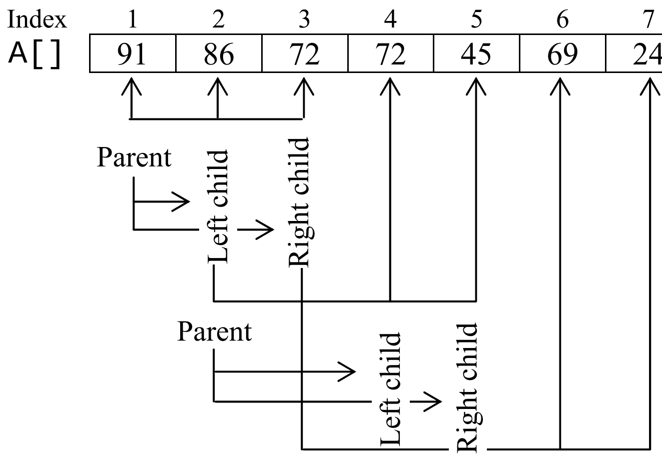
Points	➢ **In a heap, the root value is the largest, and a parent value is greater than a child value.** ➢ **HeapSort repeats root extraction and reconstruction of a heap**

A heap is a binary tree in which data are placed from shallow to deep nodes and, on the same depth level, from left to right, and the values have the following restriction:

Parent element value > child element value (or parent element value < child element value)

Hence, the elements with larger (smaller) values are gathered around the root, and those with smaller (larger) values are close to the leaves. Since the root has the element with the largest (smallest) value, this data structure is suitable for extracting the largest (smallest) value.

A heap can be represented by an array in the following way. If a node corresponds to `A[i]`, its child node on the left corresponds to `A[2×i]`, and the child node on the right corresponds to `A[2×i + 1]`. Then, an array A contains the elements in the following way:



(Notes)
If the first element is a parent, its left child has index 1 × 2 = 2.
The right child has index 1 × 2 + 1 = 3.
If the second element is a parent, its left child has index 2 × 2 = 4.
The right child has index 2 × 2 + 1 = 5.

Subquestion 1: [Correct Answer] A – c, B – a

A:

In `HeapSort`, after switching the first node and the `Idx`-th node (`Swap(1, Idx)`), `MakeHeap` is called. `MakeHeap` is a subprogram that reconstructs a heap. To do this, `MakeHeap` compares the values of parent nodes, left child nodes, and right child nodes. After obtaining the comparison results of the parent, left child, and right child, these results are once again used to set indices to call `MakeHeap` recursively.
The index of the parent node is stored as `Top`, so the indices of the left child node and the right child node must be defined by box "A" and the next statement. If the parent node has the index `TOP`, the left child has the index (`Top` × 2), and the right child has the index (`Top` × 2 + 1). The "`R ← L + 1`" right after the blank box "A" suggests that if `L` is the index of the left child node, then `R` must be the index of the right child node. Therefore, we insert "`L ← Top × 2`" here.

B:

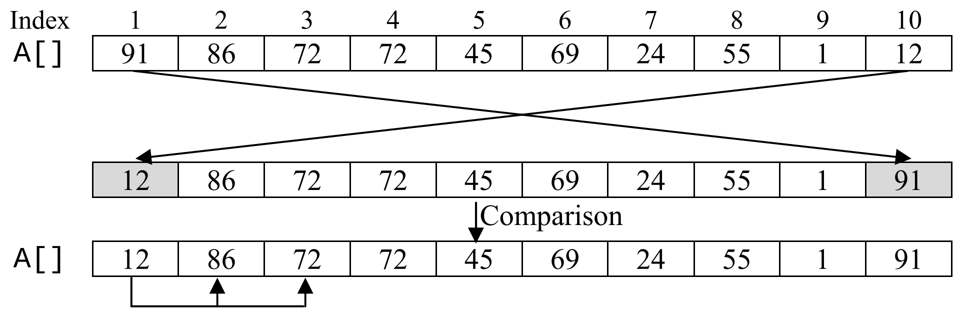
The case in which "`R ≦ Last`" does not hold is the case in which there is no right child. This is because `Last` is the index of the very last element, and `R` is the index of its right child node. In this case, we check if there is a left child. If there is, we compare the value of the parent node (whose index is `Top`) and the value of the left child (whose index is `L`). Whether or not there is a left child can be determined by "`L ≦ Last`" just as "`R ≦ Last`" used before. Hence, we insert "`L ≦ Last`" here.

