E-GENTING
PROGRAMMING COMPETITION 2014

General instructions:

1. Answer one or more of the questions.
2. The competition is an open book test.
3. The duration of the competition is 8 hours.
4. Do not discuss matters related to the questions with other contestants.
5. To receive credit for answering a question, your answer must be a credible response to the question. A credible response is an answer that solves the problem or would be likely to solve the problem with a little additional effort.
6. Provided your answer is a credible response, you will receive credit for the products of a methodical approach. For example, data flow diagrams and state transition diagrams and tables.
7. Your total score is the sum of the credit you receive from each credible response.
8. Your programs will be assessed on the ease with which they can be read and understood.
    - Indenting must be clean and consistent.
    - Variable names should describe the contents of the variables.
    - Coupling between modules should be visible.
    - Each module should do one thing well.
9. The questions are worth the following marks:

No	Name	Marks
1.	Car Customisation	450
2.	Hide and Seek	350
3.	Order Status Report	300
4.	Greetings by Name	200

10. Unless otherwise stated, your programs may be written in any mainstream programming language under any mainstream operating system.
11. Unless otherwise stated, you may make use of all the standard library functions of your chosen language and operating system.
12. The words ‘must’, ‘must not’, ‘required’, ‘should’, ‘should not’, and ‘may’ are to be interpreted as described in RFC 2119[1].
13. You are NOT expected to answer all questions.

---

[1] *Key words for use in RFCs to Indicate Requirement Levels*, RFC 2119, S. Bradner, March 1997.

1 CAR CUSTOMISATION

The Car Customisation Corporation (CCC) operates a workshop that installs custom accessories in cars. At the beginning of each day, the sales team lists all the customisation requests from customers in a single text file called the customisation file. Each line in the customisation file represents a car and the customisations to the car ordered by the customer. Each line begins with a car serial number, followed by the Stock Keeping Unit (SKU) identifiers for the rim, tyre, audio system and GPS system. Each field is separated by a comma and has a maximum length of 15 characters. The serial numbers and SKU identifiers are alphanumeric values. An SKU identifier field may be blank to indicate that the accessory does not need to be changed. A sample of the customisation file is shown in Figure 1.

```
T02BZOI826,AINSA222,INSW2978,297DIUW,8HD972
T02BZOI827,AINSA222,INSW2978, ,8HD972
T02BZOI828,A82IDU22,INSW2978,229FIF8676,D28HHDY
```

Figure 1 – Sample Customisation File

In the workshop, a human operator reads the customisation file and manually configures a robotic system that physically removes and installs the accessories.

Due to human errors on the part of the operator, the cars are sometimes fitted with the wrong accessories, leading to slower delivery times and lower customer satisfaction. To overcome this problem, CCC would like you to write a new module to read the customisation file and automatically configure the robotic system. The module should have the declaration shown in Figure 2. The module should use the Application Programming Interface (API) shown in Figure 3 to control the robotic assembly machines. The methods in the API are thread-safe.

The workshop has 5 assembly stations, but some of the stations may not be serviceable when the module starts. The module may assume that no assembly station will become unserviceable and no new assembly station will become serviceable while the module is running.

A batch of cars is queued to each assembly station. The module must process the cars in the order they arrive at the assembly station. The cars may arrive at the assembly station in a different order to the order of the lines in the customisation file.

The module must verify that all the fields in the customisation file are valid. Car serial numbers that are not found in any assembly station queue are invalid. If an invalid value is found, the module should append an error message to an error file and then ignore the value. The error message should contain the car serial number and the invalid value. Each error message should occupy a single line in the error file.

The module must direct the robotic system to remove existing accessories that are not required or are to be replaced before it installs new accessories. In addition, rims must be

removed before tyres can be installed or removed. Rims must not be installed before tyres are put on the rims.

The module should call the `InstallAccessory` function with an accessory type and a valid SKU identifier to install an accessory. The module should call the `RemoveAccessory` function with an accessory type to remove an accessory.

