



April 2009

Fundamental IT Engineer Examination (Afternoon)

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	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 (Q6-Q7 and Q8-Q9)**

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]




1	Ⓐ	●	Ⓒ	Ⓓ
---	---	---	---	---

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


Inquiries about the exam questions will not be answered.

Company names and product names appearing in the test questions are trademarks or registered trademarks of their respective companies. Note that the ® and ™ symbols are not used within.

[Explanation of the Pseudo-Code Description Format]

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

[Operator]

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

[Logical type constant]

true false

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

Q1 Read the following description of a program and the program itself, and then answer the Subquestion.

[Program Description]

There are three functions named `Preorder`, `Inorder` and `Postorder`. Each function visits the nodes of a binary tree by calling itself recursively, and prints out the node numbers visited. For each function, input parameter `node` indicates a node number. Parameters `node.left` and `node.right` used in each function indicate the next node number on the left side and on the right side respectively. If the next node does not exist, `null` is set for these parameters. In the Figure below, for example, if `node=2`, then `node.left=null` and `node.right=4`.

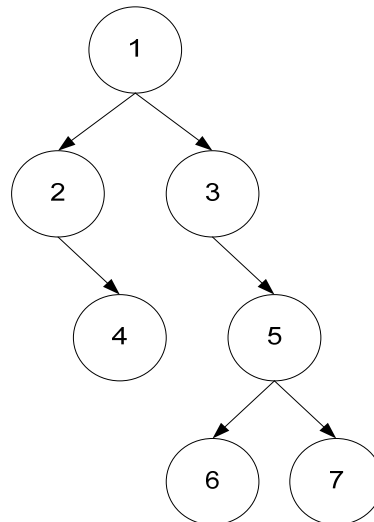


Figure Example of a Binary Tree

[Program]

```
/* Function Preorder */
```

```
O Input parameter: node
```

```
O Function Preorder(node)
```

```
    ▲ node!=null          /* if (node!=null) */
    │
    │   ■ Print "→",node
    │   ■ Preorder(node.left)
    │   ■ Preorder(node.right)
    ▼
```

```
/* Function Inorder */
```

```
O Input parameter: node
```

```
O Function Inorder(node)
```

```
    ▲ node!=null          /* if (node!=null) */
    │   ■ Inorder(node.left)
    │   ■ Print "→",node
    │   ■ Inorder(node.right)
    ▼
```

```
/* Function Postorder */
```

```
O Input parameter: node
```

```
O Function Postorder(node)
```

```
    ▲ node!=null          /* if (node!=null) */
    │   ■ Postorder(node.left)
    │   ■ Postorder(node.right)
    │   ■ Print "→",node
    ▼
```

Subquestion

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

For the binary tree shown in the Figure, when `Preorder(1)` is executed, then A will be printed out as a result. Similarly, when `Inorder(1)` is executed, the result will be B, and when `Postorder(1)` is executed, the result will be C.

Answer group

- a) →1→2→4→3→5→6→7
- b) →1→2→6→4→3→5→7
- c) →2→4→1→3→6→5→7
- d) →2→4→1→6→5→7→3
- e) →4→2→6→7→5→3→1
- f) →4→2→7→6→5→3→1

Q2 Read the following description about planning of production, and then answer the Subquestions 1 through 3.

The enterprise M produces car's tires. It receives customer orders, do planning of a production. For a production of tires, wholly the necessary materials are available on the stock. The crucial materials, like rubber and carbon powder, usually have the grand quantity and have grand number of minimum on stock. Assume that there is no problem with delay time of the orders for the missing materials.

[Procedure Description]

1) The description of Summary Products by Customer is shown in the table SPC.

SPC

TIRE_ID	ORDER_QUANTITY	CUSTOMER_ID

2) The description of tire's types and the quantity of each crucial material (there are 7 materials, named MAT_A, MAT_B, MAT_C, MAT_D, MAT_E, MAT_F and MAT_J are needed to be managed) for the production of an amount of a tire's type is shown in the Table TIRE, as follows:

TIRE

<u>TIRE_ID</u>	PRO_QUANTITY	MAT_A	MAT_B	MAT_C	MAT_D	MAT_E	MAT_F	MAT_J
P185R/70HR1	100	10	30	50	1000	60	80	60

The underlined column's name denotes the primary key.

The columns MAT_A, MAT_B, MAT_C, ..., MAT_J define the needed quantity of the corresponding material for the production of the amount PRO_QUANTITY of the TIRE_ID type of tires.

The above example shows that the production of 100 tires of type "P185R/70HR13" needs 10 units of the Material A, 30 units of the Material B, 50 units of the Material C, ..., and 60 units of the Material J.

Subquestion 1

From the answer group below, select the correct answer to obtain the result table named TotalOrder as shown below. Where, the field SUM_ORDER_QUANTITY is the total quantity for each tire's type from all the orders for a planned production.

TotalOrder	
TIRE_ID	SUM_ORDER_QUANTITY

Answer group

- a) `SELECT TIRE_ID, ORDER_QUANTITY into TotalOrder FROM SPC`
- b) `SELECT TIRE.TIRE_ID, ORDER_QUANTITY
into TotalOrder FROM SPC, TIRE`
- c) `SELECT TIRE_ID, SUM(ORDER_QUANTITY) into TotalOrder FROM SPC`
- d) `SELECT SPC.TIRE_ID, SPC.ORDER_QUANTITY
into TotalOrder FROM SPC GROUP BY SPC.TIRE_ID`
- e) `SELECT SPC.TIRE_ID, SUM(ORDER_QUANTITY)
into TotalOrder FROM SPC GROUP BY SPC.TIRE_ID`
- f) `SELECT SPC.TIRE_ID, SUM(ORDER_QUANTITY)
AS SUM_ORDER_QUANTITY
into TotalOrder FROM SPC GROUP BY SPC.TIRE_ID`

Subquestion 2

To obtain the total amount of each material which is needed for producing the tires for the planned production, the following SQL code is executed to get the table TotalMaterial.

```
SELECT TotalOrder.TIRE_ID, SUM_ORDER_QUANTITY,  
       ((SUM_ORDER_QUANTITY/PRO_QUANTITY) * MAT_A) AS SUM_MAT_A,  
       ((SUM_ORDER_QUANTITY/PRO_QUANTITY) * MAT_B) AS SUM_MAT_B,  
       ((SUM_ORDER_QUANTITY/PRO_QUANTITY) * MAT_C) AS SUM_MAT_C,  
       ((SUM_ORDER_QUANTITY/PRO_QUANTITY) * MAT_D) AS SUM_MAT_D,  
       ((SUM_ORDER_QUANTITY/PRO_QUANTITY) * MAT_E) AS SUM_MAT_E,  
       ((SUM_ORDER_QUANTITY/PRO_QUANTITY) * MAT_F) AS SUM_MAT_F,  
       ((SUM_ORDER_QUANTITY/PRO_QUANTITY) * MAT_J) AS SUM_MAT_J  
into TotalMaterial FROM TotalOrder, TIRE  
WHERE TotalOrder.TIRE_ID = TIRE.TIRE_ID
```

From the answer group below, select the correct statement about the number of records of the table TotalMaterial.

