October 2010

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 autumn Fundamental IT Engineer Examination conducted?

Answer group

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

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

[Sample Answer]

Sample	Ⓐ	●	Ⓒ	Ⓓ

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

Inquiries about the exam questions will not be answered.

Notations used for pseudo-language

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

[Declaration, comment, and process]

Notation		Description
○		Declares names, types, etc. of procedures, variables, etc.
/* text */		Describes comments in 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

(continued on next page)

[Operators and their priorities]

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

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

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

Q1. Read the following description concerning the quality characteristics of software products, and then answer Subquestion.

ISO/IEC 9126-1 defines six quality characteristics, shown in Table 1, concerning the quality of software products.

Table 1 Six quality characteristics (ISO/IEC 9126-1)

Quality characteristic	Brief description of characteristic	Quality sub-characteristics (excerpts)
Functionality	A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.	Suitability, Accuracy, Security, Interoperability
Usability	A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.	Operability, Learnability, Understandability
Reliability	A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.	Recoverability, Fault tolerance, Maturity
Efficiency	A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.	Time behavior, Resource behavior
Maintainability	A set of attributes that bear on the effort needed to make specified modifications.	Stability, Analyzability, Testability, Changeability
Portability	A set of attributes that bear on the ability of software to be transferred from one environment to another.	Adaptability, Installability, Replaceability

Among these quality characteristics, it is important to consider reliability, efficiency, maintainability, and portability during the coding phase.

A software development company created an internal standard for program development with consideration for quality characteristics, and established a procedure for reviewing the code of the developed program, in order to improve the quality of its software products. Table 2 below is an example of comments given to a new program developer during a recent code review.

Table 2 Example of comments given to a new program developer

Source code	Comment	Key quality characteristic (quality sub-characteristic)
○ Real: Ave, Count, Total ⋮ • Ave ← Total ÷ Count ⋮	Change the process as follows: ⋮ ▲ Count > 0 • Ave ← Total ÷ Count • Ave ← 0 ▼ ⋮	A
⋮ ■ r: 1, r ≤ rMax, 1 ■ c: 1, c ≤ cMax, 1 ▲ X[r, c] > Sub(r) + 1 · X[r, c] ← Sub(r) + 1 ▼ ■ ■ ⋮	In this process, function Sub takes a long time to complete, but its return value is dependent on its argument only. Optimize the process as follows: ⋮ B ⋮	Efficiency (Time behavior)
⋮ /* Dynamic allocation of main memory */ • GetMain(Addr, Len) ⋮ /* Dynamic releasing of main memory */ • FreeMain(Addr) ⋮	The program uses standard functions provided by the system to allocate and release main memory dynamically. In general, for failures such as an accidental update of memory area already released or outside of the allocated range, C . Use the following internally-developed functions that are equivalent in function but also offer debugging functionality: ⋮ • X_GetMain(Addr, Len, ⋯) ⋮ • X_FreeMain(Addr, ⋯) ⋮	Maintainability (Testability)
○ Integer: P1, P2, Ans ○ Integer function: Fn(P1, P2) ⋮ • Ans ← Fn(P1, P2) ⋮	This program will be used generally in multiple computer models. Since D may be different for each model, the declaration statements must be changed as follows: ○ 32 bit integer: P1, P2, Ans ○ 32 bit integer function: Fn(P1, P2) ⋮	E

Subquestion

From the answer groups below, select the correct answers to be inserted into the blanks □ in Table 2.

Answer group for A and E

a) Portability (Adaptability)

b) Efficiency (Resource behavior)

c) Reliability (Maturity)

d) Maintainability (Analyzability)

e) Maintainability (Changeability)

Answer group for B

a)
```
■ r: 1, r ≤ rMax, 1
|  ■ c: 1, c ≤ cMax, 1
|  |  • S1 ← Sub(r) + 1
|  |  ▲ X[r, c] > S1
|  |  |  • X[r, c] ← S1
|  |  ▼
|  ■
■
```

b)
```
■ r: 1, r ≤ rMax, 1
|  • S1 ← Sub(r) + 1
|  ■ c: 1, c ≤ cMax, 1
|  |  ▲ X[r, c] > S1
|  |  |  • X[r, c] ← S1
|  |  ▼
|  ■
■
```

c)
```
■ r: 1, r ≤ rMax, 1
|  ■ c: 1, c ≤ cMax, 1
|  |  • S1 ← Sub(r) + 1
|  |  • Xrc ← X[r, c]
|  |  ▲ Xrc > S1
|  |  |  • Xrc ← S1
|  |  ▼
|  ■
■
```

d)
```
■ r: 1, r ≤ rMax, 1
|  • S1 ← Sub(r) + 1
|  ■ c: 1, c ≤ cMax, 1
|  |  • Xrc ← X[r, c]
|  |  ▲ Xrc > S1
|  |  |  • Xrc ← S1
|  |  ▼
|  ■
■
```

Answer group for C

a) failures can be detected when the data is updated, but few OS's have a function to log this problem

b) failures can be detected when the data is updated, but few hardware have a function to log this problem

c) failures are often detected when the updated data is referenced later, making it difficult to determine which code caused the problem

d) failures cannot be detected while there are allocatable main memory left

Answer group for D

a) the number of variables and functions that can be specified

b) the number of bits when they are omitted from type declarations of variables or functions

c) the number of bits of functions that can be handled by linkers

d) the number of functions that can be handled by loaders

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

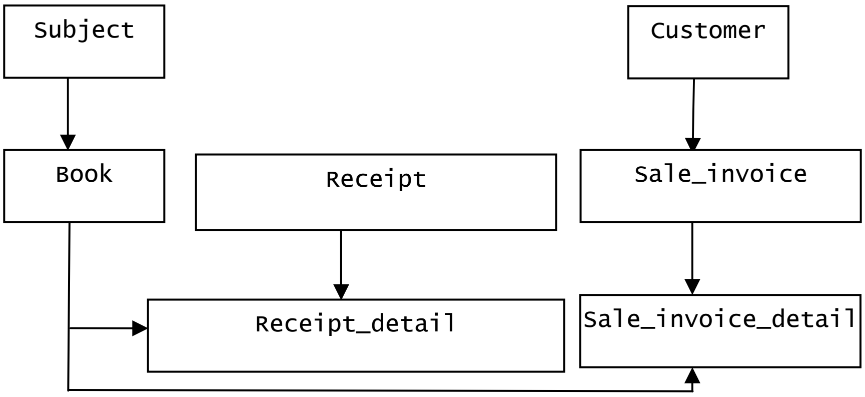
A commission book agency receives the books from publishers and sells these books to customers.