If the required accessory is already installed, the module should not remove or reinstall the accessory. If a car does not require any customisation, the module must eject the car from the assembly station without performing any installations or removals. Similarly, the module must eject the car from the assembly station when all the required customisations are completed.

Before actually installing or removing an accessory or ejecting a car, the module must verify that the assembly station is ready for the request. If an assembly station is not ready, the module should wait for a second and then test the status of the assembly station again. It should repeat this process until the assembly station is ready.

The module should make full use of all the available assembly stations concurrently.

C++

```
void PerformCustomisation (const char *file, const char *errorFile, IMachines *machines)
{
}
```

Java

```
public class Workshop {
        public static void performCustomisation (String file, String errorFile,
                Machines machines)
        {
        }
}
```

C#

```
public class Workshop {
        public static void PerformCustomisation (string file, string errorFile,
                Machines machines)
        {
        }
}
```

Descriptions

`file`
: is the file name of the customisation file

`errorFile`
: is the file name of the file to which error messages are to be written

`machines`
: is an object instance that implements the API

Figure 2 – Module Declaration

C++

```
enum AccessoryType {
        RIM     = 1,
        TYRE    = 2,
        AUDIO   = 3,
        GPS     = 4,
};
class Accessory {
public:
        AccessoryType  accessoryType;         // Accessory type
        const char     skuId[16];             // SKU identifier
};
class IMachines {
public:
        virtual int GetServiceableStations(int *stations) = 0;
                                        // Get list of serviceable assembly stations
        virtual bool IsStationReady (int stationId) = 0;
                                        // Test whether the assembly station is ready
                                        // for the next request
        virtual bool GetCarNumber (int stationId, const char *carSerNo) = 0;
                                        // Get the serial number of the car currently
                                        // at the assembly station
        virtual int GetAccessories (int stationId,
                Accessory *accessoryArray) = 0;
                                        // Get a list of the accessories installed in
                                        // the car currently at the assembly station
        virtual bool IsValidAccessory (AccessoryType accessoryType,
                const char *skuId) = 0;
                                        // Test whether a SKU identifier is valid
        virtual void InstallAccessory (int stationId,
                AccessoryType accessoryType, const char *skuId) = 0;
                                        // Install an accessory in the car currently at
                                        // the assembly station
        virtual void RemoveAccessory (int stationId,
                AccessoryType accessoryType) = 0;
                                        // Remove an accessory from the car currently at
                                        // the assembly station
        virtual void Eject (int stationId) = 0;
                                        // Eject the car from the assembly station and move
                                        // the next car in the queue to the assembly station
};
```

Java

```
public enum AccessoryType {
        RIM,
        TYRE,
        AUDIO,
        GPS,
}
public class Accessory {
        public AccessoryType   accessoryType;         // Accessory type
        public String          skuId;                 // SKU identifier
}
public interface Machines {
        int[] getServiceableStations();
                                        // Get list of serviceable assembly stations
        boolean isStationReady (int stationId);
                                        // Test whether the assembly station is ready
                                        // for the next request
        String getCarNumber (int stationId);
                                        // Get the serial number of the car currently
                                        // at the assembly station
        Accessory [] getAccessories (int stationId);
                                        // Get a list of the accessories installed in
```

```
                                    // the car currently at the assembly station
        boolean isValidAccessory (AccessoryType accessoryType, String skuId);
                                    // Test whether a SKU identifier is valid
        void installAccessory (int stationId, AccessoryType accessoryType,
                String skuId);
                                    // Install an accessory in the car currently at
                                    // the assembly station
        void removeAccessory (int stationId, AccessoryType accessoryType);
                                    // Remove an accessory from the car currently at
                                    // the assembly station
        void eject (int stationId);
                                    // Eject the car from the assembly station and move
                                    // the next car in the queue to the assembly station
}
```