Answer group

- a) It is 1.
- b) It is equal to the number of orders of the all customers for the planed production.
- c) It is equal to the number of records of the table TIRE.
- d) It is equal to the number of records of the table TotalOrder.

Subquestion 3

Based on the table TotalMaterial, which is obtained after the execution of the SQL code shown in Subquestion 2, the following SQL code calculates the sum of each material needed for the planed production, and creating the TableA.

From the answer group below, select the correct size of the TableA.

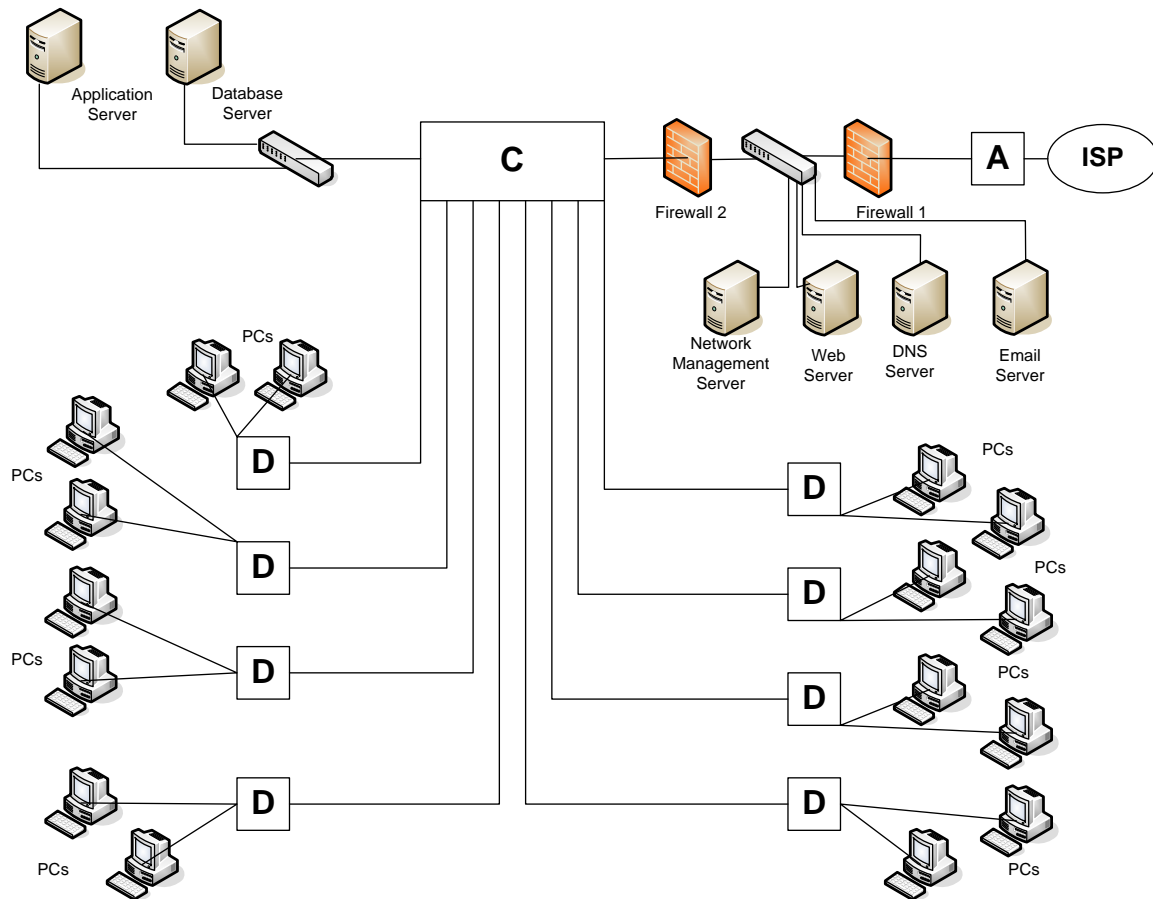
```
SELECT SUM(SUM_MAT_A) AS SUM_MAT_A, SUM(SUM_MAT_B) AS SUM_MAT_B,  
       SUM(SUM_MAT_C) AS SUM_MAT_C, SUM(SUM_MAT_D) AS SUM_MAT_D,  
       SUM(SUM_MAT_E) AS SUM_MAT_E, SUM(SUM_MAT_F) AS SUM_MAT_F,  
       SUM(SUM_MAT_J) AS SUM_MAT_J  
into TableA FROM TotalMaterial
```

Answer group

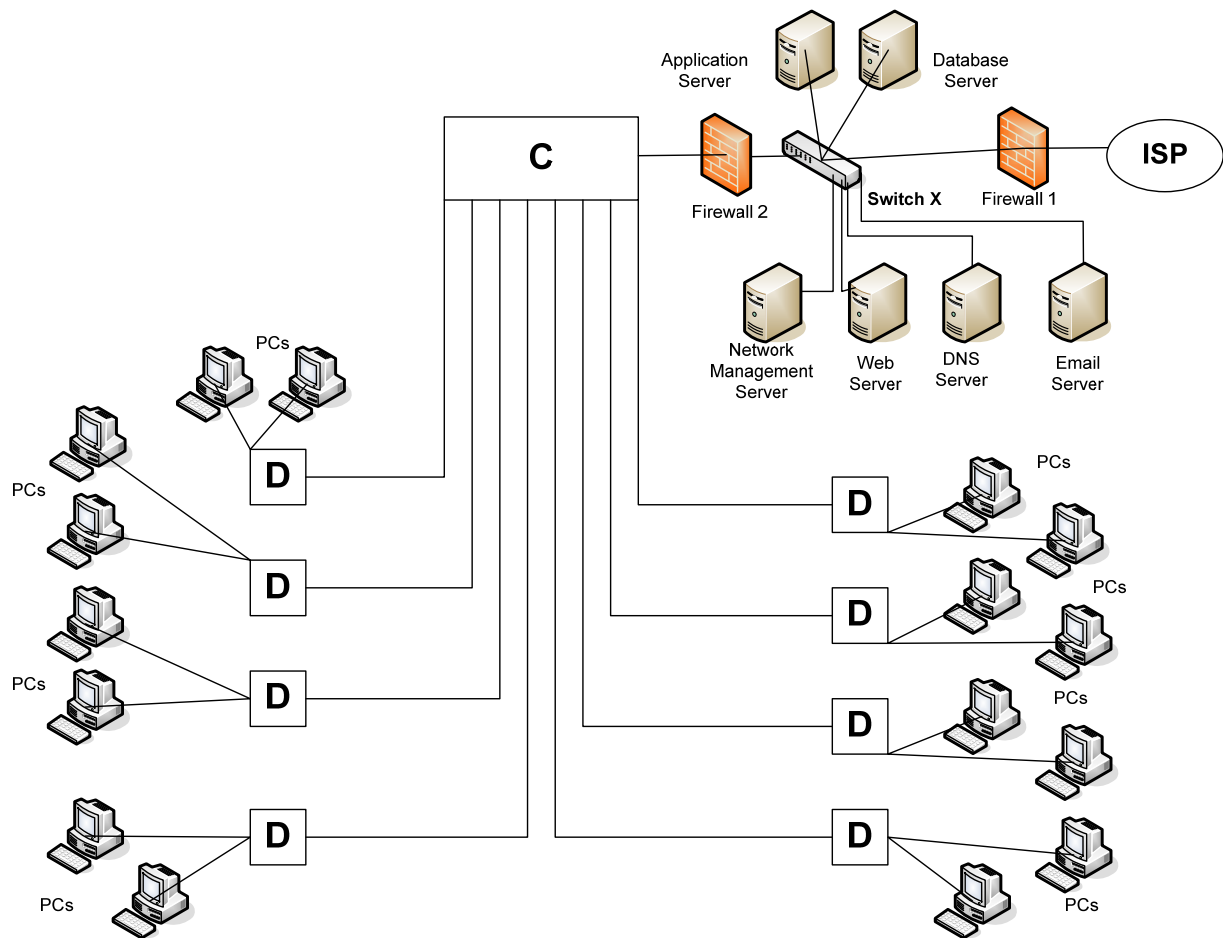
- a) 1 row and 7 columns
- b) 7 rows and 1 column
- c) 7 rows and 7 columns
- d) n rows and 7 columns
- e) The number of rows is depending on the tire's type and 7 columns