The money received from the sales by commission is 1% of total amount of the price of the books.

When receiving the books, receive_date, book_id, book_name, subject_id, subject_name, receive_quantity, and price should be recorded by the staff.

When selling the books, sale_date, book_id, book_name, subject_id, subject_name, sale_quantity, price, customer_id, customer_name, and customer_address should be recorded by the staff.

Structure of the tables is as follows.



Here, Subject, Book, and Customer are master tables containing information about all book's subjects, books, and customers, respectively.

Sale_invoice, Sale_invoice_detail are tables containing sale information when customers buy the books.

Receipt, Receipt_detail are tables containing information about the books when the staff receives the books from publishers.

Subquestion 1

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

The list below shows the fields and primary keys of the tables.

Subject table:
  Fields: `subject_id, subject_name`
  Primary key: `subject_id`
Book table:
  Fields: `book_id, book_name, price,` $\boxed{\text{A}}$
  Primary key: `book_id`
Customer table:
  Fields: `customer_id, customer_name, customer_address`
  Primary key: `customer_id`
Receipt table:
  Fields: `receipt_id, receive_date`
  Primary key: `receipt_id`
Receipt_detail table:
  Fields: `receipt_id, no, receive_quantity,` $\boxed{\text{B}}$
  Primary key: `receipt_id, no`
Sale_invoice table:
  Fields: `invoice_id, sale_date,` $\boxed{\text{C}}$
  Primary key: `invoice_id`
Sale_invoice_detail table:
  Fields: `invoice_id, no, sale_quantity,` $\boxed{\text{B}}$
  Primary key: `invoice_id, no`

Answer group:

a) `book_id` b) `customer_id` c) `customer_name`
d) `invoice_id` e) `price` f) `receipt_id`
g) `sale_quantity` h) `subject_id` i) `subject_name`

Subquestion 2

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

The following SQL statement lists all subjects, corresponding number of the books, and commission received from the sales of these books, during the specified time from the beginning date to the end date. The list is arranged in the descending order of commission. Here, "`:BEGINDATE`" and "`:ENDDATE`" are host variables which contain the beginning date and the end date.

```
SELECT  Subject.subject_id, subject_name,
   [      D      ], SUM(sale_quantity * price)/100
FROM  Subject, Book, Sale_invoice, Sale_invoice_detail
WHERE  Subject.subject_id = Book.subject_id
   AND Sale_invoice_detail.invoice_id = Sale_invoice.invoice_id
   AND [      E      ]
GROUP BY [      F      ]
ORDER BY [      G      ]
```

Answer group for D

- a) `AVG(sale_quantity)`
- b) `COUNT(sale_quantity)`
- c) `MAX(sale_quantity)`
- d) `SUM(sale_quantity)`

Answer group for E

- a) `Book.book_id = Sale_invoice.book_id AND`
  `sale_date between :BEGINDATE and :ENDDATE`
- b) `Book.book_id = Sale_invoice_detail.book_id AND`
  `sale_date between :BEGINDATE and :ENDDATE`
- c) `Book.book_id = Subject.book_id AND`
  `sale_date between :BEGINDATE and :ENDDATE`
- d) `sale_date between :BEGINDATE and :ENDDATE`

Answer group for F and G

- a) `(sale_quantity * price)/100 DESC`
- b) `Subject.subject_id`
- c) `Subject.subject_id, subject_name`
- d) `SUM(sale_quantity * price)/100`
- e) `SUM(sale_quantity * price)/100 DESC`

Q3. Read the following description concerning Hamming code, and then answer Subquestion.

Hamming code provides a practical solution for error detection and error correction in data transmission. Hamming code can be applied to data bits of any length, and uses the relationship between data bits and parity bits. For example, a 7-bit ASCII code requires 4 parity bits that can be added to the end of data bits or interspersed with original data bits. Figure 1 shows 7 data bits and 4 parity bits. Parity bits are placed in bit positions 1, 2, 4 and 8. In Hamming code, each p bit is the even parity bit for one combination of data bits, as shown below:

p_1: covers bit positions 1, 3, 5, 7, 9, 11
p_2: covers bit positions 2, 3, 6, 7, 10, 11
p_4: covers bit positions 4, 5, 6, 7
p_8: covers bit positions 8, 9, 10, 11

11	10	9	8	7	6	5	4	3	2	1
d	d	d	p_8	d	d	d	p_4	d	p_2	p_1

Figure 1. Positions of parity bits in Hamming code

Bit errors in transmitted data can be detected by testing all of the parity bits in the transmitted data. Assuming that 11-bit data in Hamming code as shown in Figure 1 was transmitted. If all 4 parity bits are correct, there is no error. But, if the parity bits p_8 and p_2 indicate parity errors, then bit position 8+2=10 is in error. Thus, Hamming code can correct bit errors.
Note: The correction can be done for 1-bit errors only.

Subquestion

From the answer groups below, select the correct answers to be inserted into the blanks □ in the following description.
Assuming that Hamming codes are configured as shown in Figure 1.

(1) The Hamming code constructed from 7-bit data 001 1101 is $\boxed{\text{A}}$.

(2) The Hamming code 110 0111 0111 was received. The code indicates that there is an incorrect bit. The correct Hamming code is $\boxed{\text{B}}$.

(3) The Hamming code 011 1010 1101 was received. The code indicates that there is an incorrect bit. The correct original 7-bit data is $\boxed{\quad\text{C}\quad}$.

(4) The Hamming code constructed from 7-bit data 100 1101 was received. The parity bits p_4, p_2 and p_1 indicate parity errors. The received Hamming code was $\boxed{\quad\text{D}\quad}$.