Subquestion 2: [Correct Answer] C – g, D – c

According to the processes of MakeHeap, let us track how the array in the heap diagram shown in the question changes.

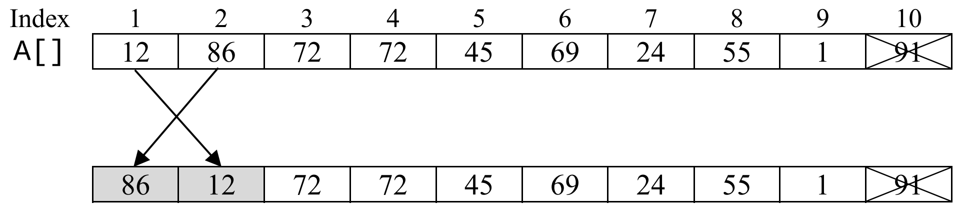
(1) First swapping

The root `A[1]` and the last node `A[10]` are switched. Then, the heap is reconstructed.

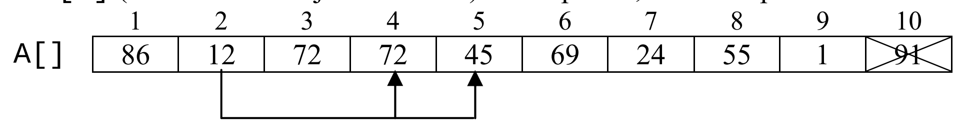


(2) Second Swapping

The value 86 (`A[2]`, the left child) is the largest value as the result of the comparisons, so `A[1]` and `A[2]` are switched.

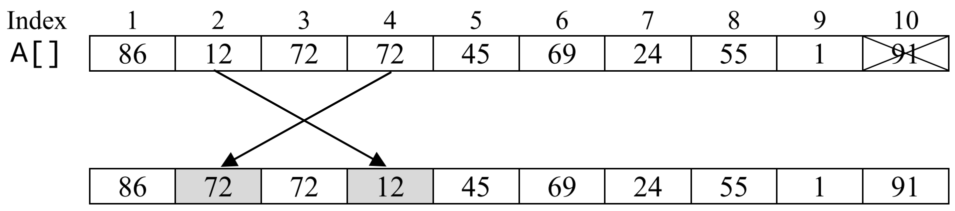


Next, with `A[2]` (which has been just switched) as the parent, the same process is executed.

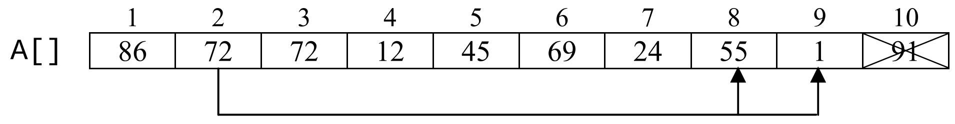


(3) Third swapping

As the result of the comparisons, 72 (`A[4]`, the left child) is the largest value, so `A[2]` and `A[4]` are switched.

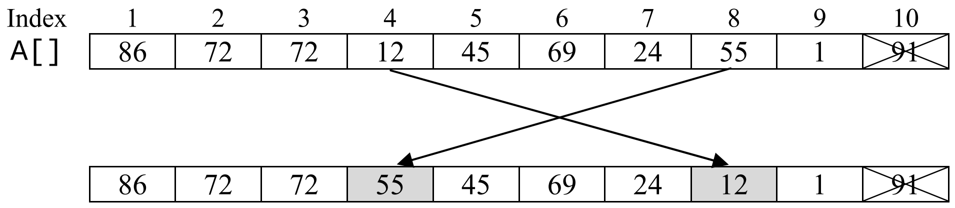


Next, with `A[4]` (which has just been switched) as the parent, the same process is executed.



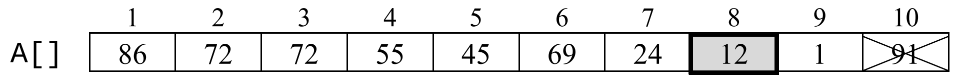
(4) Fourth swapping

As the result of the comparisons, 55 (`A[8]`, the left child) is the largest value, so `A[4]` and `A[8]` are switched.



Next, we make comparisons with `A[8]` as the parent, but there is no child node, so the process ends here. Consequently, swapping has occurred 4 times.

The final state of array A is as shown below, so the value 12 is stored as the array element whose index number is "8."