C#

```
public enum AccessoryType {
        RIM     = 1,
        TYRE    = 2,
        AUDIO   = 3,
        GPS     = 4,
}
public class Accessory {
        public AccessoryType   AccessoryType;         // Accessory type
        public string          SkuId;                 // SKU identifier
}
public interface Machines {
        int[] GetServiceableStations();
                                    // Get list of serviceable assembly stations
        bool IsStationReady (int stationId);
                                    // Test whether the assembly station is ready
                                    // for the next request
        string GetCarNumber (int stationId);
                                    // Get the serial number of the car currently
                                    // at the assembly station
        Accessory [] GetAccessories (int stationId);
                                    // Get a list of the accessories installed in
                                    // the car currently at the assembly station
        bool IsValidAccessory (AccessoryType accessoryType, string skuId);
                                    // Test whether a SKU identifier is valid
        void InstallAccessory (int stationId, AccessoryType accessoryType,
                string skuId);
                                    // Install an accessory in the car currently at
                                    // the assembly station
        void RemoveAccessory (int stationId, AccessoryType accessoryType);
                                    // Remove an accessory from the car currently at
                                    // the assembly station
        void Eject (int stationId);
                                    // Eject the car from the assembly station and move
                                    // the next car in the queue to the assembly station
}
```

Descriptions

`GetServiceableStations`
: returns an array of assembly station identifiers that identify the serviceable assembly stations. The C++ version writes the assembly station identifiers as an array at the location referenced by the `stations` pointer and returns the number of elements written.

`IsStationReady`
: returns true if the assembly station identified by the assembly station identifier is ready to process the next request.

`GetCarNumber`
: returns the car serial number of the car at the assembly station identified by the assembly station identifier. If all the cars queued to the station have been processed, it returns null. In the case of the C++ function, if a car is at the assembly station, the function loads the car serial number into the string that `carSerNo` points to and returns true. If all the cars queued to the station have been processed, it returns false.

`GetAccessories`
: returns an array of accessories that are already installed in the car currently at the assembly station identified by the assembly station identifier. The C++ version writes the accessories into the array that `accessoryArray` points to and returns the number of accessories written.

`IsValidAccessory`
: returns true if the SKU identifier is a valid SKU identifier for the specified accessory type.

`InstallAccessory`
: installs the accessory in the car at the assembly station identified by the assembly station identifier.

`RemoveAccessory`
: removes the accessory from the car at the assembly station identified by the assembly station identifier.

`Eject`
: ejects the car from the assembly station identified by the assembly station identifier and moves the next car in the queue, if any, to the assembly station.

`stationId`
: is a 32-bit integer that uniquely identifies an assembly station.

`accessoryType`
: is a enumerated/integer value that identifies the accessory type.

`skuId`
: is a string that identifies the SKU.

Figure 3 - Machines Application Programming Interface

2 HIDE AND SEEK

An information agency has operatives who need to carry sensitive information. The agency would like a product that would allow their operatives to carry information without there being so much as a clue that information is being carried at all. Encryption is all very well, but a file full of encrypted data on a USB memory stick can be hard to explain away.

Their idea is that it might be possible to hide information in encrypted form in the unused space on a file system and then recover the information at a later date by scanning the media from beginning to end. In this way, if an operative were ever intercepted, the operative would be able to deny everything, asserting that the apparently random data on the unused space on the file system was nothing more than a consequence of a file system scrubbing process used to ensure that no data was being carried.

The agency would like you to create two programs, one called ‘hide’ and the other called ‘seek’ to accomplish this objective.

The hide program must accept a stream of plaintext from its standard input stream. Before encrypting the plaintext and writing the ciphertext to the file system, the hide program must create a temporary file and fill the file with random data until the file system is full. It must then encrypt 8 copies of each plaintext page and write each of the encrypted pages at a random location in the file that are not occupied by other encrypted pages. Each plaintext page must have the structure presented in Figure 5. Each encrypted page must have the structure presented in Figure 4. The 8 copies mitigate against the risk of casual use of the file system overwriting all the copies of a particular page and the distinct copy number prevents the multiple copies of a page encrypting to the same ciphertext, which might give a clue to a cryptanalyst. When the hide program has finished writing to the temporary file, it must delete the temporary file from the file system directory.

