Fundamental IT Engineer Examination (Afternoon)

Questions must be answered in accordance with the following:

Question Nos.	**Q1 – Q6**	**Q7 , Q8**
Question Selection	**Compulsory**	**Select 1 of 2**
Examination Time	**13:30 – 16:00 (150 minutes)**	

Instructions:

1. Use a pencil. If you need to change an answer, erase your previous answer completely and neatly. Wipe away any eraser debris.

2. Mark your examinee information and test answers in accordance with the instructions below. Your answer will not be graded if you do not mark properly. Do not mark or write on the answer sheet outside of the prescribed places.

   (1) **Examinee Number**
   Write your examinee number in the space provided, and mark the appropriate space below each digit.

   (2) **Date of Birth**
   Write your date of birth (in numbers) exactly as it is printed on your examination admission card, and mark the appropriate space below each digit.

   (3) **Question Selection**
   For **Q7** and **Q8**, mark the Ⓢ of the question you select to answer in the "Selection Column" on your answer sheet.

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

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

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

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

   [Sample Answer]

Sample	Ⓐ	●	Ⓒ	Ⓓ

Notations used for pseudo-language

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

[Declaration, comment, and process]

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

[Logical constants]

`true`, `false`

[Operators and their priorities]

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

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

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

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

There are two programs; Program 1 and Program 2. Program 1 is implemented for the processor A and Program 2 is implemented for the processor B.
The following table shows the occurrence of each instruction type in the two programs.

Instruction	Program 1 on Processor A	Program 2 on Processor B
Integer Addition/Subtraction	100	20
Integer Multiplication	200	30
Load/Store Word	150	50
Floating Point Addition/Subtraction	10	60
Floating Point Multiplication	40	160

The second table shows the number of cycles needed for each instruction type.

Instruction	Processor A	Processor B
Integer Addition/Subtraction	1	5
Integer Multiplication	2	10
Load/Store Word	6	2
Floating Point Addition/Subtraction	10	15
Floating Point Multiplication	20	5

The clock frequency of processor A is 230 MHz and of processor B is 110 MHz.

Subquestion

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

(1) Program 1 on processor A completes its execution in [A] microseconds, while Program 2 on processor B completes its execution in [B] microseconds. Here, take no account of any overhead nor interruptions.

(2) The MIPS (Million Instructions Per Second) value is [C] for Program 1 on processor A, and [D] for Program 2 on processor B.

(3) Program 3 is newly created. The following table shows the occurrence of each instruction type in Program 3. Since the number of non-floating point instructions are negligible, consider only floating point instructions. Then, MFLOPS (Million Floating Point Operations per Second) value is 15.33 for processor A, and [E] for processor B. So the program execution on processor A is faster than on processor B.

Instruction	Program 3
Integer Addition/Subtraction	100
Integer Multiplication	0
Load/Store Word	100
Floating Point Addition/Subtraction	100,000
Floating Point Multiplication	100,000

Answer group

a) 8 b) 10 c) 11 d) 16
e) 20 f) 23 g) 40 h) 50

Q2. Read the following description concerning a specification of some hard disk's file system, and then answer Subquestions 1 and 2.

A file system manages the files, directories, its storage in the hard disk, and how the files get loaded into memory and used by programs. Each hard disk partition is divided into groups. A group size is ideally in accordance with the structure of the tracks and sectors of the hard disk to make for efficient retrieval. There is no significant performance loss when retrieving data in the same group. Performance loss is noticeable when retrieving data across different groups.

Figure 1 shows the structure of a group for some file system. Each group contains a superblock, a group descriptor, a block bitmap, an inode bitmap, an inode table, and finally data blocks.

<table>
<tr><td>Superblock</td><td>Contains entire file system information of the hard disk partition. A hard disk partition is a logical division of the hard disk.</td></tr>
<tr><td>Group Descriptor</td><td>Contains information local to the group being loaded.</td></tr>
<tr><td>Block Bitmap</td><td>Contains n bits corresponding to the data blocks 1 to n. A bit with value 1 indicates that the corresponding data block is being used, and value 0 indicates unused.</td></tr>
<tr><td>Inode Bitmap</td><td rowspan="2">Each file is described with exactly one inode. Inodes are located in the inode table. Each entry has an inode ID. The inode table contains all inodes and is fixed in size. See Figure 2 for example.

Inode bitmap contains the status of the inode table entries. A bit with value 1 indicates that the corresponding entry is being used, and value 0 indicates unused.</td></tr>
<tr><td>Inode Table
<table>
<tr><th>Inode ID</th><th>File information</th></tr>
<tr><td>01</td><td></td></tr>
<tr><td>02</td><td></td></tr>
<tr><td>03</td><td></td></tr>
<tr><td>:</td><td></td></tr>
<tr><td>n</td><td></td></tr>
</table></td></tr>
<tr><td>Data Block 1</td><td rowspan="5">Contains the file's data. The file's data is subdivided into one or more data blocks. Each data block is fixed in size with 4096 bytes each. Data up to 4096 bytes will use 1 data block, data up to 8192 bytes will use 2 data blocks, etc.</td></tr>
<tr><td>Data Block 2</td></tr>
<tr><td>:</td></tr>
<tr><td>Data Block n−1</td></tr>
<tr><td>Data Block n</td></tr>
</table>

Figure 1 Structure of a group

File information in inode table contains mode (file permission and kind of file), owner information, size, time stamp, and 8 direct pointers (dbptr 01 to 08) to data blocks plus 2 indirect pointers (dbptr 09 and 10) to more data blocks. If a file makes use of 8 data blocks or less, then up to 8 direct pointers will point to the data blocks used. If the file makes use of 9 data blocks or more, each indirect pointer will point to the data block containing the next 1024 direct pointers for succeeding data blocks.