As mentioned earlier, swapping has occurred 4 times.

Subquestion 3: [Correct Answer] d

A heap is constructed by making the root (`A[1]`) the maximum value. So we construct a heap with the subtree with `A[i]` as the parent. We continue this process of constructing a heap with a subtree while subtracting 1 each time until `A[1]` is reached; then a heap is constructed. A node without children has no need to construct another heap, so the initial value of `i` should be the index of the first node having a child.

The left child of `A[j]` is (`A[j × 2]`), and the right child is (`A[j × 2 + 1]`), so the parent of `A[k]` is `A[k / 2]`. Hence, the index of the last node with a child is the quotient of the number of elements (`Last`) by 2, i.e., (`Last` / 2). In this particular case, since `Last` = 10, the index of the last node with a child is (10 / 2 =) 5. Hence, we need to reconstruct heaps while each time subtracting 1 from `Idx`, which begins with (`Last` / 2), until `Idx` = 1 (while `Idx` ≧ 1). Therefore, we insert "`Idx: Last / 2, Idx >= 1, -1`."

Q5:

Points	
	➢ **In state transition, the transition destination is determined only by current state and input.** ➢ **Note that both the red flag and the white flag are down initially.**

Subquestion 1: [Correct Answer] A – b, B – d, C – a

S0 is the start state, in which we have "red flag down" and "white flag down." This can be checked by (2) of [Explanation of Program].

a) From the text of the subquestion, transition (i) is the operation "raise the red flag," so state S1 is "red flag up" and "white flag down."

b) Transition (ii) is from S3 to S0. Because S0 is the state "red flag down" and "white flag down", S3 must be a state in which at least one of the flags is up. On the other hand, S1 is the state "red flag up" and "white flag down", we can see that S3 must be the state "white flag up" and "red flag down."
Therefore, transition (ii) is the motion "lower the white flag."

c) S1 is the state "red flag up" and "white flag down," and S3 is the state "red flag down" and "white flag up," so S2 is the state "red flag up" and "white flag up." Here is a summary of these states.

State	Red flag	White flag
S0	Down	Down
S1	Up	Down
S2	Up	Up
S3	Down	Up

Hence, transition (iii) is the motion "raise the red flag."

Subquestion 2: [Correct Answer] a, g

The information in the initial-value file, according to (2) of [Explanation of Program], consists of (response detection time, incorrect response window display time, number of repetitions).

The process that uses response detection time can be determined as follows. Note (4) of [Explanation of Program]: *There are 5 types of operations which can be received from the input device:* raise the red flag, lower the red flag, raise the white flag, lower the white flag, and move neither flag (when a response is not detected within the response detection time).

This description makes it clear that response detection time is necessary in the process related to these operations. This process is "response detection."

Further, the process that uses incorrect response window display time can be determined as follows. Note (5) of [Explanation of Program]:

The program detects the player's response. If correct, it adds "1" to the number of correct responses and sends the correct response window to the output device. If incorrect, it sends the incorrect response window to the output device and, after the incorrect window display time in the initial value file has elapsed, it sets the flag in the window to the correct state.

This description makes it clear that incorrect response window display time is necessary in the process related to this. This process is "incorrect response processing."

Q6:

Points	➢ **The first character is "<," so scan from the second character.** ➢ **In the "for" statement the index stops at the ">," so scan for the next character with +1.**

Subquestion: [Correct Answer] A – a, B – c, C – a, D – d

The program is complicated, but a specific example is given in Figure 2 and Table 2, so tracking the program is made easy. Additionally, comments in the program are appropriate. As [Program Description] (2) (iii) explains, there are no syntax errors in given tagged character strings, so a character string beginning with a "<" is the beginning of a tag while ">" indicates the end of the tag, and they always make a pair. In the program, the part "XXXX" in "<XXXX>" is stored as a tag name. Also, by matching (corresponding) the statement "`mlstr = parse_ml_date(mlstr +1, 1);`" with "`char *parse_ml_date(char *mlstr, int level)`", we see that scanning begins with the second character of `mlstr` and that the level begins with 1. This assumes that the initial character of `mlstr` is "<".

A:
Referring to Table 2, the pointer to "`STUDENT`" is stored in `elmtbl[elmnum].tag`. The initial value of `elmnum` is 0, so first, the pointer to "`STUDENT`" (address of the letter "S") is stored in `elmtbl[0].tag`. Therefore, "`mlstr`" is inserted here.
Further, in the next statement, the depth of the nesting is set, and the reading is skipped until ">." The fact that "\0" is stored in the position of ">" indicates that the process is completed.