The seek program must open the file system media as if the media were a file, read each page and endeavour to validate the digital signature of the encrypted pages. If the digital signature is successfully authenticated, the seek program should assume that the page belongs to the document being recovered. It should then decrypt the ciphertext into the structure presented in Figure 5 and put the plaintext in the correct position in the recovered document. When the document is completely assembled, the seek program should emit the document to its standard output stream.

The seek program should identify when a page is missing and display a warning showing the page number.

Byte positions	Contents
0 to 991	Encrypted value of the structure presented in Figure 5
992 to 1003	Random data
1004 to 1023	Digital signature

Figure 4 - Encrypted Page Structure

Byte positions	Contents
0 to 3	Page number
4 to 5	Copy number
6 to 7	Plaintext size in bytes
8 to 975	Plaintext, padded if necessary with random data

Figure 5 - Plaintext Page Structure

The page numbers in the plaintext page must start at zero and increment sequentially for each page of plaintext.

Note that the unencrypted header is 976 bytes long and its encrypted value is 992 bytes long. The encrypted value is 16 bytes longer to accommodate a random initialisation vector for the cipher block chaining (CBC) mode of encryption.

As mentioned earlier, the copy number must have a distinct value for each encrypted copy of a plaintext page. To frustrate known plaintext attacks, the hide program should select random, rather than sequential, plaintext copy numbers.

The plaintext size in bytes is the number of bytes in the plaintext field that are actually used. For all but the last page in the plaintext document, the plaintext size will be 968. For the last page, the plaintext size will be the number of bytes in the last page that are actually used. The hide program must pad the remaining storage with random data.

The hide and seek programs must use the functions listed in Figure 6 to perform the encryption and digital signature functions.

The programs must accept an encryption key that is used by the encryption and digital signature functions. The encryption key is a text string. The hide and seek programs should receive only a single encryption key from the user and pass that key to both the encryption and digital signature functions.

For the purpose of this project, the seek program can access the file system media by opening the pseudo-file ‘wholedisk’.

C++

```
void Encrypt(const char *key, const void *plaintext, void *ciphertext);
                                        // Encrypt the plaintext into ciphertext
void Decrypt(const char *key, const void *ciphertext, void *plaintext);
                                        // Decrypt the ciphertext into plaintext
void GetSignature(const char *key, const void *data, void *signature);
                                        // Derive digital signature
unsigned char GetRandomByte ();
                                        // Get a cryptographically secure random byte
```

Java

```
public class Util {
        static public void Encrypt(String key, byte[] plaintext, byte[] ciphertext);
                                        // Encrypt the plaintext into ciphertext
        static public void Decrypt(String key, byte[] ciphertext, byte[] plaintext);
                                        // Decrypt the ciphertext into plaintext
        static public void GetSignature(String key, byte[] data, byte[] signature);
                                        // Derive digital signature
        static public byte GetRandomByte ();
                                        // Get a cryptographically secure random byte
}
```

C#

```
public class Util {
        static public void Encrypt(string key, byte[] plaintext, byte[] ciphertext);
                                        // Encrypt the plaintext into ciphertext
        static public void Decrypt(string key, byte[] ciphertext, byte[] plaintext);
                                        // Decrypt the ciphertext into plaintext
        static public void GetSignature(string key, byte[] data, byte[] signature);
                                        // Derive digital signature
        static public byte GetRandomByte ();
                                        // Get a cryptographically secure random byte
}
```

Descriptions

`Encrypt`
: encrypts `plaintext` to `ciphertext`.

`Decrypt`
: decrypts `ciphertext` to `plaintext`.

`GetChecksum`
: calculates checksum of `data` and store in `checksum`.

`GetRandomByte`
: generates and returns a cryptographically secure random byte.