Inode ID		39	8D	A5	FA
File Information	mode				
	owner	W	X	Y	Z
	size	20480	32000	5120	23552
	time stamp				
	dbptr 01	A01	B01	A04	E01
	dbptr 02	D06	C01	C02	E02
	dbptr 03	C03	D01	nil	E03
	dbptr 04	A02	F01	nil	E04
	dbptr 05	B05	B04	nil	E05
	dbptr 06	nil	C04	nil	E06
	dbptr 07	nil	D04	nil	nil
	dbptr 08	nil	F04	nil	nil
	dbptr 09	nil	nil	nil	nil
	dbptr 10	nil	nil	nil	nil

Note: “nil” indicates “unused”.

Figure 2 Inode of 4 different files in the same group

Addr	01	02	03	04	05	06
A	*A01 data ...*	*A02 data ...*		*A04 data ...*		
B	*B01 data ...*			*B04 data ...*	*B05 data ...*	
C	*C01 data ...*	*C02 data ...*	*C03 data ...*	*C04 data ...*		
D	*D01 data ...*			*D04 data ...*		*D06 data ...*
E	*E01 data ...*	*E02 data ...*	*E03 data ...*	*E04 data ...*	*E05 data ...*	*E06 data ...*
F	*F01 data ...*			*F04 data ...*		

Note: The addressing is in row-order form.
It begins with A01 through A06 and followed by the next row and so on.

Figure 3 Data blocks of the group in Figure 2

Subquestion 1

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

Figure 2 represents 4 files found in a group, and Figure 3 shows the data blocks of the group. Then, the block bitmap in the group contains the value [ ] in hexadecimal.

Answer group

a) D26 F25 FE4
b) D9F 9F9 FE4
c) E52 ED1 EDC
d) FA2 F41 EDC

Subquestion 2

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

The file system takes into consideration design issue of fragmentation but does not eliminate it. In the example shown in Figure 2 and 3, internal fragmentation (unused space) will exist where the file with inode [A] will be most fragmented and the file with inode [B] will have ZERO fragmentation. Such a form of fragmentation is tolerable in order to attain faster retreival of data. The ratio of internal fragmentation across the whole group becomes [C] as the number of large files increases and the reverse occurs as the number of small files increases.

Assuming that a series of append file operations are performed in the following order:

- File of inode 8D doubles in size
- File of inode A5 increases by 13000 bytes
- File of inode FA increases by 10000 bytes
- File of inode 39 increases by 8000 bytes

The group will run out of data blocks when the file with inode [D] tries to write its updates. In this situation, a new group is brought in and the free data block from that group is used. The corresponding dbptr(s) of inode [D] will be pointing to the data block of another group.

Answer group

a) larger	b) smaller	c) 39
d) 8D	e) A5	f) FA

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

The following is a part of the functions of a sales and distribution company. One of their business functions is sending the sales teams to townships by van cars carrying sales items to sell.

For each sales trip, the sales team need to:

1. Issue the specific quantity of sales items from a warehouse
2. Make the sales trip and sell the sales items
3. Return to the office and give back the remaining sales items to the warehouse

Transfer history of sales items are stored in **TransferHis** table. There are two types of transfer records for each trip:

1. From warehouse to van car before going trip
2. From van car to warehouse after the trip

For the same trip, the trip code (TripCode) and township name (TspName) will be fixed. The sales trip may take a few days.

For simplicity, suppose that there is only one item. So, the item code is not stored in the table.

Sample data in the following TransferHis table shows:

- On 1-Jan-10, WH01 transfers 100 stocks to Van01 that will go to Bago. Trip code is T001.
- On 3-Jan-10, Van01 returns 30 stocks to WH01 after selling stocks at Bago. Trip code is T001. Thus, sold quantity is 70.
- …

Table **TransferHis**

TranID	TDate	TripCode	TspName	FromLoc	ToLoc	Qty
1	1-Jan-10	T-001	Bago	WH01	Van01	100
2	3-Jan-10	T-001	Bago	Van01	WH01	30
3	4-Jan-10	T-002	Mhaw Bi	WH02	Van02	200
4	5-jan-10	T-002	Mhaw Bi	Van02	WH02	50
5	6-jan-10	T-003	Insein	WH01	Van03	300
6	7-jan-10	T-003	Insein	Van03	WH01	80

Subquestion 1

An SQL statement is created to generate the sales summary of each trip in the following format. From the answer group below, select the correct answer to be inserted into the blank [ ] in the following SQL statement.

TripCode	TspName	Qty
T-001	Bago	70
T-002	Mhaw Bi	150
T-003	Insein	220

```
Select T1.TripCode, T1.TspName, T1.Qty-T2.Qty as Qty
From TransferHis T1 left join TransferHis T2 on
```

[ ]

Answer group

a)
```
      T1.TripCode = T2.TripCode
Where T1.FromLoc <> T2.FromLoc and
      T1.TspName = T2.TspName
```

b)
```
      T1.TspName = T2.TspName
Where T1.TDate <= T2.TDate and
      T1.TripCode = T2.TripCode
```

c)
```
      T1.TripCode = T2.TripCode
Where T1.Tdate <= T2.TDate and
      T1.FromLoc <> T2.FromLoc
```

d)
```
      T1.TripCode = T2.TripCode
Where T1.FromLoc = T2.ToLoc and
      T1.Tdate <> T2.TDate
```

Subquestion 2

SELECT clause is very flexible. The same result as above Subquestion 1 can be obtained by the following SQL statement. From the answer group below, select the correct answer to be inserted into the blank [ ] in the following SQL statement. Note that part of the conditions are intentionally shaded by ▨.

```
Select T1.TripCode, T1.TDate, T1.TspName, T1.Qty-T2.Qty Qty from
[                                                              ]
where T1.TripCode = T2.TripCode and
      T1.▨▨▨▨▨ ▨ T2.▨▨▨▨▨ and
      T1.▨▨▨▨▨ ▨ T2.▨▨▨▨▨
```

Answer group

a)
```
Select * From TransferHis T1
left join TransferHis T2 on T1.TripCode=T2.TripCode
```

b)
```
(Select * From TransferHis) T1, (Select * From TransferHis) T2
```

c)
```
(Select * From TransferHis) T1
union (Select * From TransferHis) T2
```

d)
```
(Select TripCode, TDate, TspName, FromLoc, ToLoc, Qty
From TransferHis ) T1
union  all
(Select TripCode, TDate, TspName, FromLoc, ToLoc, Qty
From TransferHis ) T2
```