Answer group for A

a) 001 0110 0101

b) 001 0110 0111

c) 001 1110 0111

d) 001 1110 1111

Answer group for B

a) 110 0100 0111

b) 110 0110 0111

c) 110 0110 1111

d) 110 0111 0111

Answer group for C

a) 011 0101

b) 011 0111

c) 111 0101

d) 111 0111

Answer group for D

a) 011 0001 1010

b) 100 0010 1111

c) 100 1010 0101

d) 110 1110 0111

Q4. Read the following description concerning information security controls, and then answer Subquestion.

Alpha publishing company employs 100+ people and publishes at least 50 new books every year. The personnel primarily consist of editors, artists, authors, marketing staff and clerical staff. All personnel work in the office except for authors. Authors are contracted by editors to write books outside the office for a given time. Alpha publishes, prints and sells their books to schools and bookstores. Alpha's website is constantly updated to contain information about recent and upcoming books. The collection of book titles and sampler of its content is available at the website. The website also allows customers to inquire by chatting or sending emails. Table 1 shows the company's security environment.

Table 1. The company's security environment

Work / Materials	Security Requirements	Security Concerns
Internal business information	Confidentiality	Leakage of confidential information. Social engineering attack to enter more sensitive parts of the system.
Manuscript, edited work and cover design	Integrity, Confidentiality, Accountability	Leakage of softcopies to the public before official release (losing revenues). Insufficient change controls for revisions of the work.
Marketing presentations	Availability	Availability of materials for viewing through the website and for marketing presentations in different locations.

[Work flow]

(1) Authors write their books using their personal computers. They submit their work to the editor through their public email, or by going to the office and giving the files using a portable storage device.

(2) The editor and artists work on the manuscript to incorporate artwork, layout, cover design and other elements to complete the book. The author is regularly updated of the work by sending him/her information on the designs through email.

(3) A marketing staff is assigned to the project to work on all the marketing materials. They design road show presentations and the corresponding web page of the book.

(4) Once the editorial work is done, the book goes to the printing press for initial printing, then distributed to the respective customers.

The company established the security measures as follows, and summarized the evaluation of the threats as in Table 2 and the security controls for the threats as in Table 3.

[Security measures]

(1) Employees make use of IDs with RFID. An RFID system is used for identifying their entry and exit in secured areas.

(2) Employees use workstations. It's the policy of the company that project-related files and important business data are stored in servers, while clerical, communications and other files are stored in their workstation. The servers are located in a separate, secured and temperature controlled room with fire suppression system.

(3) It's a policy that employees cannot bring in or out any files. Employees are not allowed to bring portable storage devices. USB ports are deactivated and files are brought in or out through a controlled terminal with manager clearance.

(4) A router with a firewall with configured stateful packet inspection is installed for the Internet connection. The network is segmented into workgroups with NAT, and there is no WiFi access in the office.

(5) CCTV cameras are installed at the entry points of the office and at the server room. Recorded data is kept in DVDs for at least one year.

(6) The backup machines in the server room do daily backup of all the files in the servers.

(7) The company instituted a weekly patch management cycle for all workstations every Friday at 4PM.

Table 2. Evaluation of threats

	Threats	Damage	Current control levels and security issues	Risk Level / Frequency
I.1	System failure	Interruption of work	Servers do not have any stand-by	Medium / Low
I.2	Espionage, Sabotage, Theft of works or manuscript	Loss of revenues from compromised work	Employee trust is preferred over delays	Medium / Low
I.3	Spread of virus	Interruption of work, Revert to back-up	Virus often brought in by authors through portable storage devices or files sent	High / Medium
E.1	Fire, Natural calamities	Loss of property, Loss of files	Office is not proofed against earthquake	High / Low
E.2	Internet attackers, crackers and worms	Interruption of work, Leakage of files	Monitored and protected	Medium / Medium
E.3	Burglar - Theft of equipment	Interruption of work, Leakage of files	Monitored and protected	High / Low
E.4	Theft of works or manuscript	Loss of revenues from compromised work	Servers in the DMZ are not well protected	High / Medium

(Note) I.1 - I.3: Internal threats; E.1 - E.4: External threats.

Table 3. Security controls for threats

	Threats	Detection	Prevention	Response
I.1	System failure	24-hour system monitoring	As item $\boxed{\text{ A }}$ in [Security measures]	Diagnosis and repair
I.2	Espionage, Sabotage, Theft of works or manuscript	CCTV and RFID	CCTV and RFID Also as item (5) in [Security measures]	Identify source and terminate employment
I.3	Spread of virus	Detecting automatically	As item $\boxed{\text{ B }}$ in [Security measures]	Report to security manager
E.1	Fire, Natural calamities	(Not planned)	(Not planned)	Report to local government
E.2	Internet attackers, crackers and worms	Monitoring of suspicious in-bound network traffic	As item $\boxed{\text{ C }}$ in [Security measures]	Report to security manager
E.3	Burglar - Theft of equipment	CCTV and RFID	CCTV and RFID	Report to local authorities
E.4	Theft of works or manuscript	When theft or loss is reported	Protecting servers in the DMZ	Identify and charge offenders

Subquestion

From the answer group below, select the appropriate answers to be inserted into the blanks $\boxed{\quad\quad\quad}$ in Table 3 and the blanks $\boxed{\quad\quad\quad}$ in the following description.

The items (1) through (7) in [Security measures] can be classified under 3 categories; physical access controls, logical access controls, and media controls.
The items (1), $\boxed{\text{ D }}$ and $\boxed{\text{ E }}$ are related to physical access controls. Among these 3 items, the item $\boxed{\text{ E }}$ mainly aims at detection, and others mainly aim at prevention.
The item related to media controls is $\boxed{\text{ F }}$.

Answer group

a) (1)	b) (2)	c) (3)	d) (4)
e) (5)	f) (6)	g) (7)	

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

A company develops a payroll system for its employees' monthly payment. Payment records of the last month are kept in Old PayData file. Whenever promotion is awarded to employees or appointment is made for new employees, new records are created in a file called Promotion file. Every month, Old PayData file is updated with Promotion file, and New PayData file is generated. Then, payroll calculation is done using New PayData file.