B:
This is processing a tag value, so according to Table 2, we see that the pointer to "`BILL`" is stored `in elmtbl[elmnum].value`. However, in the "for" statement three lines up from this, the index stops at the position of ">." So 1 needs to be added and the position should be at "B." Therefore, "`mlstr + 1`" is inserted.

C:
After processing the tag value, we scan for the next "<" or "</." This is the next "for" statement after the blank B. Just as in the beginning tag process, the index stops at the position of "<," so the program determines whether or not the next character is "/." If it is, the depth of the nesting does not change, but if not, there is a lower-level tag structure. Therefore, here, we need to add 1 to the index by inserting "`elmnum++`".

D:
As explained under C, if the next character is not "/", there is a lower-level tag. In this case, "`parse_ml_date`" is recursively called, and the analysis of the text continues. Therefore, here we insert the condition that the character is not "/."
This is "`*(mlstr + 1) != '/'`".

Q7:

Points	➢ **A thread is started up as a separate execution unit, simultaneously with main.** ➢ **The positive direction for *x* is left; the positive direction for *y* is down.**

Subquestion1 : [Correct Answer] A – c, B – d

A thread is a unit of program execution, and it can be started as a separate execution unit at the same time as the method `main`, which is executed initially. In Java, there are two methods of creating a thread. The first is to inherit the class Thread, and the second is to implement a `Runnable` interface. In this program, the second of these methods is used.

A:
In the abstract class `Motion`, in addition to the constructor `Motion` and method "`run( )`", which continues to display the movement of the point, the abstract method "`update`" is defined to determine the movement of the point. In the method "`run ( )`", a point is drawn at the coordinate position defined by "`while(true){…}`". The "`while`" statement does the following: (1) drawing the point, (2) stopping for 40 milliseconds, (3) blank A, and (4) erasing the point. Here, there is a need to move the point between (2) and (4) above, so in blank A, it is necessary to call "`update()`" which moves the point and changes the contents of the point. Therefore, we need to insert "`point = update(point)`." Note that "Point current = point;" immediately before blank A is a process for retraction to erase the point before the move with "Space.erase(current)" immediately after blank A.

B:
Blank B is inside "`new Thread (…)`" of "`for (int i = 0; i < point.length; i++)`", which is a repeated process. This indicates that as many threads as the number of points to be drawn are generated. For a thread, as mentioned above, the `Runnable` interface must be implemented, but the class `Motion` is an abstract class and cannot be generated as a thread. In other words, the class `SimpleMotion`, which inherits this abstract class `Motion`, is generated as a thread. Therefore, "`new SimpleMotion(points[i])`" is inserted here.

Subquestion2 : [Correct Answer] b

In the figure given for Subquestion 2, the bottom left corner is designated as the origin (0, 0). Hence, the initial signs for each of the motions (i), (ii), (iii), and (iv) are as follows:

(i) X is negative, Y is positive
(ii) X is positive, Y is positive
(iii) X is negative, Y is negative
(iv) X is positive, Y is negative

From [Program 3], we see that in the method "`update()`" `x` and `y` values are each calculated from `directionX` and `directionY`, respectively. Line 2 defines the initial values of `directionX` and `directionY`; both are 1. So X is positive, and Y is positive. Hence, both `x` and `y` are moving in the increasing direction, i.e., arrow (ii). The answer is "The point moves as shown by arrow (ii)."

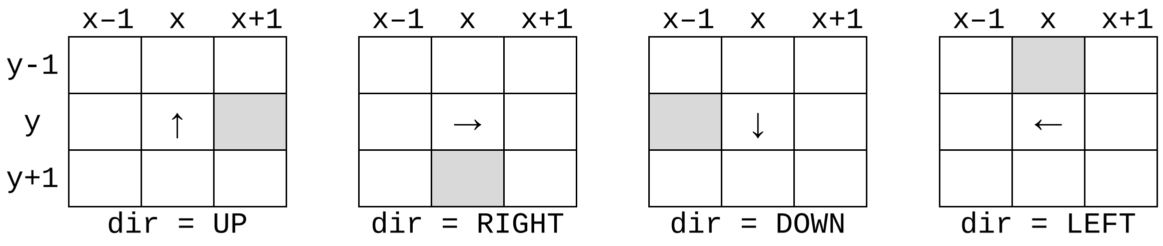
Q8:

Points	➢ **Directions of motion are ↑ for up, → for right, ↓ for down, and ← for left.** ➢ **"rcheck( )" is the function that determines if it is possible to move to the right.**

Note first that in Lines 1 through 4, `UP`, `RIGHT`, `DOWN`, and `LEFT` are defined as constants representing the up, right, down, and left directions, respectively, and corresponding to 0, 1, 2, and 3. The question itself does not give any specifics on the directions, but the item names give clues. In Subquestion 1, the initial condition is `x` = 1 and `y` = 0, pointing to the "Start point" of Figure 1. Here, the only possible direction of motion is down, so you see that `DOWN` indeed means the down direction. Now, consider the processes of the functions "`rcheck()` ", "`fcheck()`", and "`go()`".

(1) `rcheck( )`

When "`dir` = `UP`" this returns "`M[y][x + 1]`", so it is returning the coordinates of the square to the right. If "`dir` = `RIGHT`" it returns "`M[y + 1][x]`", so it is returning the coordinates of the square below. If "`dir = DOWN`" it returns "`M[y][x - 1]`" which is the coordinates of the square to the left, and if "`dir` = `LEFT`" it returns "`M[y - 1][x]`", which is the coordinates of the square right above it. In the figures below, each of the arrows (↑, →, ↓, and ←) indicates the direction determined by the value of "`dir`," and the shaded square is the position of the returned coordinates.



In each of the figures above, if the gray area is either `ROAD` or `EXIT`, then the program can proceed to the shaded square. So, to proceed to the shaded square, we let "`dir = (dir + 1) %4`". The table below shows what happens to the "`dir`" by this operation.

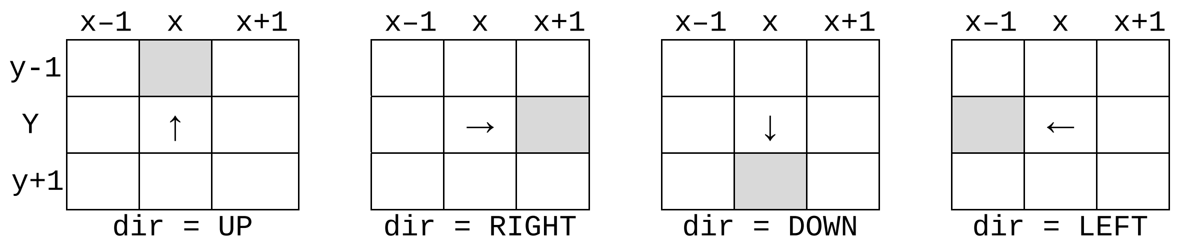
Current "`dir`"	Value	`(dir + 1) %4`	New "`dir`"
↑	0	1	→
→	1	2	↓
↓	2	3	←
←	3	0	↑

The new "`dir`" points to the square to the right of the moving direction. Hence, the program first checks if it is possible to move to the square on the right. Note that if it is possible, the point moves to that square. If not, the program calls `fcheck()` and decides in which direction the point should move next.

(2) `fcheck()`

Consider just as in `rcheck()`.

```
dir = UP:        M[y – 1][x]       → Returns the coordinates of the square above
dir = RIGHT:     M[y][x + 1]       → Returns the coordinates of the square to the right
dir = DOWN:      M[y + 1][x]       → Returns the coordinates of the square below
dir = LEFT:      M[y ][x – 1]      → Returns the coordinates of the square to the left
```



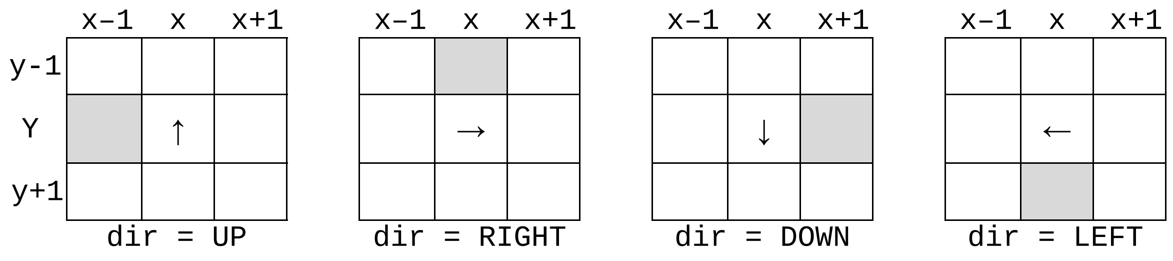
If the shaded area in each of the figures above is either `ROAD` or `EXIT`, then the program can proceed in the direction of the shaded area, so the program directly calls "`go()`" and proceeds. This is checking whether or not it is possible to go to the next square in the moving direction.