Q3 Read the following description about a network configuration, and then answer the Subquestion.

[Network Topology 1]



[Network Topology 2]



[Description]

Network Topology 1 is topology of company X. The gateway of company X's network is the device. There are 4 servers (network management, web, DNS and Email) placed between firewall 1 and firewall 2. The area contains these servers are called . Other servers are application server and database server for internal use of company X. Accessing to the Internet and exchanging/sharing data between the personal computers in company X are distributed by and .

Company X uses a leased-line provided by the Internet Service Provider (ISP). The global IPs that the ISP supplies to company X is 203.88.212.144/29. Therefore, company X has IP addresses.

Now, according to some bussiness demands, all of six servers (application, database, network management, web, DNS and Email) could be connected from the Internet. First, the administrator of company X planned some implementations as in Network Topology 2. But with this new topology, the problem appears with company X is .

Subquestion

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

Answer group for A

- | | |
|------------|-----------|
| a) Adapter | b) Hub |
| c) Modem | d) Router |

Answer group for B

- | | |
|-----------------------------|-----------------------------|
| a) De-Militarized Zone | b) External Accessible Zone |
| c) Internal Accessible Zone | d) Private Zone |

Answer group for C and D

- | | |
|----------------|------------------------|
| a) Core switch | b) Distribution switch |
| c) Hub | d) Modem |
| e) Router | |

Answer group for E

- | | |
|-------|-------|
| a) 4 | b) 8 |
| c) 16 | d) 32 |

Answer group for F

- a) Firewall 1 must be taken away.
- b) Firewall 2 must be taken away.
- c) Not enough IP address for assigning to the servers.
- d) The personal computers in company X become less secure than before.

Q4 Read the following description of a program and the program itself, and then answer the Subquestion.

[Program Description]

Consider the algorithm for a program used to order and output in ascending order the elements in the array *S*. *S* is a structure array, and each element has the following four data items.

character Name
integer Age
float Height
float Weight

Sample data:

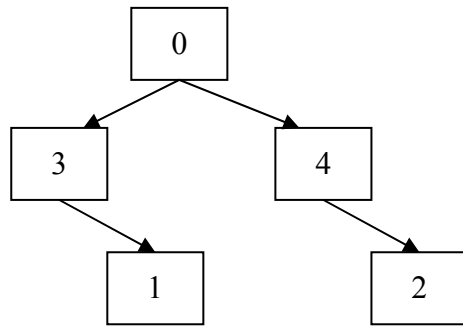
Element no.	Name	Age	Height	Weight
0	David Moore	19	162.5	65.4
1	Lisa Brown	14	158.0	48.4
2	John Abraham	18	182.0	82.5
3	Anne Peterson	12	148.0	46.8
4	Jason Bates	16	178.5	70.0

The program will create a binary tree that indicates the order without actually sorting the array data. Elements will be ordered using *Height* as the key.

Along with the array *S* [MAXNUM], two integer arrays *Lower* [MAXNUM] and *Upper* [MAXNUM] are used in the program to represent the binary tree. Here, MAXNUM is the number of data.

Fig 1. shows the execution result of the program for the sample data, where the elements are arranged in ascending order, with the element numbers in the sequence 3, 1, 0, 4, 2.

Fig 2. shows the contents of the arrays *Lower* and *Upper*. The value -1 indicates that there is no element connected next.



**Fig 1. Ascending Order
Arrangement of Elements**

Element No.	Lower	Upper
0	3	4
1	-1	-1
2	-1	-1
3	-1	1
4	-1	2

**Fig 2. Representation of
Array Content**

Initially, the elements are connected on the “Upper” side in the order in which they appeared in the definition of the structure array *s*, creating the binary tree shown in Fig 3.

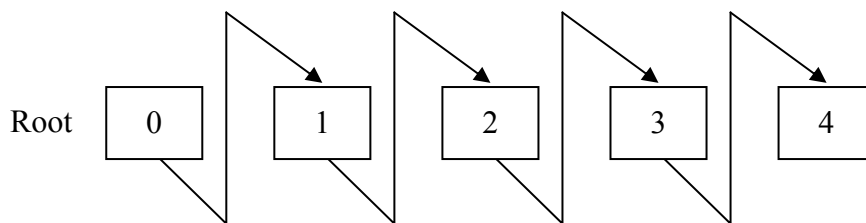


Fig 3. Connection Status Prior to Sorting

Next, the program calls the recursive function `BinTreeSort` to sort the elements in the binary tree according to Height in ascending order. The parameter is the element number of the root. The prototype for the function `BinTreeSort` is as follows:

```
BinTreeSort (integer)
```

Lastly, the program calls the recursive function `DisplayData` to output the contents of the structure array *s* in ascending order according to Height, following the binary tree. The parameter is the element number of the root. The prototype for the function `DisplayData` is as follows:

```
DisplayData (integer)
```

[Program]