Figure 6 – Cryptographic Functions

3 ORDER STATUS REPORT

A soft drink factory uses one system to track customer orders and another to track stock. Your job is to write a reporting program that consolidates data from the two systems into a single Order Status Report.

The Order Status Report must contain the following information:

1. 'Committed Stock' section containing:
    a. for each customer that has at least one order that can be completely fulfilled from current stock, but which is not yet delivered:
        i. customer number;
        ii. for each order that can be completely fulfilled from current stock, but which is not yet delivered:
            1. order number;
            2. for each flavour ordered:
                a. flavour number;
                b. quantity ordered;
2. 'Backorders' section containing:
    a. for each customer that has at least one order for which there is a stock shortage:
        i. customer number;
        ii. for each order for which there is a stock shortage:
            1. order number;
            2. for each flavour with a shortage:
                a. flavour number;
                b. quantity ordered;
                c. available quantity and percentage;
                d. shortage quantity and percentage;
3. 'Shortages' section containing:
    a. for each flavour with a shortage:
        i. flavour number;
        ii. shortage quantity;
    b. total shortage quantity;
4. 'Surplus Stock' section containing:
    a. for each flavour with a surplus:
        i. flavour number;
        ii. for each production batch with a surplus for the flavour:
            1. batch number;
            2. surplus quantity;
        iii. if more than one production batch with a surplus for the flavour:
            1. total surplus quantity;
    b. grand total surplus quantity;

A 'surplus quantity' is a quantity of stock that is not allocated to an order. Surplus quantities exclude stock that will eventually be used to fulfil an existing order. A 'shortage quantity' is a quantity of stock that has been ordered by a customer, but which cannot be delivered from current stocks.

The reporting program must allocate stock to earlier orders first. Since order numbers are consecutive from earlier orders to the later orders, the program may use them to determine the sequence of the orders. Similarly, the reporting program must allocate stock from older batches first. The reporting program may assume that batches with smaller batch numbers were produced first.

The Order Status Report should be laid out in generally the same way as the example in Figure 7. Quantities and percentages should be right-aligned. Quantities should have a comma to separate groups of thousands. For example, twelve million six hundred should be displayed as 12,000,600. Percentages should be rounded to integer values so as to ensure that the sum of the available and shortage percentages adds up to 100 and the sum of the rounding errors squared is minimised.

The 'Committed Stock' and 'Backorders' sections should be sorted by customer number, order number and then flavour number. The 'Shortages' section should be sorted by flavour number. The 'Surplus Stock' section should be sorted by flavour number and then batch number.

```
ORDER STATUS REPORT

COMMITTED STOCK
--------------------------------------------------------------------------------
CUSTOMER  ORDER     FLAVOUR    QUANTITY
123       456       789           1,000
          458       790           1,500
                    791           2,000
124       459       790           3,000

BACKORDERS
--------------------------------------------------------------------------------
CUSTOMER  ORDER     FLAVOUR       ORDER    AVAILABLE       SHORTAGE
                                QUANTITY  QUANTITY PERC  QUANTITY PERC
124       457       789           1,000       200  20%       800  80%
                    788           1,000       200  20%       800  80%

SHORTAGES
--------------------------------------------------------------------------------
FLAVOUR    QUANTITY
788             800
789             800
------------------
TOTAL         1,600

SURPLUS STOCK
--------------------------------------------------------------------------------
FLAVOUR   BATCH      QUANTITY
790       556           2,800

791       557           1,000
          558           1,200
          ------------------
          TOTAL         2,200

----------------------------
GRAND TOTAL             5,000
```

Figure 7 – Order Status Report Example

The reporting program must extract the information it needs to produce the Order Status Report from a database that contains tables for both systems. The reporting program should make use of the database indices, where appropriate, to minimise the execution time of the reporting program.