The record format of Old PayData file and Promotion file, with sample data, are as following. Old PayData record and New PayData record have the same format.

Old PayData (with sample data)

EmpID	BasicPay	Increment	Bonus	LoanDeduction	TravelAllowance
0002	30000	3000	5000	0	10000
0004	30000	3000	5000	2000	10000
0006	50000	5000	8000	0	10000
0008	40000	4000	5000	1000	10000

Promotion (with sample data)

EmpID	BasicPay	Increment	Bonus
0004	40000	4000	5000
0005	30000	3000	5000
0006	60000	6000	8000

New PayData (with resulted data derived from the above sample data)

EmpID	BasicPay	Increment	Bonus	LoanDeduction	TravelAllowance
0002	30000	3000	5000	0	10000
0004	40000	4000	5000	2000	10000
0005	30000	3000	5000	0	10000
0006	60000	6000	8000	0	10000
0008	40000	4000	5000	1000	10000

The conditions and the procedures to generate New PayData file by matching and updating of Old PayData file with Promotion file is as follows.

Conditions:

1. There is no duplicated EmpIDs in Old PayData file and Promotion file. Range of EmpID is from 0001 to 9999.
2. Old PayData file and Promotion file are sorted in ascending order of the EmpID.

Pre Process:

1. Open all files. (executed implicitly)
2. Read a record from Old PayData file, and read a record from Promotion file.

Iteration Process:

1. Compare the EmpIDs of Old PayData record with Promotion record.
2. If they are equal, update the Old PayData record with Promotion record by simply replacing BasicPay, Increment and Bonus from Promotion record. LoanDeduction and TravelAllowance in the Old PayData record remain unchanged. Then, write updated record to New PayData file.
3. If EmpID exists only in Old PayData record, current Old PayData record is copied to New PayData file.
4. If EmpID exists only in Promotion record, write the data in Promotion record to New PayData file where LoanDeduction is set to 0 and TravelAllowance is set to 10000.
5. Exit the loop when both Old PayData file and Promotion file reach end of file.

Post Process:

1. Close all files. (executed implicitly)

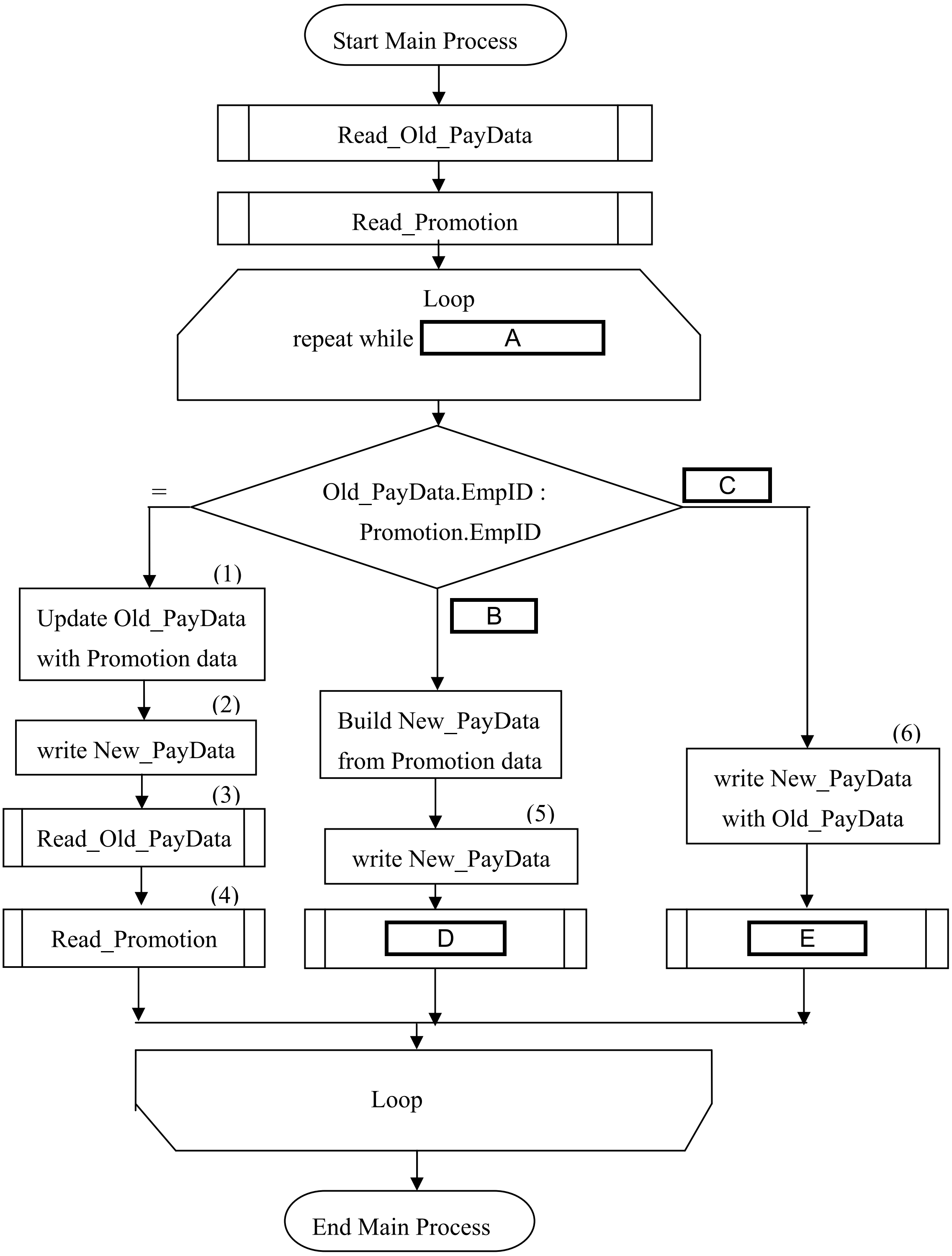
In the above program design, reading a record from Old PayData file is designed as a separate procedure called Read_Old_PayData, and reading a record from Promotion file is designed as a separate procedure called Read_Promotion.