Subquestion 3

The result of the above Subquestion 2 is inserted into a Table **TripSales** with the fields TripCode, TspName and Qty. Then, another SQL statement is created to choose a record that has the second largest sales quantity. From the answer group below, select the correct answer to be inserted into the blank [ ] in the following SQL statement.

```
Select * From TripSales
Where Qty in
(
[                                        ]
)
```

Answer group

a) `Select Qty From TripSales`
   `Where Qty <> (Select max(Qty) from TripSales)`

b) `Select Qty From TripSales`
   `Where Qty not in (Select max(Qty) From TripSales)`

c) `Select max(Qty) From TripSales`

d) `Select max(Qty) From TripSales`
   `Where Qty not in (Select max(Qty) From TripSales)`

Q4. Read the following description concerning a backbone network, and then answer Subquestions 1 through 4.

The Figure below shows a network configuration of a system development company.

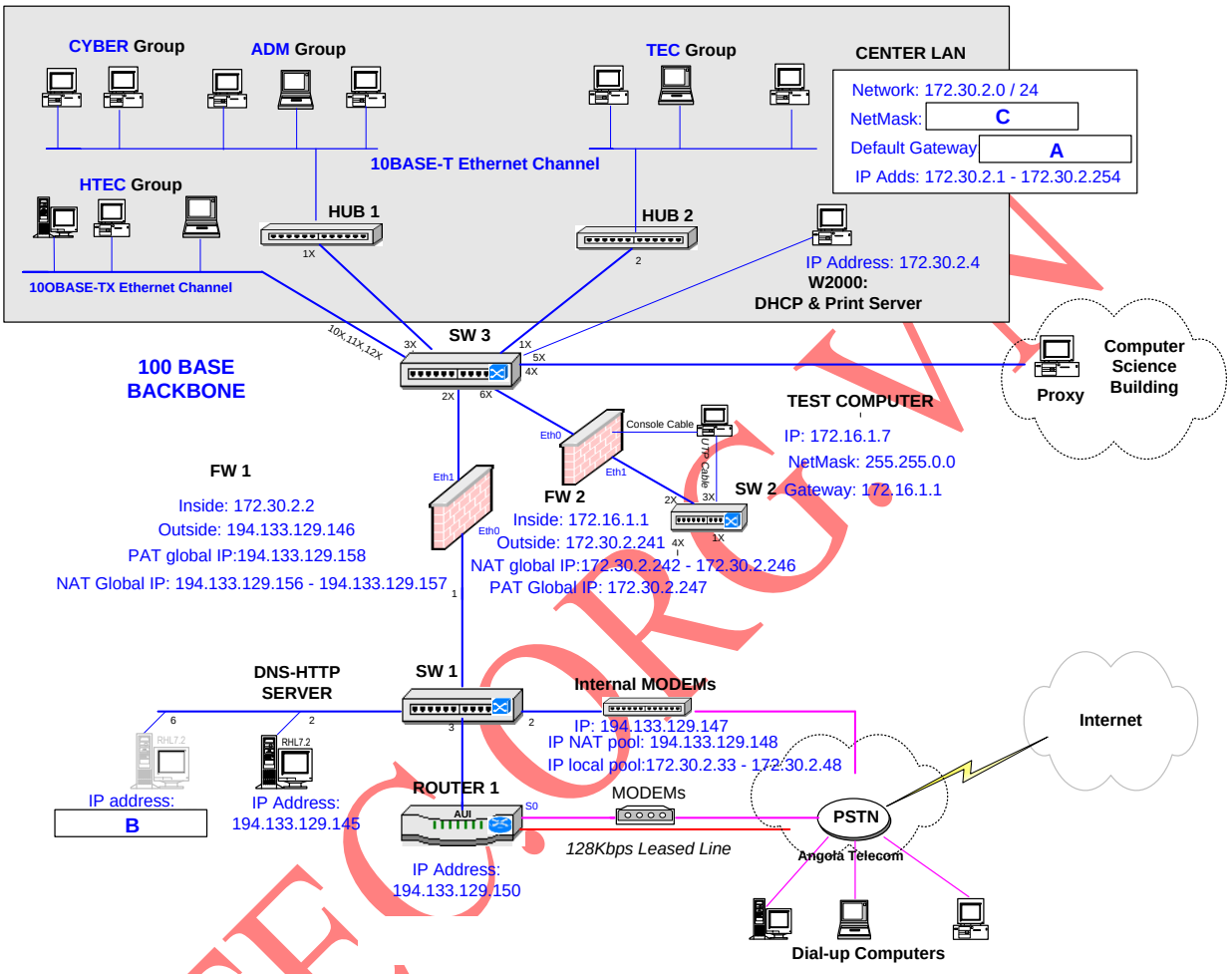


Figure network configuration

The following table shows some IP addresses on the network 194.133.129.144/28.

Table Some IP addresses on network 194.133.129.144/28

Name	Local IP address	Global IP address
DNS-auth		194.133.129.145
FW 1	172.30.2.2	194.133.129.146
Internal MODEMs	172.30.2.33 - 172.30.2.48	194.133.129.147
ASYNCH 1 Dial-up		194.133.129.152
ASYNCH 2 Dial-up		194.133.129.153
ASYNCH 3 Dial-up		194.133.129.154
ASYNCH 4 Dial-up		194.133.129.155

Subquestion 1

From the answer group below, select the appropriate IP address for Default Gateway to be inserted into the blank [A] in the Figure.

Answer group

a) 170.30.2.4
b) 172.16.1.1
c) 172.30.2.2
d) 194.133.129.150

Subquestion 2

From the answer group below, select the appropriate IP address for HTTP Server to be inserted into the blank [B] in the Figure.