The customer order system stores data in the following tables:

```
create table custOrder (
     orderNo        integer not null,
     orderCustNo    integer not null,
     delivered      smallint not null
);
create unique index orderNoInd on
     custOrder (orderNo);
create table orderItem (
     itemOrderNo    integer not null,
     itemFlavourNo  integer not null,
     itemQty        integer not null
);
create unique index orderNoFlavourNoInd on
     orderItem (itemOrderNo, itemFlavourNo);
```

`custOrder`
: is a table that stores customer orders.

`custOrder.orderNo`
: is the order number of the order. Earlier orders have smaller order numbers.

`custOrder.delivered`
: is the flag that indicates whether the order was delivered to the customer or not. If the order is delivered, the flag is non-zero.

`orderItem`
: is a table that stores the items that customers ordered.

`orderItem.itemOrderNo`
: is the order number of the customer order that has the item.

`orderItem.itemFlavourNo`
: is the flavour number of the flavour the customer ordered.

`orderItem.itemQty`
: is the quantity of the flavour that the customer ordered.

The stock system stores data in the following tables:

```
create table stock (
     stockBatchNo   integer not null,
     stockFlavourNo integer not null,
     stockQty       integer not null
);
create unique index batchNoInd on
     stock (stockBatchNo);
```

```
create index flavourNoInd on
     stock (stockFlavourNo);
```

`stock`
: is a table that stores stock quantity of production batches.

`stock.stockBatchNo`
: is the batch number of a production batch.

`stock.stockFlavourNo`
: is the flavour number of flavour produced in the production batch.

`stock.stockQty`
: is the quantity of items from the production batch still in stock. Stock quantity is reduced after deliveries are made.

4 GREETINGS BY NAME

A company wishes to send Christmas greeting cards to its customers. The customers are listed in a text file. Each line in the file contains a name. The names do not have proper capitalization, but fortunately, are limited to only the Roman alphabetic characters (A to Z). The surname in the name is enclosed in square brackets. The square brackets are guaranteed to be correctly paired and surnames are guaranteed to be contiguous but may contain spaces. Figure 8 shows some examples of names.

```
JOHN [SMITH]
pETER [RYAN] RObbin
christine [wong wang]
saram mikah [morni]
[Chee] ky li
```

Figure 8 - Examples of Names with Surnames Underlined

You job is to write a program that corrects the capitalization of the names, extracts the surnames and displays two lines of text for each name. The first letter of each word must be a capital letter while the rest of the word should be lower case letters.

The first line of text to be displayed for each name must begin with "To:" followed by the full name. The second line should begin with the greeting "Wishing you a Merry Christmas", followed by a comma, then a title and end with the surname. The title should be "Mr" if the name belongs to a man, and "Ms" is the name belongs to a woman. If a name does not have a surname or the gender of the owner cannot be determined, the program must display the full name instead of the title and surname. Figure 9 shows an example output.

```
To: Peter Ryan Robbin
Wishing you a Merry Christmas, Mr Ryan.

To: Saram Mikah Morni
Wishing you a Merry Christmas, Ms Morni.
```

Figure 9 - Example Output

The program must accept 3 parameters. The first parameter is the file name of the file containing the names. The second parameter is the file name of a file that contains names usually given to boys. The third parameter is the file name of a file that contains names usually given to girls. Figure 10 shows an example of names usually given to boys. Figure 11 shows an example of names usually given to girls. The program may assume the names in these files are mutually exclusive and have proper capitalization.

```
John
Peter
Robbin
Ky
```

Figure 10 – Names Usually Given to Boys

```
Christine
Sarah
Li
```

Figure 11 – Names Usually Given to Girls

The program must compare the names in the customer file with the lists of names usually given to boys and girls to determine the gender of the customers. If any part of a name can be found in the list of names usually given to boys but not in the list of names usually given to girls, the program should assume that the name belongs to a man. If any part of a name can be found in the list of names usually given to girls but not in the list of names usually given to boys, the program should assume that the name belongs to a woman.