In each of these procedures, if a file reaches end of file, then end-of-file flag EOF is set to 'Yes' (initial value is 'No') and EmpID is set to 999999 (high-value).

Subquestion 1

The following flowchart shows the procedure to generate New PayData file by matching and updating of Old PayData file with Promotion file.

From the answer groups below, select the correct answers to be inserted into the blanks □ in the following flowchart.



Note: Marks (1) through (6) at the upper right of process boxes are reference numbers used in Subquestion 2.

Answer group for A

a) Old_PayData.EOF = ‘No’ AND Promotion.EOF = ‘No’
b) Old_PayData.EOF = ‘No’ OR Promotion.EOF = ‘No’
c) Old_PayData.EOF = ‘Yes’ AND Promotion.EOF = ‘Yes’
d) Old_PayData.EOF = ‘Yes’ OR Promotion.EOF = ‘Yes’

Answer group for B through E

a) >   b) <
c) Read_Old_PayData   d) Read_Promotion

Subquestion 2

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

It is decided to change part of the condition and the iteration process as follows. Changed parts are indicated by underlines.

Condition:

1. There is no duplicated EmpIDs in Old PayData file, but duplicated EmpIDs may possibly exist in Promotion file. Range of EmpID is from 0001 to 9999.

Iteration Process:

2. If they are equal, update the Old PayData record with the non-blank data in the Promotion record. All other data in the Old PayData record remain unchanged. Then, write updated record to New PayData file.

For example, given the following sample data.

Old PayData (with sample data)

EmpID	BasicPay	Increment	Bonus	LoanDeduction	TravelAllowance
0004	30000	3000	5000	2000	10000
0006	50000	5000	8000	0	10000

Promotion (with sample data)

EmpID	BasicPay	Increment	Bonus
0004		4000	
0004	40000		

The first Promotion record updates the Increment of EmpID 0004 in Old PayData file to 4000, then the second Promotion record updates the BasicPay of EmpID 0004 in Old PayData file to 40000. So the resulted New PayData file is as follows.

New PayData (with resulted data derived from the above sample data)

EmpID	BasicPay	Increment	Bonus	LoanDeduction	TravelAllowance
0004	40000	4000	5000	2000	10000
0006	60000	6000	8000	0	10000

To reflect the changes described above, the flowchart must be changed as follows.

1. Change the detailed process of the process box (1).
2. Remove $\boxed{\text{F}}$.

Answer group for F

a) process box (2)

b) process boxes (2) and (3)

c) process boxes (3) and (6)

d) process boxes (4) and (5)

e) process boxes (4) and (6)

f) process box (5)

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

Heap is a complete binary tree which satisfies that the value of every internal node is greater (or less) than or equal to the values of its children. This is called the heap property. Figure 1a shows an example of a heap with 10 nodes.

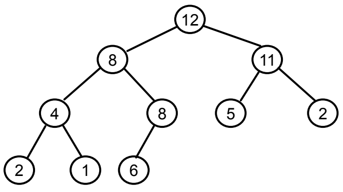


Fig 1a. Max heap

In this program, a heap is implemented in an array *T* that is indexed from 1 to N, where N is the number of elements in the heap. The root of the heap is located at *T*[1], and for each index *i*, element *T*[*i*] has children *T*[2*i*] and *T*[2*i*+1], and the parent *T*[*i*/2] (if either or both children are present). Figure 1b shows the array representation for the heap shown in Figure 1a.

Index	1	2	3	4	5	6	7	8	9	10
Value	**12**	**8**	**11**	**4**	**8**	**5**	**2**	**2**	**1**	**6**

Fig 1b. Array representation

The crucial characteristic of this data structure is that the heap property can be restored efficiently if the value of a node is modified.

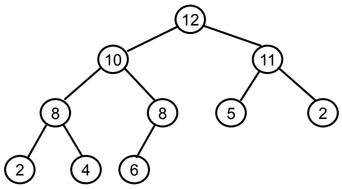


Fig 2. Shifting up

If the value of a node increases to the extent that it becomes greater than the value of its parent, it suffices to exchange these two values, and then continues the same process upwards in the tree if necessary until the heap property is restored. This operation is called shifting up. For example, if the value 1 in Figure 1a is modified so that it becomes 10, the heap property can be restored by exchanging 10 with its parent 4, and exchanging it again with its new parent 8, obtaining the result shown in Figure 2.

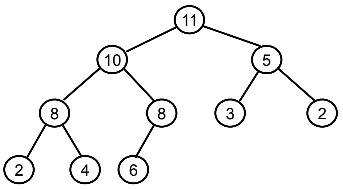


Fig 3. Shifting down

To the contrary, if the value of a node is decreased so that it becomes less than the value of at least one of its children, it suffices to exchange the modified value with the larger of the values of its children, and then continues this process downwards in the tree if necessary until the heap property is restored. This operation is called shifting down. For example, if the value 12 in the root of Figure 2 is modified so that it becomes 3, the heap property can be restored by exchanging 3 with its larger child 11, and exchanging it again with its new larger child 5, obtaining the result shown in Figure 3.

There are 3 subprograms: `alter_heap`, `shift_down` and `shift_up`. The subprogram `alter_heap` sets the value `v` in `T[i]`. The subprograms `shift_up` and `shift_down` perform shifting down and shifting up operations respectively. The following table shows the parameters used in these subprograms.