Hence, you can see that if `rcheck()` determines that proceeding to the square on the right is not possible, then the program checks if it is possible to move to the next square in the direction of motion.

Now, if the program calls `rcheck()` or `fcheck()` and finds that neither motion is possible, then, as you can see from Line 28, the value of "`dir`" is changed in the direction of the left square. When this happens, "`go()`" is not called, so it involves only the change of "`dir`."

Current "`dir`"	Value	(`dir` + 3) %4	New "`dir`"
↑	0	3	←
→	1	0	↑
↓	2	1	→
←	3	2	↓

The relationship between the current direction and the new direction (shaded) is as follows. Note that the direction is turned by 90 degrees to the left (counterclockwise).



(3) `go()`

The relationship between the value of "`dir`" and the direction of motion is as follows. Note that, unlike "`rcheck()`", "`fcheck()`" and "`go()`" actually changes the coordinates.

```
dir = UP      :y--             → Moves to the square above
dir = RIGHT   :x++             → Moves to the square on the right
dir = DOWN    :y++             → Moves to the square below
dir = LEFT    :M[y][x – 1]     → Moves to the square on the left
```

Subquestion1 : [Correct Answer] a

First, the program calls “`rcheck( )`” and sees if it is possible to proceed. Because the initial value of “`dir`” is `DOWN` (= 2), the program checks the square to the right of the direction of motion (that is `M[0][0]`). Going in that direction is not possible, so then the program calls “`fcheck()`” to check the direction of motion (square below, which is Square 1). Because it is possible to move to Square 1, “`go()`” is called, and the position moves to `M[1][1]` (Square 1).

Next, “`rcheck()`” is called again, and the square to the right of the direction of motion (`M[1][0]`) is checked, but again this is a square to which moving is not allowed. So “`fcheck()`” is called, and the square in the direction of motion (`M[2][1]`) is checked. Proceeding to that square is possible, so now the program moves to `M[2][1]`. These processes are repeated, and the path moves from Square 1 to Square 2.

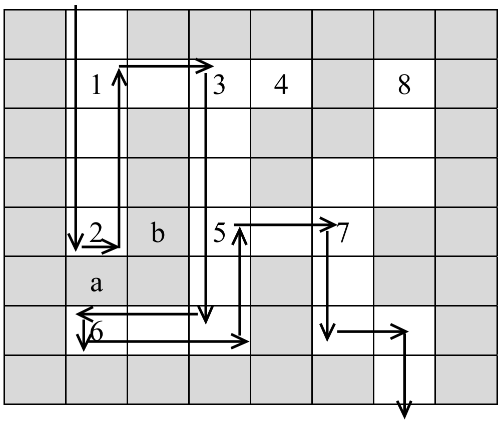
`M [0] [1]`

▨	↓	▨	▨	▨	▨	▨	▨
▨	1		3	4	▨	8	▨
▨	↓	▨		▨	▨		▨
▨	↓	▨		▨			▨
▨	2	b	5		7	▨	▨
▨	a	▨		▨		▨	▨
▨	6						▨
▨	▨	▨	▨	▨	▨		▨

At `M[4][1]`, the program cannot go anywhere either by “`rcheck()`” or “`fcheck()`”, so the statement of Line 28 is executed. “`go()`” turns the direction of motion by 90 degrees to the left, so the direction is changed to `dir` = 1 (`RIGHT`, →). Next, the program calls “`rcheck()`” and the square to the right of the direction of motion (the square below, `M[5][1]`, between Square 2 and Square 6) is checked. However, it cannot move there, so “`fcheck()`” is called to check the square in the direction of motion (`M[4][2]`, between Square 2 and Square 5). This too is a square impossible to move to. Again, the statement of Line 28 is executed, turning the direction of motion by 90 degrees to the left, to the up direction “↑.” In this state, the direction “↑” moves the point from Square 2 to Square 1. However, at Square 1, the program checks the square to the right of the direction ↑ of motion, causing it to move toward Square 3. Consequently, we have the following:

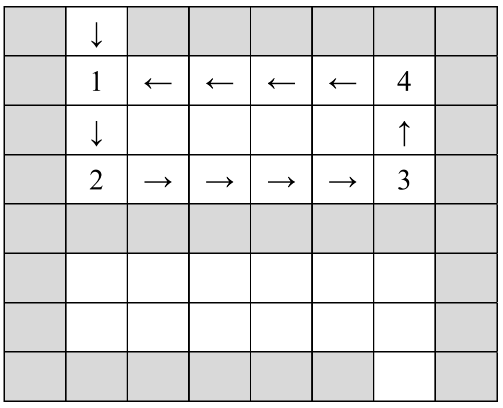
Start point → Square 1 → Square 2	(then returns)
Square 2 → Square 1	(then turns right)
Square 1 → Square 3	(at Square 3, the direction is “→”, so it moves to the right, which is down).
Square 3 → Square 5 → (“↓”) → `M[5][3]` → Square 6	(at Square 5, it moves in the “↓” direction).
Square 6 → (“→”) → `M[6][3]`→ Square 5	(at Square 6, a left 90-degree turn occurs twice, changing the direction to “→.” At `M[6][3]`, it turns to the “↑” direction.)
Square 5 → Square 7	(at Square 5, the direction changes to the right, toward Square 7)
Square 7 → End point.	

Therefore, the path is “Start point→*1*→*2*→*1*→*3*→*5*→*6*→*5*→*7*→End point.”



Subquestion2 : [Correct Answer] b

As explained under Subquestion 1, we follow the movement. Here is how it moves:



Square 1: The program looks to the right of the direction of motion, but it cannot move to the right. So it moves forward in the direction of motion.
Square 2: The program looks to the right, but it cannot move there. So it looks forward, but forward movement is also impossible. So it changes the direction to the left (→).
Square 3: Again, the program checks the square on the right and the square ahead in front. Neither motion is possible, so it changes the direction to the left of the direction of motion, changing to ↑.
Square 4: Again, the program checks the square on the right and the square ahead in front. Neither motion is possible, so it changes the direction to the left of the direction of motion, changing to ←.

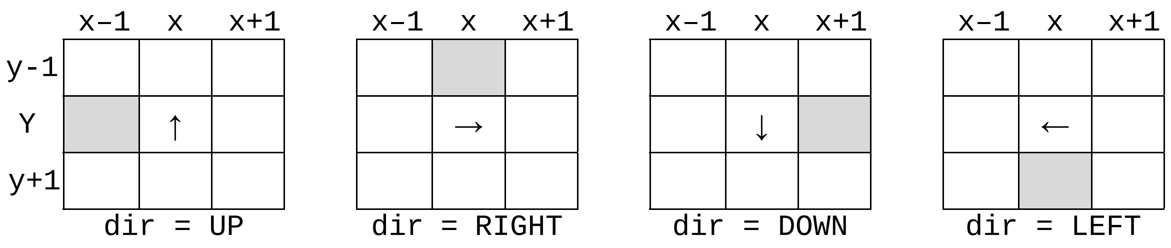
Therefore, the motion *1*→*2*→*3*→*4* is repeated. The answer then is "Start point→*1*→*2*→*3*→*4*→*1*→*2*→*3*→*4*→… are repeated endlessly."

Subquestion3 : [Correct Answer] a

After the "`while`" (Line 20) statement, we add a "`print`" statement. Now, since the condition for carrying on the "`while`" statement is "`M[y][x]! = EXIT`", the program escapes the "`while`" condition when "`M[y][x] = EXIT`". Then, since the last "`M[y][x]`" (the end point) is not printed, a "`print`" statement needs to be added after Line 29 also.
Therefore, a "`print`" statement should be added immediately after each of Lines 20 and 29.

Subquestion4 : [Correct Answer] b

The processing of "`lcheck()`" is as follows:



Whereas "`rcheck()`" checked the right side of the direction of motion, "`lcheck()`" checks the left side. The program is to use "`lcheck()`" to see if it is possible to move to the left. If it is possible, the direction of motion is to be turned by 90 degrees to the left. For this, we change Line 23 so that, just like Line 28, the motion gets turned by 90 degrees to the left. Since Line 23 is changed to make this 90-degree turn to the left, Line 28 needs to be modified so that it rotates the direction to the right by 90 degrees.
Hence, we see that the following changes are necessary:

(Line number)
23 `dir = (dir+1) % 4;` → `dir = (dir+3) %4;` /* 90 degrees to the left */
28 `dir = (dir+3) % 4;` → `dir = (dir+1) %4;` /* 90 degrees to the right */

Therefore, the changes are the following: "`+1` on Line 23 must be changed to `+3` and `+3` on Line 28 must be changed to `+1`."