Answer group

a) 194.133.129.144
b) 194.133.129.145
c) 194.133.129.146
d) 194.133.129.151

Subquestion 3

From the answer group below, select the appropriate NetMask to be inserted into the blank [C] in the Figure.

Answer group

a) 255.0.0.0
b) 255.255.0.0
c) 255.255.2.0
d) 255.255.255.0

Subquestion 4

From the answer group below, select the appropriate broadcasting address for the network shown in the Table.

Answer group

a) 194.133.129.156
b) 194.133.129.157
c) 194.133.129.158
d) 194.133.129.159

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

Company B manufactures dairy products whose main ingredient is milk, such as butter, cheese, and yogurt.

Milk is procured from Dairy farms J, K, and L, and kept in separate tanks. It is decided by the manufacturing plan whether milk from a single dairy farm is used, or milk with a standard mixing ratio is used, for each manufacturing unit. A lot number is assigned to one manufacturing unit, by which manufacturing date and type of product can be distinguished.

Company B has decided to create a program that calculates the required amount of milk from each dairy farm for each manufacturing date according to its manufacturing plan.

[Explanation of Required Amount File]

For each manufacturing date, product code, and lot number, decided by the manufacturing plan, the "required amount" (positive integer value) of the ingredient milk and the "dairy farm code", which indicates whether the milk comes from one dairy farm or is mixed, are recorded. The dairy farm code is "J", "K", or "L" (for dairy farm J, K, or L, respectively) or "M" for "mixed".

Figure 1 shows the format of the required amount file. The records are sorted in ascending order by "manufacturing date", "product code", and "lot number", in this order of priority.

Manufacturing Date	Product Code	Lot Number	Required Amount of Milk	Dairy Farm Code

Figure 1 Format of the Required Amount File

[Explanation of Standard Mixing Ratio File]

This is an indexed file with "product code" as the key. For each product, the standard mixing ratio of the milk from each dairy farm is recorded such that the total is 100% (integer values between 0 and 100, inclusive).

Figure 2 shows the format of the standard mixing ratio file.

Product Code	% of J	% of K	% of L

Figure 2 Format of the Standard Mixing Ratio File

[Explanation of Total Amount File]

For each manufacturing date, the required amount of milk from each dairy farm is calculated and is recorded as the total required amount.

Figure 3 shows the format of the total amount file.

Manufacturing Date	Required Amount from J	Required Amount from K	Required Amount from L

Figure 3 Format of the Total Amount File

[Program Description]

The program reads the required amount file and performs one of the processes (1) through (3) for each record.

(1) If the dairy farm code is “M”, it is “mixed”, so the program calculates the required amount from each of the dairy farms J, K, and L, using the information in the standard mixing ratio file and the required amount file. In this calculation, any decimal parts are rounded off, and the rounding errors are corrected by the required amount of milk from dairy farm L.

(2) If the dairy farm code is either “J”, “K”, or “L”, it means a “single farm”, so the entire required amount comes from dairy farm J if the code is “J”; from dairy farm K if the code is “K”; and from dairy farm L if the code is “L”.

(3) If the dairy farm code is not “J”, “K”, “L” nor “M”, it is an error, so the content of the record is written to the error file.

Except for error cases, the required amount of milk from each of the dairy farms J, K, and L is calculated for each manufacturing date and is written to the total amount file.

Figure 4 shows the input/output relational diagram, and Figure 5 shows the module structure.

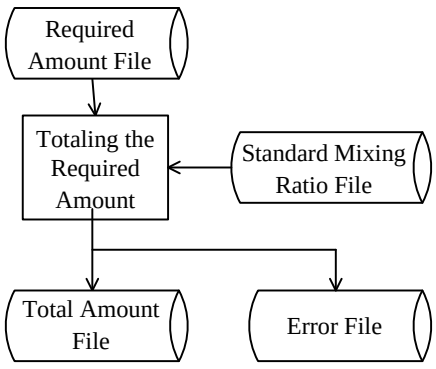


Figure 4 Input/Output Relational Diagram

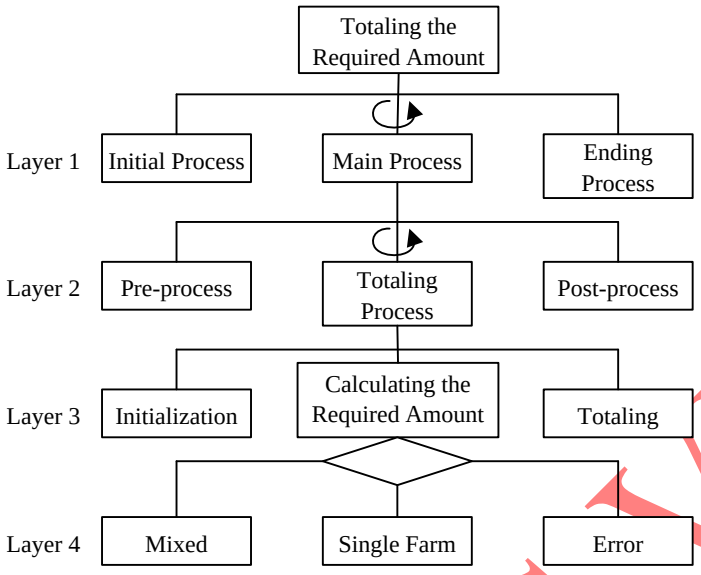


Figure 5 Module Structure

Subquestion 1

Figure 6 shows a flowchart of the main process and calculation of the required amount. From the answer groups below, select the correct answers to be inserted into the blanks [ ] in Figure 6.