Main Algorithm

```
Main Program ( )
    MAXNUM ← 5
    Index ← 0
    While (Index < MAXNUM)
        Upper[Index] ← Index + 1
        Lower[Index] ← -1
        Index ← Index + 1
    Endwhile
    Upper[MAXNUM - 1] ← -1
    BinTreeSort(0)
    DisplayData(0)
```

BinTreeSort Algorithm

```
Function BinTreeSort (Root)
    Data ← Upper[Root]
    If (Data = -1)
        Exit
    Endif
    Upper[Root] ← -1
    While (Data != -1)
        Next ← A
        If (S[Data].Height >= S[Root].Height)
            Upper[Data] ← B
            Upper[Root] ← Data
        Else
            Upper[Data] ← Lower[Root]
            Lower[Root] ← C
        Endif
        Data ← Next
    Endwhile

    Data ← Upper[Root]
    If (Data != -1)
        BinTreeSort(Data)
    Endif
    Data ← Lower[Root]
    If (Data != -1)
        BinTreeSort(Data)
    Endif
```

DisplayData Algorithm

```
Function DisplayData (Root)
    If (Root = -1)
        Exit
    Endif
    DisplayData(  )
    Print S[Root].Name, S[Root].Age, S[Root].Height, S[Root].Weight
    DisplayData(Upper[Root])
```

Subquestion

From the answer group below, select the correct answers to be inserted in the blanks

in the above program.

Answer group

- a) -1
- b) Data
- c) Lower[Data]
- d) Lower[Root]
- e) Next
- f) Root
- g) Upper[Data]
- h) Upper[Root]

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

Company Y implements a customer support system. Part of the system deals with assigning personnel to the different requests. The section chief uses this part of the system, and monitors and assigns the different requests.

[File Description]

- (1) Customer code, password, location, address, customer name and phone no are recorded to a Customer file. The customers log into the system and enter their requests.

The record format of the Customer file:

<u>Customer Code</u>	Password	Location	Address	Customer Name	Phone No
----------------------	----------	----------	---------	---------------	----------

- (2) Personnel file contains personnel code, password, expertise, tel no and personnel name.

The record format of the Personnel file:

<u>Personnel Code</u>	Password	Expertise	Tel No	Personnel Name
-----------------------	----------	-----------	--------	----------------

- (3) Customer requests are entered by the customer indicating the request (or complaint) and due date. A dispatcher further processes it to create the Request file. The dispatcher diagnoses each request and identifies the priority level and the expertise needed. Requests with the earliest due date are scheduled first. Then, for those with the same due dates, tasks with higher priority level are scheduled first. Completed requests are removed from the Request file.

The record format of the Request file:

<u>Request Code</u>	Customer Code	Date of Request	Expertise Needed	Priority Level	Due Date	Assigned
---------------------	---------------	-----------------	------------------	----------------	----------	----------

- (4) The section chief assigns the requests to personnel with matching expertise. This creates an entry in the Assignment file. The section chief looks for the earliest available date that a given personnel, with the matching expertise, can do the task. All requests with due dates within 7 days of the current day are called critical requests. These requests must be assigned. An error will be indicated if this condition is not met. Once a request is completed, the completion date is entered.

The record format of the Assignment file:

<u>Request Code</u>	<u>Personnel Code</u>	Visit Date	Job Status	Completion Date
---------------------	-----------------------	------------	------------	-----------------

[Program Description]

- (1) The section chief logs into the system and presses the Commit function key (F10). A valid login immediately goes to Personnel view. Otherwise, it remains the same.
- (2) The Request file is read and transferred to a temporary matrix called assignment calendar. This is used as the working table for the section chief. Once the assignments are completed, the whole matrix is committed and the Assignment file is updated.
- (3) Personnel view - One view presents a list of personnel in table form with their corresponding assignments. Blank cells mean no assigned tasks.
 - a. The first column contains the personnel code followed by the expertise. The remaining columns will contain the request codes assigned for that day. 01 is for the current day, and 15 is for the 15th day from the current day.

The Assignment Calendar (with sample codes):

Personnel Code	Expertise	01	02	03	04	14	15
PR-01A27	TECH							
PR-01B12	UI							
PR-01E06	TECH							
PR-21C11	ELX							

- b. The section chief encodes the request codes into the empty grids of the matrix corresponding to date assignments. Personnel can get several assignments across different dates.
 - c. When the Check function key (F3) is used, the inputs are checked. The primary display is set as black on white. Request codes turn “blue” if they are scheduled after the due date. Request codes turn “green” if they are assigned to personnel without the matching expertise.
Then, the following message line is displayed.
“ERROR: ## unassigned critical requests. WARNING: ## unmatched expertise, ## late services.” (## represents a 2 digit number)
- (4) Request view - This view-only screen displays the list of all uncompleted requests in the Request file. All requests whose due dates are less than 7 days of the current day are colored “red”. This is used as a quick reference window in making the schedule. The Request function key (F5) opens the window while the Escape key closes it again and brings it back to Personnel view.
- (5) Customer view - This view-only screen displays customer requests and the assigned dates in table form. The first column contains the customer name, the second contains the request code, and the remaining columns represents the dates from the

current day to the 15th day. The personnel code is placed in the corresponding date column of the visit. An * (asterisk) is placed in the column when the request is due. If the due date and the visit date are the same, only the personnel code will be seen. The Customer function key (F6) opens the window, while the Escape key closes it again and brings it back to Personnel view.

- (6) Commit function key (F10) commits the whole session, updates the files, and displays the Exit Screen which displays the number of updates made in the files. If errors are still exist, then no commit will occur, and will remain on the same screen.