PERTANDINGAN PENGATURCARAAN E-GENTING 2014

Arahan-arahan am:

1. Jawab satu atau lebih soalan yang diberikan.
2. Peserta-peserta dibenarkan mengguna buku rujukan.
3. Masa yang diperuntukan untuk pertandingan ini adalah 8 jam.
4. Perbincangan dengan peserta lain mengenai hal-hal soalan dan jawapan tidak dibenarkan.
5. Untuk menerima markah bagi jawapan untuk sesuatu soalan, jawapan anda mestilah merupakan penyelesaian yang munasabah bagi soalan tersebut. Penyelesaian yang munasabah ialah jawapan yang menyelesaikan masalah soalan atau jawapan yang mungkin menyelesaikan masalah soalan dengan sedikit usaha tambahan.
6. Sekiranya jawapan anda adalah penyelesaian yang munasabah, anda akan menerima markah untuk hasilan pendekatan teratur seperti gambar rajah aliran data, gambar rajah peralihan keadaan, jadual dan sebagainya.
7. Jumlah markah anda adalah jumlah markah yang anda perolehi dari setiap penyelesaian yang munasabah.
8. Program-program anda akan dinilai berdasarkan betapa mudahnya kod-kod sumber boleh dibaca dan difahami.
    - Takukan mestilah bersih dan sejajar.
    - Nama-nama pembolehubah harus menggambarkan isi kandungan pembolehubah-pembolehubah berkenaan.
    - Pergandingan di antara modul-modul mestilah nyata.
    - Setiap modul harus melakukan satu perkara dengan baik.
9. Markah yang diperuntukan kepada setiap soalan adalah seperti berikut:

No	Tajuk	Markah
1.	Pengubahsuaian Kereta	450
2.	Sembunyi Dan Cari	350
3.	Laporan Status Pesanan	300
4.	Salam Dengan Nama	200

10. Kecuali dinyatakan, program anda boleh ditulis dengan menggunakan mana-mana bahasa pengaturcaraan utama di bawah mana-mana sistem operasi utama.
11. Kecuali dinyatakan, anda boleh menggunakan semua fungsi piawaian perpustakaan dalam bahasa pengaturcaraan dan sistem operasi yang anda pilih.
12. Anda TIDAK perlu menjawab semua soalan.

1 PENYESUAIAN KERETA

Syarikat Penyesuaian Kereta (CCC) mengendalikan sebuah bengkel yang memasang aksesori-aksesori dalam kereta. Pada permulaan setiap hari, pasukan jualan menyenaraikan semua arahan penyesuaian daripada pelanggan-pelanggan dalam satu fail teks yang dikenali sebagai fail penyesuaian. Setiap baris di dalam fail tersebut mewakili satu kereta dan penyesuaian kepada kereta tersebut yang dipesan oleh pelanggan. Setiap baris bermula dengan satu nombor siri kereta, diikuti dengan kata-kata pengenalan (ID) Unit Penyimpanan Stok (SKU) untuk rim, tayar, sistem audio, dan sistem GPS. Setiap nilai dipisahkan dengan satu koma dan mempunyai panjang maksima adalah 15 aksara. Nombor siri dan ID SKU terdiri daripada abjad dan angka sahaja. Nilai sesuatu ID SKU mungkin kosong bagi aksesori yang tidak perlu ditukar. Satu contoh fail penyesuaian adalah seperti dalam Figure 1.

```
T02BZOI826,AINSA222,INSW2978,297DIUW,8HD972
T02BZOI827,AINSA222,INSW2978, ,8HD972
T02BZOI828,A82IDU22,INSW2978,229FIF8676,D28HHDY
```

Figure 1 – Contoh Fail Penyesuaian

Dalam bengkel tersebut, seorang pengendali membaca fail penyesuaian dan mengkonfigurasi satu sistem robotik yang akan menanggal dan memasang aksesori-aksesori secara fizikal.