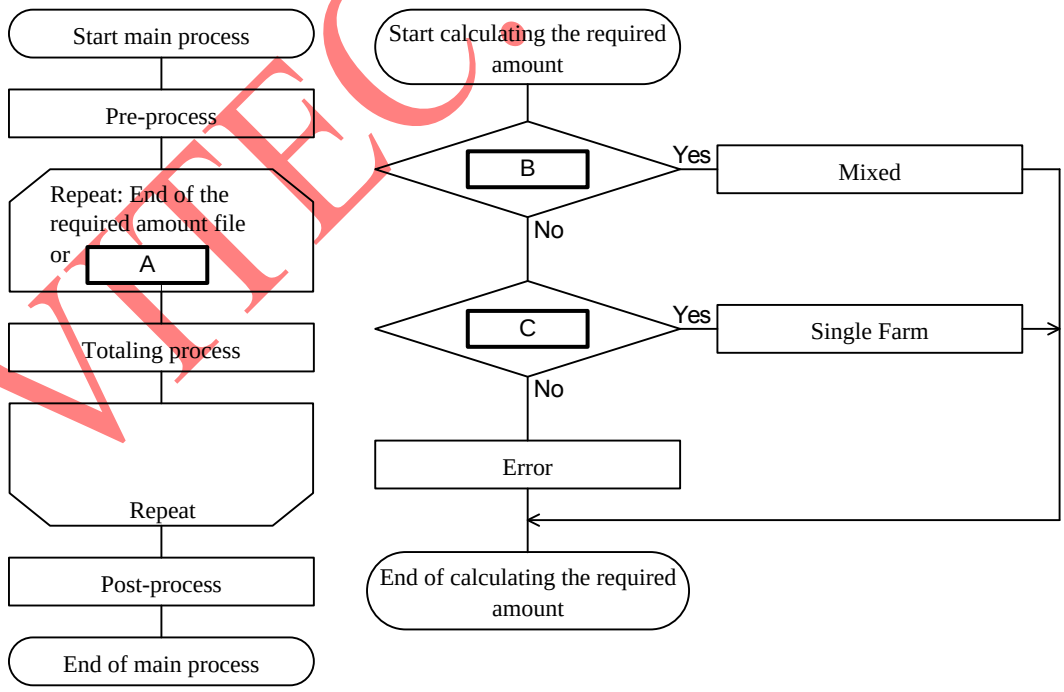


Figure 6 Flowchart of Main Process and Calculation of Required Amount

Answer group for A

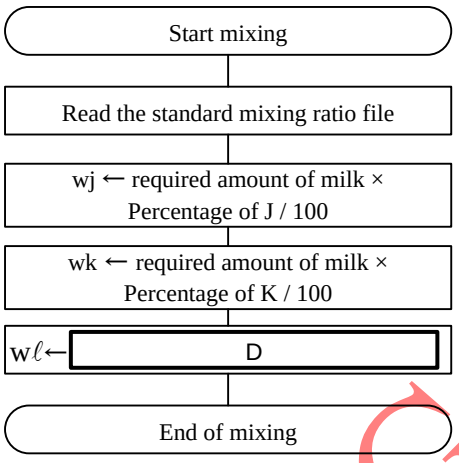
a) manufacturing date is changed
b) product code is changed
c) dairy farm code is changed
d) lot number is changed

Answer group for B and C

a) (dairy farm code ≠ “J”) or (dairy farm code ≠ “K”) or (dairy farm code ≠ “L”)
b) (dairy farm code ≠ “M”) or ((dairy farm code ≠ “J”) and (dairy farm code ≠ “K”) and (dairy farm code ≠ “L”))
c) (dairy farm code ≠ “M”) or (dairy farm code ≠ “J”) or (dairy farm code ≠ “K”) or (dairy farm code ≠ “L”)
d) (dairy farm code = “J”) or (dairy farm code = “K”) or (dairy farm code = “L”)
e) (dairy farm code = “M”) or ((dairy farm code ≠ “J”) and (dairy farm code ≠ “K”) and (dairy farm code ≠ “L”))
f) dairy farm code ≠ “M”
g) dairy farm code = “M”

Subquestion 2

Figure 7 shows a flowchart of mixing. From the answer group below, select the correct answer to be inserted into the blank [ ] in Figure 7.



Note: "required amount of milk" is the value read from the required amount file.
"percentage of J" and "percentage of K" are the values in the standard mixing ratio file.
The required amount of milk from dairy farms J, K, and L are stored into wj, wk, and wℓ, respectively.
In division, any decimal parts of the quotient are rounded off.

Figure 7 Flowchart of Mixing

Answer group

a) required amount of milk − wj
b) required amount of milk − wj - wk
c) required amount of milk − wk
d) required amount of milk × (100 − percentage of J − percentage of K) ÷ 100
e) required amount of milk × percentage of L ÷ 100

Q6. Read the following description of programs and the programs themselves, and then answer Subquestions 1 and 2.

The two programs, compress_Stream and decompress_Stream, implement the LZW compression and decompression algorithms. LZW is a dictionary-based compression and decompression technique. When compressing a data stream, it encodes the stream by referencing a dictionary; it outputs a stream of code numbers that correspond to a string entry in the dictionary, into the compressed file. When decompressing a data stream, it follows the same process in the reverse order. The algorithm is simple and takes advantage of frequently recurring sub-strings in the data stream.

[Description of Program 1 (compress_Stream)]

Compression begins with the dictionary that initially contains all possible single character entries for a given input stream. Then, the program reads the input stream one character at a time, and outputs a code number that corresponds to an entry's index position in the dictionary.

Each time the program comes across a new sub-string (for example, "AB"), the program adds it to the dictionary. Meanwhile, each time the program comes across a character or sub-string, which has come across previously, the program proceeds onto the next character and concatenates it with the current character or current sub-string to get a new sub-string. The next time the program comes across that sub-string, the program outputs its corresponding code number.

Assuming that the array `dictionary` used in the program is defined externally, and all possible single characters appear in the given input stream have already been loaded.

Here is an execution example. When the input stream contains "A B B A B A B A K", then the compressed result "1 2 2 4 7 3" is written to the output stream. About this example, initial contents of the dictionary is shown in table 1, final contents of the dictionary is shown in table 2, and a summary of the process is shown in table 3.