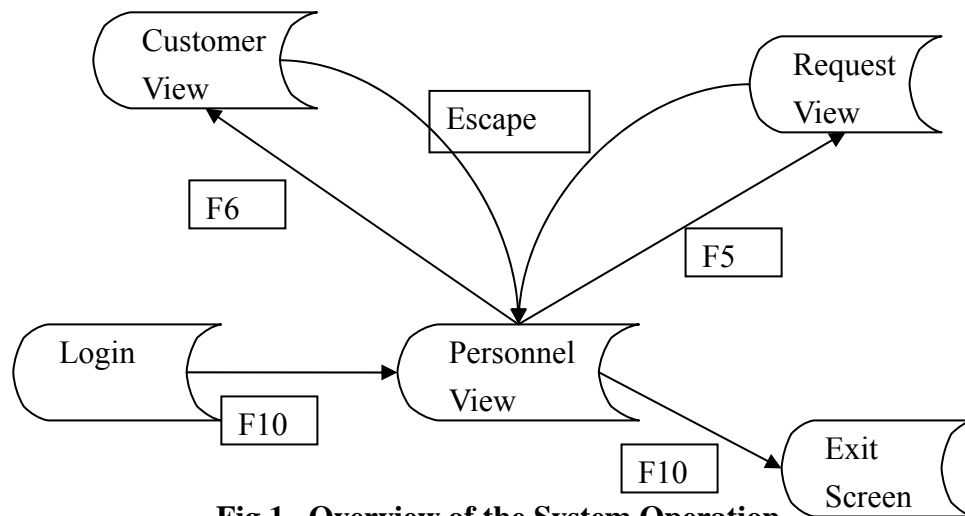


Fig 1. Overview of the System Operation

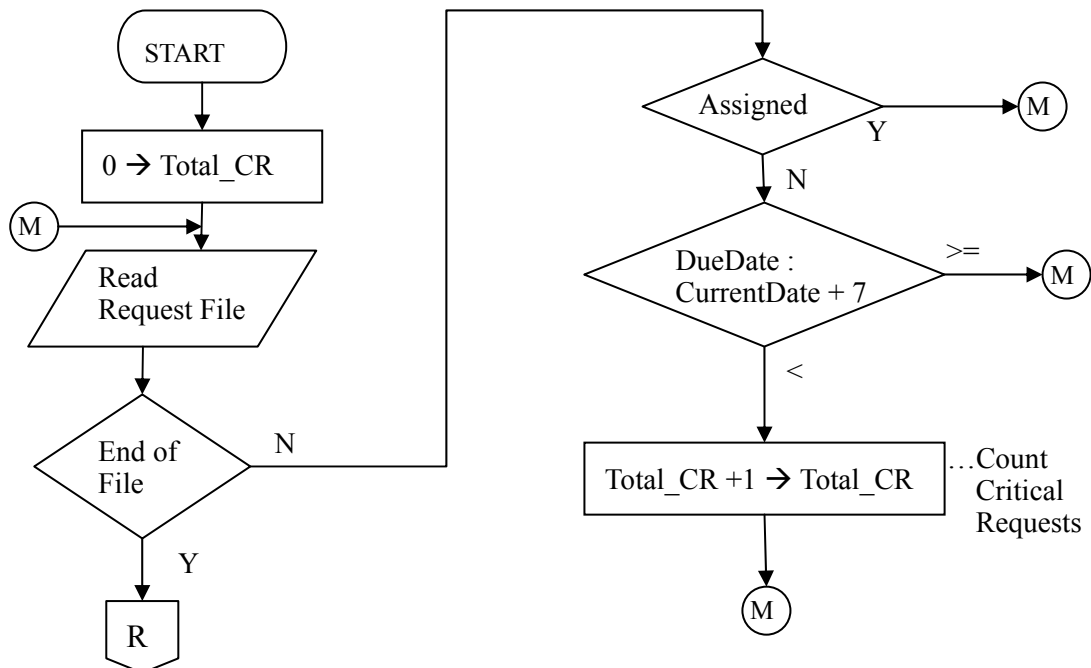


Fig 2. Flowchart for Preparing Personnel View (1 of 2)

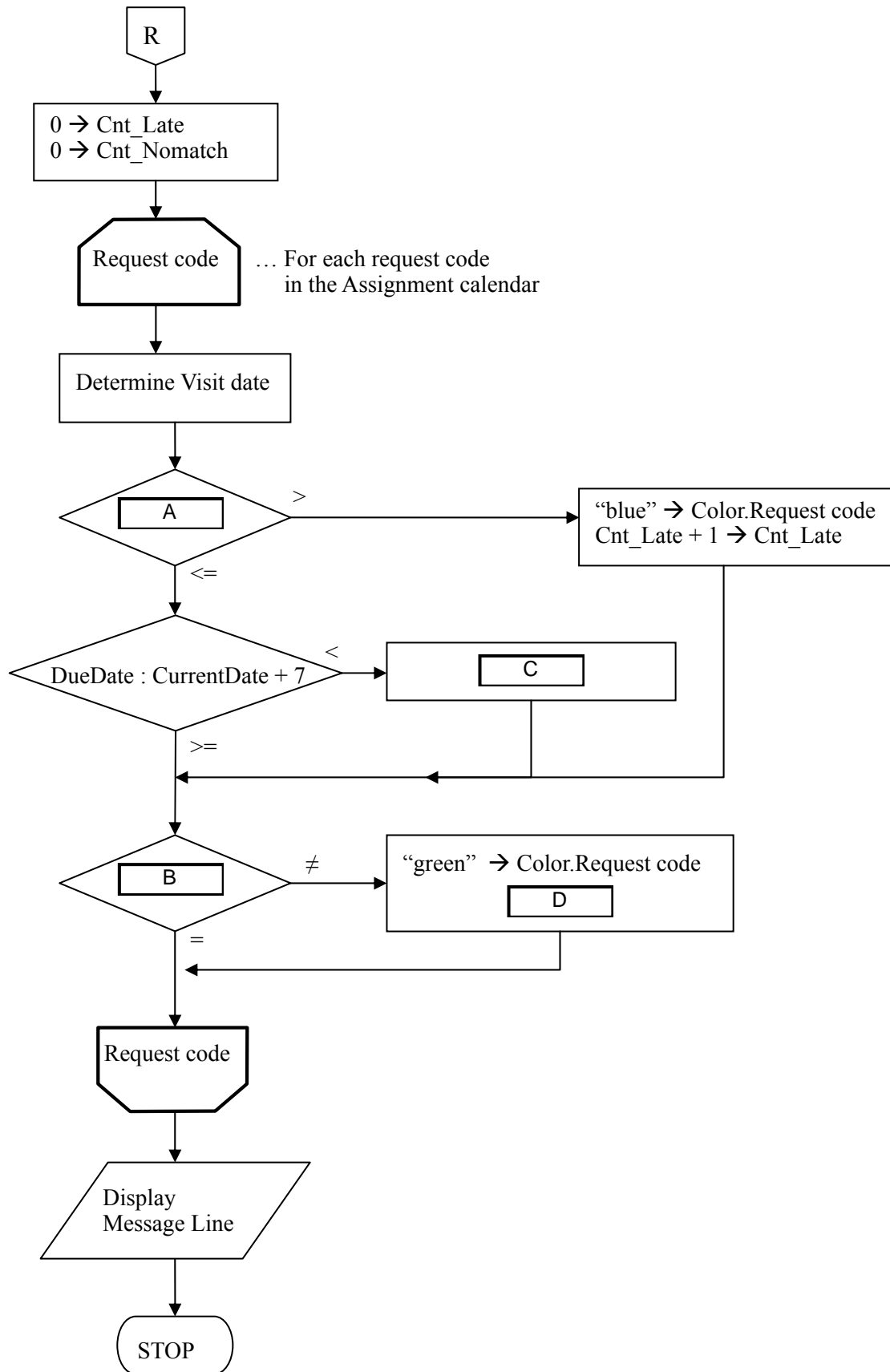


Fig 2. Flowchart for Preparing Personnel View (2 of 2)

Subquestion 1

From the answer groups below, select the correct answers to be inserted in the blanks

in Fig 2.