Parameter	Input/Output	Explanation
T[ ]	Input	Integer type array to store elements of heap
n	Input	Number of elements in heap
i	Input	The index of array T indicating the target element or the starting node
v	Input	Integer type value that is set to T[i]

[Program 1]

```
○ Subprogram: alter_heap(T[ ], n, i, v)
/* T[ ] is a heap.  The value v is set to T[i], and the heap property is re-established.  */
/* Assuming that 1 ≤ i ≤ n.  */
○ Integer type: x
• x ← T[i]
• T[i] ← v
▲  v < x
│ • [    A    ]
├──────
│ • [    B    ]
▼

○ Subprogram: shift_up(T[ ], i)
/* Shifts the node i up so as re-establish the heap property in T[ ].  */
/* Assuming that 1 ≤ i ≤ n.  */
○ Integer type: j, k, w
• k ← i
/* If j = k, then the node has arrived at its final position.  */
■ /* post-test iteration process  */
│ • j ← k
│ ▲  j > 1 and T[    C    ] < T[j]
│ │ • k ← [    C    ]
│ │ • w ← T[j]
│ │ • T[j] ← T[k]
│ │ • T[k] ← w
│ ▼
■ j ≠ k
```

```
○ Subprogram: shift_down(T[ ], n, i)
/* Shifts the node i down so as re-establish the heap property in T[ ]. */
/* Assuming that 1 ≤ i ≤ n. */
○ Integer type: j, k, w
• k ← i
/* If j = k, then the node has arrived at its final position. */
■ /* post-test iteration process */
│ • j ← k
│ ▲ 2×j ≤ n and T[2×j] > T[j]
│ │ • k ← 2×j
│ ▼
│ ▲ 2×j < n and T[2×j+1] > T[   D   ]
│ │ • k ← 2×j+1
│ ▼
│ ▲ j ≠ k
│ │ • w ← T[j]
│ │ • T[j] ← T[k]
│ │ • T[k] ← w
│ ▼
■ j ≠ k
```

Subquestion 1

From the answer groups below, select the correct answers to be inserted into the blanks □ in Program 1.

Answer group for A and B

a) shift_down(T[ ], n, 1)
b) shift_down(T[ ], n, i)
c) shift_down(T[ ], n, x)
d) shift_up(T[ ], 1)
e) shift_up(T[ ], i)
f) shift_up(T[ ], x)

Answer group for C and D

a) j ÷ 2
b) 2 × j
c) 2 × j + 1
d) k
e) 2 × k
f) n

It remains to be seen how to create a heap starting from the elements of array `T[ ]` in an undefined order. Suppose, for example, that array `T[ ]` contains the initial values shown in Figure 4a.

Index	1	2	3	4	5	6	7	8	9	10
Value	**1**	**6**	**9**	**2**	**7**	**5**	**2**	**7**	**4**	**10**

Fig 4a. Initial values in array T[]

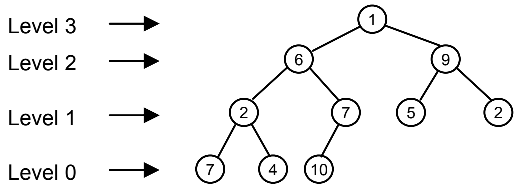


Fig 4b. Initial values

First, make each of the sub-trees, whose roots are at level 1, into a heap. This is done by shifting down these roots, as illustrated in Figure 4c.

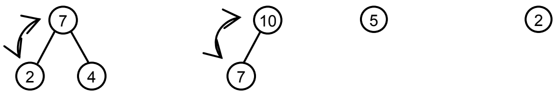


Fig 4c. Level 1 sub-trees are made into heaps

The sub-trees at level 2 are then transformed into heaps, again by shifting down their roots. Figure 4d shows the process for the left sub-tree.

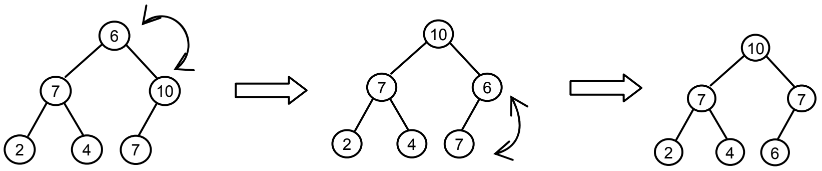


Fig 4d. Level 2 left sub-tree is made into a heap

The right sub-tree at level 2 is already a heap. The contents of array `T[ ]` at this point are as follows.

1	2	3	4	5	6	7	8	9	10
1	**10**	**9**	**7**	**7**	**5**	**2**	**2**	**4**	**6**

Fig 4e. Array representation after the operation in Fig 4d

It only remains to shift down its root to obtain the desired heap. The final process goes as follows.

1	2	3	4	5	6	7	8	9	10
10	**1**	**9**	**7**	**7**	**5**	**2**	**2**	**4**	**6**

1	2	3	4	5	6	7	8	9	10
E									

1	2	3	4	5	6	7	8	9	10
10	**7**	**9**	**4**	**7**	**5**	**2**	**2**	**1**	**6**

Fig 5. Array representation

The subprogram make_heap that realizes this process is shown in Program 2.

[Program 2]

```
○ Subprogram: make_heap(T[ ], n)
/* Makes array T[ ] into a heap. */
○ Integer type: i
• i ← n÷2
■  i > 0
│ • [  F  ]
│ • i ← i-1
■
```

Subquestion 2

From the answer groups below, select the correct answers to be inserted into the blank □ in Figure 5 and the blank □ in Program 2.

Answer group for E

a)

1	10	9	7	5	2	7	2	4	6

b)

10	7	9	1	7	5	2	2	4	6

c)

10	7	9	7	2	5	7	2	4	6

d)

1	10	9	7	7	2	4	2	6	5

Answer group for F

a) alter_heap(T[ ], n, i, T[1])

b) shift_down(T[ ], n, 1)

c) shift_down(T[ ], n, i)

d) shift_up(T[ ], i)

e) shift_up(T[ ], n)

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

This is a C program to keep track of student registration in a class.

[Program Description]

(1) The program aims to keep the data of the students who enroll the course. For each student, the program stores student ID, first name, and last name as a student record.

(2) The program stores all the data in a binary file named “register.dat”. The records are placed in the file in ascending order of absolute value of student ID. The file may contain records having negative student IDs.

(3) The program first calls `select_menu` function to show the menu for a user. The user chooses one of the choices to manipulate the data in the file. The choices are “Add a new student”, “Delete the existing student”, “List all the students”, "Find the student", and “Exit from the program”.

(4) When the user chooses to add a new student, the program calls `add_student` function. The function needs to find the appropriate position to insert a new record, and rearrange the existing records in the file so that there is empty space to insert the new record. The algorithm for adding a new student is as follows.