Table 1 Initial contents of the dictionary

Index Position	Contents of the Entry
0	'' (null)
1	A
2	B
3	K

Table 2 Final contents of the dictionary

Index Position	Contents of the Entry
0	'' (null)
1	A
2	B
3	K
4	A B
5	B B
6	B A
7	A B A
8	A B A K

Table 3 Summary of the process (Program 1)

Current String s	Input Character ch	Is s+ch in the dictionary	Code Output	Dictionary	
				Index	Entry
''	A	Yes			
A	B	No	1	4	A B
B	B	No	2	5	B B
B	A	No	2	6	B A
A	B	Yes			
A B	A	No	4	7	A B A
A	B	Yes			
A B	A	Yes			
A B A	K	No	7	8	A B A K
K			3		

[Program 1]

```
○ Program compress_Stream
○ string type: s, code          /* code: index position of a dictionary entry */
○ character type: ch
○ procedure: getInStream(char)
                        /* reads the next character into char from input stream */
○ procedure: putOutStream(code)
                                  /* writes the encoded code to output stream */
○ procedure returns integer type: addEntry(string, dictionary)
                  /* adds string into dictionary and returns its index position */
○ procedure returns integer type: findEntry(string, dictionary)
                   /* finds string in dictionary and returns its index position */
○ procedure returns integer type: stringExists(string, dictionary)
                    /* if string exists in dictionary, returns 1; else returns 0 */

• s ← ''                /* initialize s as a null string */
• code ← 0              /* initialize code to 0 corresponding to null string */
• getInStream(ch)               /* read the first character */
■ ch ≠ ''               /* loop while the next character exists (not end of data) */
  ▲ stringExists(s+ch, dictionary) = 1    /* "+" means "concatenation" */
    • s ← s + ch
  ▼ • code ← findEntry(s, dictionary)
    • putOutStream(code)
    • addEntry(s+ch, dictionary)
    • s ← ch
  • getInStream(ch)             /* read the next character */
■
• code ← findEntry(s, dictionary)
• putOutStream(code)
```

Subquestion 1

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

Program 1 is executed for the following two cases:

(1) When " A A A A " (4 consecutive A's) is given from the input stream, the compressed result " [A] " is written to the output stream.

(2) When " A A A A A " (5 consecutive A's) is given from the input stream, the compressed result " [B] " is written to the output stream.

The dictionary initially contains '' (null) and A at index positions 0 and 1, respectively.

Answer group

a) 1 2 1 b) 1 2 2 c) 1 3

d) 1 4 e) 3 1 f) 4 1

[Description of Program 2 (decompress_Stream)]

The decompression process follows the same path as the compression in reverse order, however, with slight difference. One of these is that it does not need the final contents of the dictionary; it needs only the initial contents.

Decompression begins with reading a code from the compressed input stream, looks for the corresponding sub-string in the dictionary, and then writes this sub-string to the output stream. The first character of this sub-string is concatenated to the current string, and the concatenated string is added to the dictionary. This process rebuilds the dictionary that was used in the compression stage. The decoded string becomes the current string, and then, the process iterates.

Here is an execution example. When the compressed result " 1 2 2 4 7 3 " is given from the input stream, then the decompressed string " A B B A B A B A K " is written to the output stream. About this example, a summary of the process is shown in table 4. Assuming that the array `dictionary` used in the program is defined externally, and contains the initial values as shown in Table 1.

Table 4 Summary of the process (Program 2)

oldcode	newcode	Is newcode in the dictionary	s (1st)	*Note (2nd)	String Output	ch	Dictionary Index	Dictionary Entry
1			A		A			
1	2	Yes	B	A	B	B	4	A B
2	2	Yes	B	B	B	B	5	B B
2	4	Yes	A B	B	A B	A	6	B A
4	7	No	A B		A B A	A	7	A B A
7	3	Yes	K	A B A	K	K	8	A BA K

*Note: When "Is newcode in the dictionary" = Yes, s is updated twice.

Thus, the input stream "1 2 2 4 7 3" is decoded as "A B B A B A B A K". It may also be noted that the dictionary was rebuilt with the same entries during the compression stage.

[Program 2]

```
O Program decompress_Stream
O string type: s
O character type: ch
O integer type: oldcode, newcode        /* previously and newly read code */
O procedure: getInStream(code)
                                        /* reads the next code from input stream */
O procedure: putOutStream(string)
                                        /* writes decoded string to output stream */
O procedure returns character type: getFirstChar(s)
                                        /* returns the first character of string s */
O procedure returns integer type: addEntry(string, dictionary)
                                /* adds string into dictionary and returns its index position */
O procedure returns string type: getString(code, dictionary)
                                /* returns the string at the index position code in dictionary */
O procedure returns integer type: codeExists(code, dictionary)
                                /* if code exists in dictionary, returns 1; else returns 0 */
```

- getInStream(oldcode)
- s ← getString(oldcode, dictionary)
- putOutStream(s)

```
• getInStream(newcode)
■ newcode ≠ ''              /* loop while the next code exists (not end of data) */
  ▲ codeExists(newcode, dictionary) = 1
    • s ← getString( [ C ] , dictionary)
    • putOutStream(s)
    • ch ← getFirstChar(s)
    • s ← getString( [ D ] , dictionary)
  ──────────────────────────────────────
    • s ← getString(oldcode, dictionary)
    • ch ← getFirstChar(s)
    • putOutStream( [ E ] )
  ▼
  • addEntry(s+ch, dictionary)          /* "+" means "concatenation" */
  • oldcode ← newcode
  • getInStream(newcode)
■
```

Subquestion 2

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

Answer group

a) ch
b) ch + s
c) s
d) s + ch
e) newcode
f) oldcode

Concerning questions **Q7** and **Q8**, **select one** of the two questions.
Then, mark (S) in the selection area on the answer sheet, and answer the question.
If two questions are selected, only the first question will be graded.

Q7. Read the following description of a C program and the program itself, and then answer Subquestion.