Answer group for A and B

- a) Completion date : Visit date
- b) Due date : Visit date
- c) Expertise : Expertise needed
- d) Priority level : Expertise needed
- e) Visit date : Due date

Answer group for C and D

- a) $\text{Cnt_Nomatch} + 1 \rightarrow \text{Cnt_Nomatch}$
- b) $\text{Cnt_Nomatch} - 1 \rightarrow \text{Cnt_Nomatch}$
- c) $\text{Total_CR} + 1 \rightarrow \text{Total_CR}$
- d) $\text{Total_CR} - 1 \rightarrow \text{Total_CR}$
- e) "red" \rightarrow Color.Request code
 $\text{Total_CR} + 1 \rightarrow \text{Total_CR}$
- f) "red" \rightarrow Color.Request code
 $\text{Total_CR} - 1 \rightarrow \text{Total_CR}$

Subquestion 2

From the answer groups below, select the correct answers to be inserted in the blanks

in the following description.

In processing and preparing the Assignment calendar, the Assignment file is read. To allow for easy sequential processing of the file, it should be sorted/indexed by E .

Upon commit, the files that will be updated are F .

To display the Customer view, the files that will be used are G .

Answer group for E

- a) Personnel code
- b) Request code
- c) Visit date

Answer group for F and G

- a) Assignment file, Customer file, Personnel file, Request file
- b) Assignment file, Customer file, Request file
- c) Assignment file, Request file
- d) Customer file, Personnel file, Request file

Select one question from **Q6** or **Q7**, 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.

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

[Program description]

Given program compares two files on their identity and prompts the result. Both files are text files, and there is a “New-Line” character at the end of each line.

The program calls a function `CompareFiles()` with two parameters of `FILE*` type.

The program gets two filenames from a keyboard, and reads each line of data from two files to the variables `char line1[MaxLength]` and `char line2[MaxLength]`.

[Program]

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

void CompareFiles(FILE*, FILE*);
const int MaxLength=82;           //up to 80 char / line

void main() {
    char firstfileToOpen[MaxLength];
    char secondfileToOpen[MaxLength];
    FILE *FirstFile;               //file variables
    FILE *SecondFile;

    printf("Enter first file to compare:");
    scanf("%s", firstfileToOpen);  //read name typed by user
    printf("Enter second file to compare:");
    scanf("%s", secondfileToOpen); //read name typed by user
    FirstFile= A ;           // open the file
    if (FirstFile==NULL)           //was the file opened successfully?
    {
        printf("\nCould not open first file\n");
        exit (1);
    }
}
```

```

SecondFile= B ;           //open the file
if (SecondFile==NULL)      //was the file opened successfully?
{
    printf("\nCould not open second file\n");
    exit (1);
}
CompareFiles(FirstFile,SecondFile);
fclose(FirstFile);          //close files
fclose(SecondFile);
}

void CompareFiles(FILE* file1, FILE* file2) {
    char line1 [MaxLength];
    char line2 [MaxLength];
    char* ptrline1;
    char* ptrline2;

    ptrline1=fgets(line1,MaxLength,file1);
    ptrline2=fgets(line2,MaxLength,file2);
    if ( C )
    {
        while ( D )
        {
            if ( E )
            {
                printf("\nFound two different lines\n");
                exit (2);
            }
            if (feof(file1)==0)
                ptrline1=fgets(line1,MaxLength, file1);
            else break;
            if (feof(file2)==0)
                ptrline2=fgets(line2,MaxLength,file2);
            else break;
        }
        printf("\nFiles are identical\n");
        exit (0);
    }
    else
    {
        printf("\nAt least one file is empty \n");
        exit (0);
    }
}
//end of function CompareFiles

```

Subquestion

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

Answer group for A and B

- a) `fopen(FirstFile,"r")`
- b) `fopen(FirstFile,"w")`
- c) `fopen(firstfileToOpen,"r")`
- d) `fopen(firstfileToOpen,"w")`
- e) `fopen(SecondFile,"r")`
- f) `fopen(SecondFile,"w")`
- g) `fopen(secondfileToOpen,"r")`
- h) `fopen(secondfileToOpen,"w")`

Answer group for C

- a) `feof(file1)!=0 && feof(file2)!=0`
- b) `feof(file1)==0 && feof(file2)==0`
- c) `feof(file1)!=0 || feof(file2)!=0`
- d) `feof(file1)==0 || feof(file2)==0`

Answer group for D

- a) `ptrline1!=NULL && ptrline2!=NULL`
- b) `ptrline1==NULL && ptrline2==NULL`
- c) `ptrline1!=NULL || ptrline2!=NULL`
- d) `ptrline1==NULL || ptrline2==NULL`

Answer group for E

- a) `line1 == line2`
- b) `line1 && line2`
- c) `strcmp(line1,line2)!=0`
- d) `strcmp(line1,line2)==0`

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

[Program Description]

We wish to design a collection of cipher classes, including a Caesar cipher and a transposition cipher.

Because the basic operations used in these two forms of encryption are the same, both the Caesar class and the Transpose class will have methods to `encrypt()` and `decrypt()` messages, where each message is assumed to be a string of words separated by spaces. These methods will take a `String` of words and translate each word using the encoding method appropriate for that cipher. Therefore, in addition to `encrypt()` and `decrypt()`, each cipher class will need polymorphic `encode()` and `decode()` methods, which take a single word and encode or decode it according to the rules of the particular cipher.

Caesar cipher: The letters of the alphabet are shifted by three letters. The program handles lower-case letters only. For example:

PlainText:	abcdefghijklmnopqrstuvwxyz
CaesarShifted:	defghijklmnopqrstuvwxyzabc

Transposition cipher: The letters in the original message are rearranged in some methodical way. A simple rule used in the program is to reverse the order of the letters in each word. For example, "hello" becomes "olleh".