Q9:

Points	➢ **The event types are DIGIT, OPERATOR, and CLEAR.** ➢ **CLEAR has one fewer argument.**

The role of each class is as follows:

(1) Class `CalculatorEvent`
This is the class when a calculator key is pressed. There are two constructors called "`CalculatorEvent`" but this is because there are cases where an argument is necessary and cases where an argument is not necessary, depending on the type of keys pressed.

(2) Interface `CalculatorOutput`
This is an interface only to declare the two abstract methods "`display`" with different argument types, to be used as arguments of constructors of the class `Calculator`.

(3) Class `Calculator`
This is the class of the calculator itself. Note that the argument of the constructor is defined as a variable of the `CalculatorOutput` interface type. When this class gets instantiated, the designated interface-type variable gets substituted directly into the interface-type variable `output`.

(4) Class `CalculatorTest`
This is the class to test the class `Calculator` and where the method "`main`" is defined. Note that the argument of the constructor necessary to instantiate the class `Calculator` is described, including the blank B.

Subquestion1 : [Correct Answer] A – e, B – c, C – e

A:
Blank A is the constructor process when there is only one argument (`int type`). When "`type`" is CLEAR (clear key), the second argument "`value`" is not used, so some appropriate value is placed for "`value`" of the second argument, and these arguments call two constructors. For all options in the answer group, the second argument is 0. Also, for arguments to call two constructors, they use the keyword "`this`", referring to themselves. Hence, "`this(type,0)`" is inserted.

B:
Blank B is the argument when the instance "`calc`" of the class `Calculator` is generated. Based on the fact that `CalculatorOutput` is packaged as an anonymous class and that the method "`display`" outputs the designated numerical value or character string to `System.out` in this packaging, one can figure out that blank B must generate the anonymous class that packages the interface `CalculatorOutput`. Also, the argument delivered to the constructor of the class `Calculator` is a variable of the `CalculatorOutput` interface type, and the class that packages the interface "`CalculatorOutput`" must package two (one for each type) methods "`display`."

For these reasons, in this section containing blank B, the interface `CalculatorOutput` is to be packaged by an unnamed class (anonymous class), and, at the same time, it must be instantiated by a new keyword. Hence, here, the answer needs to include the name of the interface to be packaged by an anonymous class. The answer then is "`new CalculatorOutput()`."

Incidentally, an anonymous class is used in various situations, such as when we take an existing class or interface, modify it partially to create a new class, and wish to use the new class locally within a class. In such a case, packaging and inheriting take place without “`extends`” or “`implements`” key words.

C:
Blank C corresponds to the argument used when the instance “`event`” is generated when the input value is 0 through 9. The argument is delivered, considering cases like = and +, and the first argument is `CalculatorEvent.DIGIT`. For the second argument, the program can simply deliver the number entered, but the characters “0” through “9” need to be converted to numbers 0 through 9. When characters “0” through “9” correspond to 0x30 through 0x39 in hexadecimal numbers, conversion to numerical values can be accomplished simply by subtracting the hexadecimal number 0x30. Therefore, what needs to be inserted to blank C is “`CalculatorEvent.DIGIT, c - ‘0’`.”

Subquestion2 : [Correct Answer] D – a, E – d, F – i

D:
This is the case where the input is the character string “`3*4***=`”. When the multiplication key is entered multiple times consecutively, the next statement after displaying the operation result executes “`register = 0;`”, setting the variable that retains the result (“`register`”) to 0. Hence, even if the equal key is pressed in the end, the display result remains 0.

E:
At the time “`3*4 =`” is entered, both the display and the value of the variable “`accumulator`” are 12. When “`+5`” follows, the value displayed in the end is the last input value “5.” This is because “`register`” becomes 0 when the equal sign is pressed. With “register” 0, until another number is added to the input value 5, the display shows “5.” Hence, the display result is “5.”

F:
At the time “`3 + 4 / 0`” is entered, “`accumulator`” contains 7, which is the result of the calculation “`3 + 4`”, while “`register`” stores the last input value 0. Then, when the last equal key is pressed, division by 0 occurs, triggering the exception processing “`ArithmeticException`”. Hence, “`Error`” is displayed.