1) Accept the input student ID.
2) Move the file pointer to the beginning of the file.
3) Read records in the file sequentially to find the position to insert a new record. If the input student ID already exists in the file, stop the operation.
4) Accept the student's first name and last name.
5) Set the insertion position.
6) Move the file pointer to the last record of the file.
7) Read a record and write it to the next position. Then move the file pointer backward to the next record to be moved. Repeat this step until the file pointer reaches the insertion position
8) Write the new record to the insertion position.

(5) When the user chooses to delete the existing student, the program calls `delete_student` function. It does not delete the record physically, but changes the sign of the student ID. Negative value indicates that the record is deleted logically.

(6) When the user chooses to list all the students, the program calls `list_student`

function. It prints out all the records that have positive student IDs.

(7) When the user chooses to find the student, the program calls `find_student` function. It conducts a binary search among the records in the file in order to find the record of input ID.

(8) In order to make it easier to set the file pointer to a specified position, 4 utility functions based on fseek are prepared.

1) `move_from_begin` – move the file pointer forward for the specified number of records from the beginning of the file.
2) `move_from_end` – move the file pointer backward for the specified number of records from the end of the file.
3) `move_forward` – move the file pointer forward for the specified number of records from the current position.
4) `move_backward` – move the file pointer backward for the specified number of records from the current position.

[Program]

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

struct student_record {
   int  id;
   char firstname[20];
   char lastname[20];
};

int select_menu();
void move_from_begin(FILE *ptr, int r);
void move_from_end(FILE *ptr, int r);
void move_forward(FILE *ptr, int r);
void move_backward(FILE *ptr, int r);
void add_student(FILE *ptr);
void delete_student(FILE *ptr);
void list_student(FILE *ptr);
void find_student(FILE *ptr);
```

```
int main() {
   FILE *ptr;
   int choice;

   if ((ptr = fopen("register.dat","r+b")) == NULL) {
      if ((ptr = fopen("register.dat","w+b")) == NULL) {
         fprintf(stderr, "Fail to open file\n");
         return 1;
      }
   }
   while((choice=select_menu()) != 5) {
      switch(choice) {
         case 1:  add_student(ptr);       break;
         case 2:  delete_student(ptr);    break;
         case 3:  list_student(ptr);      break;
         case 4:  find_student(ptr);      break;
      }
   }
   fclose(ptr);
   return 0;
}

int select_menu() {
   int choice;

   printf("Select one option:\n");
   printf("1. Add New Student\n");
   printf("2. Delete Existing Student\n");
   printf("3. List Students\n");
   printf("4. Find Student\n");
   printf("5. Exit\n");
   printf("Enter your option: ");
   scanf("%d", &choice);
   return choice;
}

void move_from_begin(FILE *ptr, int r) {
   fseek(ptr, r*sizeof(struct student_record), SEEK_SET); }

void move_from_end(FILE *ptr, int r) {
   fseek(ptr, -1*r*sizeof(struct student_record), SEEK_END); }

void move_forward(FILE *ptr, int r) {
   fseek(ptr, r*sizeof(struct student_record), SEEK_CUR); }

void move_backward(FILE *ptr, int r) {
   fseek(ptr, -1*r*sizeof(struct student_record), SEEK_CUR); }
```

```
void add_student(FILE *ptr) {
   struct student_record student, temp_record;
   int n, query_id;
   long int insertion_position, p;
   int position_in_between=0;

   printf("Enter a Student ID:");
   scanf("%d", &query_id);
   if (query_id <= 0) {
      fprintf(stderr, "Student ID must be positive number.\n");
      return;
   }
   [      A      ] ;
   while(!feof(ptr) && !position_in_between) {
      n = fread(&student, sizeof(struct student_record), 1, ptr);
      if (n > 0) {
         if ([      B      ]) {
            fprintf(stderr, "ID %d already exists.\n", query_id);
            return;
         }
         else if (-student.id == query_id) {
            move_backward(ptr, 1);
            position_in_between = -1;
         }
         else if (abs(student.id) > query_id) {
            position_in_between = 1;
         }
      }
   }
   printf("Enter firstname and lastname:");
   scanf("%s%s", student.firstname, student.lastname);
   student.id = query_id;
   if (position_in_between==1) {
      insertion_position =
         ftell(ptr) - sizeof(struct student_record);
      [      C      ] ;
      do {
         move_backward(ptr, 1);
         fread(&temp_record, sizeof(struct student_record), 1, ptr);
         fseek(ptr, ftell(ptr), SEEK_SET);
         fwrite(&temp_record, sizeof(struct student_record), 1, ptr);
         [      D      ] ;
      } while(ftell(ptr) > insertion_position);
   }
   fwrite(&student, sizeof(struct student_record), 1, ptr);
}
```

```
void delete_student(FILE *ptr) {
   struct student_record student;
   int n, query_id;

   printf("Enter a Student ID:");
   scanf("%d", &query_id);
   if (query_id <= 0) {
      fprintf(stderr, "Student ID must be positive number.\n");
      return;
   }
   |     A     | ;
   while(!feof(ptr)) {
      n = fread(&student, sizeof(struct student_record), 1, ptr);
      if (n > 0) {
         if (student.id == query_id) {
            student.id = -1*student.id;
            move_backward(ptr, 1);
            fwrite(&student, sizeof(struct student_record), 1, ptr);
            return;
         }
      }
   }
}

void list_student(FILE *ptr) {
   struct student_record student;

   |     A     | ;
   while(!feof(ptr)) {
      if (fread(&student,sizeof(struct student_record), 1, ptr)) {
         if (student.id > 0)
            printf("%2d %-10s%-10s\n", student.id,
                   student.firstname, student.lastname);
      }
   }
}
```