Ordinary notation for writing expressions is called infix, where the operator is placed between two values. Another notation for expressions, one that is useful for stack-oriented evaluation, is called Reverse Polish notation, where the operator is placed after two values. Here are some examples:

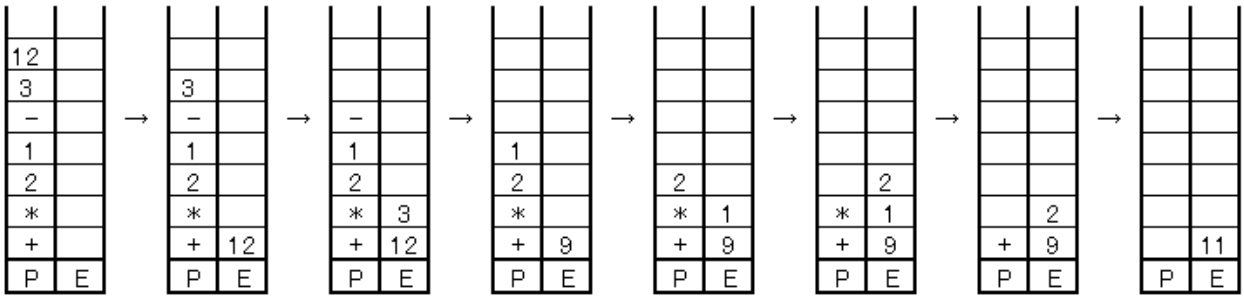
Ordinary notation		Reverse Polish notation
`1 + 2`	→	`1 2 +`
`1 * 2 + 3`	→	`1 2 * 3 +`
`(1 + 2) * (3 + 4)`	→	`1 2 + 3 4 + *`

[Program Description]

The program evaluates the Reverse Polish expression using the two-stack algorithm.
Polish stack (hereinafter, P) contains the Reverse Polish expression, and the Evaluation stack (hereinafter, E) stores the intermediate values during execution.
Here is the two-stack algorithm that evaluates a Reverse Polish expression.

Step 1. If P is empty, stop execution. The answer is obtained at the top of E.
Step 2. If P is not empty, pop P and store it into `d`. (`d`, `d1` and `d2` are used to hold data)
Step 3. If `d` is a value, push it into E.
Step 4. If `d` is an operator, pop E twice, store the first one into `d2` and the second one into `d1`. Execute operation `d` using two values `d1` and `d2`, and push the result into E. Go to step 1.

The program evaluates the expression "`12 3 - 1 2 * +`", and outputs the answer `11`.
The expression is placed in the character array `str[]`. Each value or operator is separated by comma and space, like "`12, 3, -, 1, 2, *, +`".
This algorithm is illustrated in the following diagram, using the given expression.



The program accepts +, -, and * as valid operators. Assuming that the given expression in `str[]` is correct, and irregular status of stacks will not occur during the execution.

Functions and their purposes used in the program are as follows.

Function	Purpose
`main`	To test the program using pre-defined reverse Polish expression: "`12 3 - 1 2 * +`"
`evaluate`	To evaluate the expression in P
`push`	To push data into the stack
`pop`	To pop data from the stack
`fill`	To store each value and operator into the stack from a character string that contains the expression
`empty`	Returns true boolean value if the stack is empty
`full`	Returns true boolean value if the stack is full
`initialize`	To nullify the stack and set `cnt` to 0

[Program]

```
#include <assert.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>

#define  EMPTY  0
#define  FULL   10000

struct data {
   enum {operator, value} kind;
   union {
      char  op;
      int   val;
   } u;
};

typedef  struct data        data;
typedef  enum {false, true} boolean;

struct elem {
   data         d;
   struct elem  *next;
};

typedef  struct elem  elem;

struct stack {
   int   cnt;
   elem  *top;
};

typedef  struct stack  stack;

boolean  empty(const stack *stk);
int      evaluate(stack *polish);
void     fill(stack *stk, const char *str);
boolean  full(const stack *stk);
void     initialize(stack *stk);
data     pop(stack *stk);
void     push(data d, stack *stk);

char    str[] = "12, 3, -, 1, 2, *, +";
```

```
void  main(void)
{
   stack  polish;

   printf("\n%s%s\n","Reverse Polish Expression: ", str);
   fill(&polish, str);
   printf("\n%s%d\n", "Evaluated Value: ", evaluate(&polish));
   return;
}

int evaluate(stack *polish)
{
   data   d, d1, d2;
   stack  eval;

   initialize(&eval);
   while (!empty(polish)) {
      d = [      A      ] ;
      switch (d.kind) {
      case value:
         push(d, &eval);
         break;
      case operator:
         d2 = pop(&eval);
         d1 = [      B      ] ;
         d.kind = value;
         switch (d.u.op) {
         case '+':
            d.u.val = d1.u.val + d2.u.val;
            break;

         case '-':
            d.u.val = d1.u.val - d2.u.val;
            break;
         case '*':
            d.u.val = d1.u.val * d2.u.val;
         }
         [      C      ] ;
      }
   }
   d = pop(&eval);
   return d.u.val;
}
```

```
void push(data d, stack *stk)
{
   elem  *p;

   p = malloc(sizeof(elem));
   p -> d = d;
   [      D      ];
   stk -> top = p;
   [      E      ];
}

data pop(stack *stk)
{
   data   d;
   elem   *p;

   d = stk -> top -> d;
   p = stk -> top;
   stk -> top = stk -> top -> next;
   [      F      ];
   free(p);
   return d;
}

void fill(stack *stk, const char *str)
{
   const char  *p = str;
   char        c1, c2;
   boolean     b1, b2;
   data        d;
   stack       tmp;

   initialize(stk);
   initialize(&tmp);

   while (*p != '\0') {
      while (isspace(*p) || *p == ',')
          /* isspace  returns non-zero value if *p is a white-space character  */
         ++p;
      b1 = (boolean) ((c1 = *p) == '+' || c1 == '-' || c1 == '*');
      b2 = (boolean) ((c2 = *(p + 1)) == ',' || c2 == '\0');
      if (b1 && b2) {
         d.kind = operator;
```

```
            d.u.op = c1;
        }
        else {
            d.kind = value;
            assert(sscanf(p, "%d", &d.u.val) == 1);
        /* assert evaluates the expression; if it is true, then continue, else abort */
        }
        if (!full(&tmp))
            push(d, &tmp);
        while (*p != ',' && *p != '\0')
            ++p;
    }

    while (!empty(&tmp)) {
        d = pop(&tmp);
        if (!full(stk))
            push(d, stk);
    }
}

boolean empty(const stack *stk)
{
    return ((boolean) (stk -> cnt <= EMPTY));
}

boolean full(const stack *stk)
{
    return ((boolean) (stk -> cnt >= FULL));
}

void initialize(stack *stk)
{
    stk -> cnt = 0;
    stk -> top = NULL;
}
```