```
***** Caesar Cipher Encryption *****
PlainText: animals are in the zoo
Encrypted: dqlpdov duh lq wkh crr
Decrypted: animals are in the zoo

*** Transpose Cipher Encryption ***
PlainText: animals are in the zoo
Encrypted: slamina era ni eht ooz
Decrypted: animals are in the zoo
```

Figure: Execution Results

[Program]

```
public class TestEncrypt {

    public static void main(String argv[]) {
        Caesar caesar = new Caesar();
        String plain = "animals are in the zoo";

        String secret = caesar.encrypt(plain);
        System.out.println
            (" ***** Caesar Cipher Encryption *****");
        System.out.println("PlainText: " + plain);
        System.out.println("Encrypted: " + secret);
        System.out.println
            ("Decrypted: " + caesar.decrypt(secret));

        Transpose transpose = new Transpose();
        secret = transpose.encrypt(plain);
        System.out.println
            ("\n *** Transpose Cipher Encryption ***");

        System.out.println("PlainText: " + plain);
        System.out.println("Encrypted: " + secret);
        System.out.println
            ("Decrypted: " + transpose.decrypt(secret));
    }
}

import java.util.StringTokenizer;
A {
    public String encrypt(String s) {
        if (s == null || s.equals("")) return s;
        StringBuffer result = new StringBuffer();
        StringTokenizer words = new StringTokenizer(s);
        while (words.hasMoreTokens()) {
            result.append(encode(words.nextToken()) + " ");
        }
        return result.substring(0, result.length()-1);
    }

    public String decrypt(String s) {
        if (s == null || s.equals("")) return s;
        StringBuffer result = new StringBuffer();
        StringTokenizer words = new StringTokenizer(s);
        while (words.hasMoreTokens()) {
            result.append(decode(words.nextToken()) + " ");
        }
        return result.substring(0, result.length()-1);
    }

    public abstract String encode(String word);

    public abstract String decode(String word);
}
```

```

class Caesar extends Cipher {
    public String encode(String word) {
        StringBuffer result = new StringBuffer();
        B {
            char ch = word.charAt(k);
            ch = (char) ('a' + (ch - 'a' + 3) % 26);
            result.append(ch);
        }
        return result.toString();
    }

    public String decode(String word) {
        StringBuffer result = new StringBuffer();
        for (int k = 0; k < word.length(); k++) {
            char ch = word.charAt(k);
            C ;
            result.append(ch);
        }
        return result.toString();
    }
}

D {
    public String encode(String word) {
        StringBuffer result = new StringBuffer(word);
        return result.reverse().toString();
    }

    public String decode(String word) {
        return encode(word);
    }
}

```

Subquestion

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

Answer group for A

- a) `abstract class Cipher`
- b) `class Cipher`
- c) `final class Cipher`
- d) `interface Cipher`

Answer group for B

- a) `for (int k=0; k < word.length; k++)`
- b) `for (int k=0; k<word.length(); k++)`
- c) `for (int k=word.length; k>0; k--)`
- d) `for (int k=word.length(); k>0; k--)`

Answer group for C

- a) `ch = (char) ('a' + (ch - 'a' - 3) % 26)`
- b) `ch = (char) ('a' + (ch - 'a' + 3) % 26)`
- c) `ch = (char) ('a' + (ch - 'a' - 23) % 26)`
- d) `ch = (char) ('a' + (ch - 'a' + 23) % 26)`

Answer group for D

- a) `class Transpose`
- b) `class Transpose extends Cipher`
- c) `class Transpose implements Cipher`
- d) `interface Transpose`

Select one question from **Q8** or **Q9**, 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.

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

[Program Description]

Piglatin is an encoded form of English that is often used by children as a game. It transforms an English text into a “strange and foreign-sounding language”.

A piglatin word is formed from an English word by transposing the first sound (in this program, the first letter) to the end of the word, and then adding the letter “a”. For example, the word “dog” becomes “ogda,” “computer” becomes “omputerca”, “piglatin” becomes “iglatinpa”, and so on.

Given C program accepts a line of English text up to 80 characters, and then print out the corresponding text in piglatin. A text must be started at column 1 (no leading space(s)), and words must be separated by only one space character.

An example below shows a sample input and it’s output:

Input: have a nice day

Output: aveha aa icena ayda

Briefly, the program consists of the following major steps.

- (1) Initialize I/O arrays.
- (2) Read in an entire line of text.
- (3) Determine the number of words in the line (words are separated by space(s)).
- (4) Rearrange the words into piglatin form, on a word-by-word basis, as follows:
 - (a) Locate the end of the word.
 - (b) Transpose the first letter to the end of the word and then add an “a.”
 - (c) Locate the beginning of the next word.
- (5) Display the entire line of text in piglatin form.

The program continues this procedure repetitively until the program reads a line of text whose first three letters are “end” (or “END”).

[Program]

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

void initialize(char english[], char piglatin[]);
void readinput(char english[]);
int countwords(char english[]);
void convert(int words, char english[], char piglatin[]);
void writeoutput(char piglatin[]);

void main()
{
    char english[81], piglatin[121];
    int words;
    printf("Welcome to the Piglatin Generator\n\n");
    printf("Type \'END\' to finish\n\n");
    do
    {
        /* process a new line of text */
        initialize(english, piglatin);
        readinput(english);
        if (toupper(english[0])=='E' && toupper(english[1])=='N'
            && toupper(english[2])=='D') break;
        A ;
        /* convert english into piglatin */
        if (english[0] != ' ')
        {
            convert(words, english, piglatin);
            writeoutput(piglatin);
        }
        else printf("Column 1 must not be blank, re-enter.\n");
    }
    while(words >= 0);
    printf("\naveHa aa icena ayda (Have a nice day)\n");
}
```

```

/* initialize the character arrays with blank spaces */
void initialize(char english[], char piglatin[])
{
    int count;
    for(count=0; count<81; count++)
        english[count]=' ';
    for(count=0; count<121; count++)
        piglatin[count]=' ';
    return;
}

/* read one line of English text */
void readinput(char english[])
{
    int count = 0;
    char c;
    while (  )
    {
        if (count < 80)
            english[count]=c;
        count++;
    }
    return;
}

/* scan the English text and determine the number of words */
int countwords(char english[])
{
    int words=1;
    int count;
    for(count=0; count<79; count++)
        if(  )
            words++;
    return(words);
}

```

```

/* convert each word into piglatin */
void convert(int words, char english[], char piglatin[])
{
    int n, count;
    int m1=0;
    int m2;
    for(n=1;  ; n++)
    {
        count=m1;
        while(english[count]!=' ')
            m2=count++;
        for(count=m1; count<m2; count++)
             ;
        piglatin[m2+(n-1)]=english[m1];
        piglatin[m2+n]='a';
         ;
    }
    piglatin[m2+n+1]='\0';
    return;
}

/* display the line of text in piglatin */
void writeoutput(char piglatin[])
{
    int lp=strlen(piglatin);
    int count;
    for(count=0; count<lp; count++)
        putchar(piglatin[count]);
    printf ("\n");
    return;
}

```

Subquestion

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

Answer group for A

- a) english = piglatin
- b) piglatin = english
- c) words = countwords(english)
- d) words = countwords(piglatin)

Answer group for B

- a) `(c=getchar())!='\n'`
- b) `(c=getchar())=='\n'`
- c) `c=='\n'`
- d) `c=getchar()`

Answer group for C

- a) `english[count] != ' ' && english[count+1] == ' '`
- b) `english[count] == ' ' && english[count+1] != ' '`
- c) `english[count-1] != ' ' && english[count] == ' '`
- d) `english[count-1] == ' ' && english[count] != ' '`

Answer group for D

- a) `n<=80`
- b) `n<=strlen(english)`
- c) `n<=strlen(piglatin)`
- d) `n<=words`

Answer group for E

- a) `piglatin[count + (n+1)] = english[count+1]`
- b) `piglatin[count + (n+1)] = english[count-1]`
- c) `piglatin[count + (n-1)] = english[count+1]`
- d) `piglatin[count + (n-1)] = english[count-1]`

Answer group for F

- a) `m1 = m2 + 1`
- b) `m1 = m2 + 2`
- c) `m2 = m1 + 1`
- d) `m2 = m1 + 2`

Q9 Read the following description of a Java program and the program itself, and then answer the Subquestion.