```
void find_student(FILE *ptr) {
   struct student_record student;
   int query_id, left, right, mid;

   printf("Enter a Student ID:");
   scanf("%d", &query_id);
   if (query_id <= 0) {
      fprintf(stderr, "Student ID must be positive number.\n");
      return;
   }
   left = 0;
   C;
   right = ftell(ptr) / sizeof(struct student_record);
   while (left <= right) {
      mid = (left+right)/2;
      move_from_begin(ptr, mid);
      fread(&student, sizeof(struct student_record), 1, ptr);
      if (B) {
         printf("Found the student with ID %d\n", query_id);
         printf("First name: %s\n", student.firstname);
         printf("Last name:  %s\n", student.lastname);
         return;
      }
      else if (E) {
         left = mid + 1;
      }
      else {
         right = mid - 1;
      }
   }
   fprintf(stderr, "The student with ID %d not found\n", query_id);
   return;
}
```

Subquestion

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

Answer group for A, C and D

a)	`move_backward(ptr, 1)`	b)	`move_backward(ptr, 2)`
c)	`move_forward(ptr, 1)`	d)	`move_forward(ptr, 2)`
e)	`move_from_begin(ptr, 0)`	f)	`move_from_begin(ptr, 1)`
g)	`move_from_end(ptr, 0)`	h)	`move_from_end(ptr, 1)`

Answer group for B

a) `student.id < 0`

b) `student.id < query_id`

c) `student.id == 0`

d) `student.id == query_id`

e) `student.id > 0`

f) `student.id > query_id`

Answer group for E

a) `abs(student.id) < query_id`

b) `abs(student.id) <= query_id`

c) `abs(student.id) > query_id`

d) `abs(student.id) >= query_id`

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

[Program Description]

The class AddressBook is created to maintain a collection of Person objects. AddressBook is incorporated with routines for searching, sorting and adding persons. AddressBook is able to sort the Person objects in alphabetical order of their names or in ascending order of their ages. AddressBook uses HaspMap for data management.

[Program]

```
import java.util.*;

class Person {

   public static final int NAME = 0;
   public static final int AGE = 1;
   private static final int LESS = -1;
   private static final int EQUAL = 0;
   private static final int MORE  = 1;
   private static int compareAttribute;
   private String  name;
   private int     age;
   private char    gender;

   static {
      compareAttribute = NAME;
   }

   public Person(String name, int age, char gender) {
      this.age    = age;
      this.name   = name;
      this.gender = gender;
   }

   public static void setCompareAttribute(int attribute) {
      compareAttribute = attribute;
   }

   public int compareTo(Person person, int attribute) {
      int comparisonResult;
   if (attribute == AGE) {
      int p2age = person.getAge();
      if (this.age < p2age) {
         comparisonResult = LESS;
      } else if (this.age == p2age) {
         comparisonResult = EQUAL;
      } else {
         assert this.age > p2age;
         comparisonResult = MORE;
      }
   } else {
      String    p2name = person.getName();
      comparisonResult = this.name.compareTo(p2name);
   }
   return comparisonResult;
}

public int compareTo(Person person) {
   return compareTo(person, compareAttribute);
}

public int getAge() {
   return age;
}

public char getGender() {
   return gender;
}

public String getName() {
   return name;
}

public void setAge(int age) {
   this.age = age;
}

public void setGender(char gender) {
   this.gender = gender;
}

public void setName(String name) {
   this.name = name;
}
```

```
   public String toString()  {
      return this.name    + "\t\t" +
             this.age     + "\t\t" +
             this.gender;
   }

}

interface AddressBook {
   public void add(Person newPerson);
   public Person search(String searchName);
   public Person[ ] sort(int attribute);
}

class AddressBookV1 implements AddressBook {

   private static final int  DEFAULT_SIZE = 25;
   private Map   entry;

   public AddressBookV1() {
      this(DEFAULT_SIZE);
   }

   public AddressBookV1(int size) {
      entry = new HashMap(size);
   }

   public void add(Person newPerson) {
      entry.put(newPerson.getName(),      A      );
   }

   public Person search(String searchName) {
      return (Person)      B      (searchName);
   }

   public Person[ ] sort(int attribute) {
      if (!(attribute == Person.NAME || attribute == Person.AGE)) {
         throw new IllegalArgumentException();
      }
      Person[ ] sortedList = new Person[entry.size()];
      entry.values().toArray(      C      );
      Arrays.sort(sortedList,      D      );
      return sortedList;
   }
   private Comparator getComparator(int attribute) {
      Comparator comp = null;
      if (attribute == 1) {
         comp = new       E       ;
      } else {
         assert attribute == 0:
                "Attribute not recognized for sorting";
         comp = new       F       ;
      }
      return comp;
   }

   class AgeComparator implements Comparator {
      private final int LESS = -1;
      private final int EQUAL = 0;
      private final int MORE  = 1;
      int comparisonResult;

      public int compare(Object p1, Object p2) {
         int p1age = ((Person)p1).getAge();
         int p2age = ((Person)p2).getAge();
         if (p1age < p2age) {
            comparisonResult = LESS;
         } else if (p1age == p2age) {
            comparisonResult = EQUAL;
         } else {
            assert p1age > p2age;
            comparisonResult = MORE;
         }
         return comparisonResult;
      }
   }

   class NameComparator implements Comparator {

      public int compare(Object p1, Object p2) {
         String p1name = ((Person)p1).getName();
         String p2name = ((Person)p2).getName();
         return       G       ;
      }
   }

}
```

Subquestion

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

Answer group for A

a) `entry`
b) `newPerson`
c) `newPerson.getAge`
d) `newPerson.getName`
e) `newPerson.setCompareAttribute(DEFAULT_SIZE)`
f) `null`

Answer group for B

a) `entry.get`
b) `entry.put`
c) `entry.remove`
d) `entry.values`
e) `Person.setName`

Answer group for C and D

a) `AgeComparator.compare()`
b) `DEFAULT_SIZE`
c) `getComparator(attribute)`
d) `Person`
e) `Person.AGE`
f) `Person.NAME`
g) `sortedList`

Answer group for E through G

a) `AgeComparator()`
b) `Comparator()`
c) `compare(p1name, p2name)`
d) `getComparator(Person.NAME)`
e) `NameComparator()`
f) `p1name.compareTo(p1name, comparisonResult)`
g) `p1name.compareTo(p2name)`