Subquestion

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

Answer group for A through C

a) `pop(eval)`
b) `pop(&eval)`
c) `pop(polish)`
d) `pop(&polish)`
e) `pop(d2)`
f) `push(d, &eval)`
g) `push(d, &polish)`
h) `push(d2, &eval)`

Answer group for D and E

a) `p -> next = d`
b) `p -> next = stk -> top`
c) `stk -> cnt`
d) `stk -> cnt++`
e) `stk -> top = p -> next`
f) `stk -> top = *p -> next`

Answer group for F

a) `stk -> cnt++`
b) `stk -> cnt--`
c) `stk -> top = p -> next`
d) `stk -> top = *p -> next`

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

[Program Description]

The program calculates and prints out volume, weight and cost for shipment of boxes.
In this calculation, values of the width, height, depth, weight and cost of the Box will be created by constructor, and data will be stored using multi-level inheritance hierarchy of class. Also, in the task, `super()` method is used in the class hierarchy, so all construction methods of the super class will be called.

The program consists of methods of a base class and its derived class, and `main()` method. The class `Box` has 4 constructor functions including silent, with parameter, clone (copy), and cube. `volume()` function for the calculation of volume of a box is defined by variables `width`, `height` and `depth`. Data is transferred by constructor functions in `main()`.

When executing this program, the following list will be printed out.

```
Volume of Shipment1 : 2000.0
Weight of Shipment1 : 12.0
Cost of Shipment1 : $5.56

Volume of Shipment2 :1000.0
Weight of Shipment2 :3.65
Cost of Shipment2 : $2.5

Volume of Shipment3 :2000.0
Weight of Shipment3 :12.0
Cost of Shipment3 : $5.56

Volume of Shipment4 :-1.0
Weight of Shipment4 :-1.0
Cost of Shipment4 : $-1.0
```

[Program]

```
class Box {
   private double width;
   private double height;
   private double depth;
   Box(      A      ) {
      width = ob.width;
      height = ob.height;
      depth = ob.depth;
   }
   Box(double w, double h, double d) {
      width = w;
      height = h;
      depth = d;
   }
   Box() {
      width = -1;
      height = -1;
      depth = -1;
   }
   Box(      B      ) {
      width = height = depth = len;
   }
   double volume() {
      return width * height * depth;
   }
}
class BoxWeight extends      C      {
   double weight;
   BoxWeight(      D      ) {
      super(ob);
      weight =      E      ;
   }
   BoxWeight(double w, double h,double d, double m) {
      super(w, h, d);
      weight = m;
   }
   BoxWeight() {
      super();
      weight = -1;
   }
   BoxWeight(      F      ) {
      super(len);
```

```
      weight = m;
   }
}
class [      G      ] extends BoxWeight {
   double cost;
   Shipment(Shipment ob) {
      super(ob);
      cost = ob.cost;
   }
   Shipment(double w, double h, double d, double m, double c) {
      super(w, h, d, m);
      cost = c;
   }
   Shipment() {
      super();
      cost = -1;
   }
   Shipment(double len, double m, double c) {
      super( len, m );
      cost = c;
   }
}
class ShipmentSystem {
   public static void main(String args[]) {
      Shipment shipment1 = new Shipment(10, 20, 10, 12, 5.56);
      double vol;
      vol = shipment1.volume();
      System.out.println("Volume of Shipment1 :" + vol);
      System.out.println("Weight of Shipment1 :" + shipment1.weight);
      System.out.println("Cost of Shipment1 : $" + shipment1.cost);
      System.out.println();

      Shipment shipment2 = new Shipment(10, 3.65, 2.5);
      vol = shipment2.volume();
      System.out.println("Volume of Shipment2 :" + vol);
      System.out.println("Weight of Shipment2 :" + shipment2.weight);
      System.out.println("Cost of Shipment2 : $" + shipment2.cost);
      System.out.println();

      Shipment shipment3 = new Shipment(shipment1);
      vol = shipment3.volume();
      System.out.println("Volume of Shipment3 :" + vol);
      System.out.println("Weight of Shipment3 :" + shipment3.weight);
      System.out.println("Cost of Shipment3 : $" + shipment3.cost);
```

```
        System.out.println();

        Shipment shipment4 = new Shipment();
        vol = shipment4.volume();
        System.out.println("Volume of Shipment4 :" + vol);
        System.out.println("Weight of Shipment4 :" + shipment4.weight);
        System.out.println("Cost of Shipment4 : $" + shipment4.cost);
    }
}
```

Subquestion

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

Answer group for A and B

a) `Box ob`
b) `BoxWeight ob`
c) `double len`
d) `double w, double h, double d`
e) `len`
f) `ob`
g) `Shipment ob`
h) `void`

Answer group for C and G

a) `Box`
b) `Box()`
c) `BoxWeight`
d) `BoxWeight()`
e) `Shipment`
f) `Shipment()`
g) `ShipmentSystem`
h) `ShipmentSystem()`

Answer group for D and F

a) `Box ob`
b) `BoxWeight ob`
c) `double len`
d) `double len, double m`
e) `super(len)`
f) `super(len, m)`
g) `super(len, m, c)`
h) `super(w, h, d)`

Answer group for E

a) `depth`
b) `height`
c) `ob.depth`
d) `ob.height`
e) `ob.weight`
f) `ob.width`
g) `weight`
h) `width`