A Java program consists of the following classes that represent various types of employees that are employed at a company.

Classes	Purpose
Firm	This is the executable class which has the <code>main()</code> method. It creates a <code>Staff</code> of employees and invokes the <code>payday</code> method to pay them all. The program output includes information about each employee and how much each is paid.
Staff	The <code>Staff</code> class maintains an array of objects that represent individual employees of various kinds (<code>staffList</code>). The array is declared to hold <code>StaffMember</code> references and filled with objects created from several other classes. These classes are all inherit from <code>StaffMember</code> class. The <code>staffList</code> array is filled with polymorphic references.
StaffMember	This abstract class represents a generic staff member. It does not represent a particular type of employee and is not intended to be instantiated. It serves as the ancestor of all employee classes and contains information that applies to all employees. Each employee has a name, address and phone number, so variables to store these values are declared in the <code>StaffMember</code> class and are inherited by all subclasses.
Volunteer	This class represents a staff member that works as a volunteer. A volunteer is not compensated monetarily for his or her work.
Employee	This class represents an employee that gets paid at a particular rate each period.
Executive	This class represents an executive staff member, who can earn a bonus in addition to his or her own normal pay rate.
Hourly	This class represents an employee that gets paid by the hour.

The `payday` method of the `Staff` class scans through the list of employees, printing their information and invoking their `pay` methods to determine how much each employee should be paid. The invocation of the `pay` method in some of the classes described above is polymorphic because each class has its own version of the `pay` method.

The following output shows the execution results of the program.

```
Name: Sam
Address: 123 Main Line
Phone: 555-0469
Social Security Number: 123-45-6789
Paid: 2923.07
-----
Name: Carla
Address: 456 Off Line
Phone: 555-0101
Social Security Number: 987-65-4321
Paid: 1246.15
-----
Name: Woody
Address: 789 Off Rocker
Phone: 555-0000
Social Security Number: 010-20-3040
Current hours: 40
Paid: 422.0
-----
```

[Program]

```
public class Firm {
    public static void main(String[] args) {
        Staff personnel = new Staff();
        A ;
    }
}

public class Staff {
    private StaffMember[] staffList;

    public Staff() {
        staffList = new StaffMember[3];
        staffList[0] = new Executive("Sam", "123 Main Line",
            "555-0469", "123-45-6789", 2423.07);
        staffList[1] = new Employee("Carla", "456 Off Line",
            "555-0101", "987-65-4321", 1246.15);
        staffList[2] = new Hourly("Woody", "789 Off Rocker",
            "555-0000", "010-20-3040", 10.55);
        (( B )staffList[0]).awardBonus(500.00);
        (( C )staffList[2]).addHours(40);
    }
}
```

```

public void payday() {
    double amount;
    for (int count=0; count < staffList.length; count++)
    {
        System.out.println(staffList[count]);
        amount = D ; // polymorphic
        if (amount == 0.0)
            System.out.println("Thanks!");
        else
            System.out.println("Paid: " + amount);
        System.out.println("-----");
    }
}

}

abstract public class StaffMember {
    protected String name;
    protected String address;
    protected String phone;

    public StaffMember(String eName,
                        String eAddress, String ePhone) {
        name = eName;
        address = eAddress;
        phone = ePhone;
    }

    public String toString() {
        String result = "Name: " + name + "\n";
        result += "Address: " + address + "\n";
        result += "Phone: " + phone;
        return result;
    }

    public abstract double pay();
}

public class Volunteer extends StaffMember {
    public Volunteer(String eName, String eAddress, String ePhone) {
        super(eName, eAddress, ePhone);
    }

    public double pay() {
        return 0.0;
    }
}

```

```

public class Employee extends StaffMember {
    protected String socialSecurityNumber;
    protected double payRate;

    public Employee(String eName, String eAddress, String ePhone,
                    String socSecNumber, double rate) {
        super(eName, eAddress, ePhone);
        socialSecurityNumber = socSecNumber;
        payRate = rate;
    }

    public String toString() {
        String result = E + "\nSocial Security Number: " +
                        socialSecurityNumber;
        return result;
    }

    public double pay() {
        return payRate;
    }
}

public class Executive extends F {
    private double bonus;

    public Executive(String eName, String eAddress, String ePhone,
                    String socSecNumber, double rate) {
        super(eName, eAddress, ePhone, socSecNumber, rate);
        bonus = 0;
    }

    public void awardBonus(double execBonus) {
        bonus = execBonus;
    }

    public double pay() {
        double payment = super.pay() + bonus;
        bonus = 0;
        return payment;
    }
}

```

```

public class Hourly extends Employee {
    private int hoursWorked;

    public Hourly(String eName, String eAddress, String ePhone,
                  String socSecNumber, double rate) {
        G ;
        hoursWorked = 0;
    }

    public void addHours(int moreHours) {
        hoursWorked += moreHours;
    }

    public double pay() {
        double payment = payRate * hoursWorked;
        hoursWorked = 0;
        return payment;
    }

    public String toString() {
        String result = super.toString();
        result += "\nCurrent hours: " + hoursWorked;
        return result;
    }
}

```

Subquestion

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

Answer group for A

- a) `payday()`
- b) `personnel`
- c) `personnel.payday()`
- d) `personnel.Staff()`
- e) `personnel=null`
- f) `Staff()`

Answer group for B, C and F

- a) `abstract`
- b) `Employee`
- c) `Executive`
- d) `Hourly`
- e) `Object`
- f) `Staff`
- g) `StaffMember`
- h) `super`
- i) `Volunteer`

Answer group for D and E

- a) `payrate+bonus`
- b) `payrate+super.bonus`
- c) `staffList[count].name + staffList[count].bonus`
- d) `staffList[count].pay()`
- e) `staffList[count].pay(null)`
- f) `staffList[count].toString()`
- g) `super.toString()`

Answer group for G

- a) `Employee (eName, eAddress, ePhone, socSecNumber, rate)`
- b) `Executive (eName, eAddress, ePhone, socSecNumber, rate)`
- c) `super()`
- d) `super(eName, eAddress, ePhone, socSecNumber, rate)`
- e) `super(null)`
- f) `Volunteer(eName, eAddress, ePhone) + super(socSecNumber, rate)`