Disebabkan oleh kesilapan pengendali, kadang kala kereta dipasang dengan aksesori-aksesori yang silap dan menyebabkan kerja lewat diselesai dan kepuasan pelanggan yang rendah. Untuk mengatasi masalah ini, CCC ingin anda mengaturcarakan satu modul baru untuk membaca fail penyesuaian dan mengkonfigurasi sistem robotik secara automatik. Modul tersebut harus mempunyai pengisytiharan seperti dalam Figure 2. Modul tersebut harus menggunakan antaramuka pengaturcaraan aplikasi (API) dalam Figure 3 untuk mengawal mesin-mesin pemasangan robotik. Fungsi-fungsi dalam API tersebut adalah ‘thread-safe’.

Bengkel tersebut mempunyai 5 stesen pemasangan, tetapi sesetengah stesen mungkin tidak beroperasi apabila modul tersebut bermula. Modul tersebut boleh menganggap bahawa tiada stesen pemasangan yang akan berhenti operasi dan tiada stesen pemasangan baru yang akan bermula operasi apabila modul tersebut sedang dilaksanakan.

Sekumpulan kereta akan diatur di setiap stesen pemasangan. Modul tersebut mesti memproses kereta-kereta tersebut mengikut giliran ketibaan ke stesen pemasangan. Kereta-kereta tersebut mungkin tiba ke stesen pemasangan dalam susunan yang berbeza dengan susunan baris dalam fail penyesuaian.

Modul tersebut mesti mengesahkan bahawa semua nilai dalam fail penyesuaian adalah sah. Nombor siri kereta yang tidak dijumpai di mana-mana stesen pemasangan adalah tidak sah. Jika sesuatu nilai yang tidak sah ditemui, modul tersebut harus menulis satu mesej ralat ke dalam satu fail ralat dan kemudian mengabaikan nilai tersebut. Mesej ralat

tersebut harus mengandungi nombor siri kereta and nilai yang tidak sah tersebut. Setiap mesej ralat harus menduduki satu baris dalam fail ralat tersebut.

Modul tersebut mesti mengarah sistem robotik untuk menanggalkan aksesori-aksesori sedia ada yang tidak diperlukan atau yang perlu digantikan sebelum ia memasang aksesori-aksesori baru. Tambahan juga, rim mesti ditanggalkan sebelum tayar boleh dipasang atau ditanggal. Rim tidak boleh dipasang sebelum tayar dipasangkan pada rim.

Modul tersebut harus memanggil fungsi `InstallAccessory` dengan satu jenis aksesori dan satu ID SKU yang sah untuk pemasangan aksesori. Modul tersebut harus memanggil fungsi `RemoveAccessory` dengan satu jenis aksesori untuk penanggalan aksesori.

Jika aksesori yang diperlukan telah dipasang, modul tersebut tidak harus menanggal atau memasang semula aksesori tersebut. Jika sesebuah kereta tidak memerlukan sebarang penyesuaian, modul tersebut mesti mengeluarkan kereta tersebut daripada stesen pemasangan tanpa melakukan sebarang pemasangan atau penanggalan. Modul tersebut mesti juga mengeluarkan kereta daripada stesen pemasangan apabila semua penyesuaian telah siap.

Sebelum memasang atau menanggal aksesori atau mengeluarkan sesebuah kereta, modul tersebut mesti mengesahkan bahawa stesen pemasangan sedia menerima arahan. Jika sesebuah stesen pemasangan belum sedia, modul tersebut harus menunggu selama satu saat dan cuba menguji status stesen pemasangan tersebut sekali lagi. Modul tersebut harus mengulangi proses ini sehingga stesen pemasangan itu telah sedia.

Modul tersebut harus mengguna sepenuhnya semua stesen-stesen pemasangan yang sedia ada secara serentak.

C++

```
void PerformCustomisation (const char *file, const char *errorFile, IMachines *machines)
{
}
```

Java

```
public class Workshop {
        public static void performCustomisation (String file, String errorFile,
                Machines machines)
        {
        }
}
```

C#

```
public class Workshop {
        public static void PerformCustomisation (string file, string errorFile,
                Machines machines)
        {
        }
}
```

Penerangan

`File`
merupakan nama fail bagi fail penyesuaian

`errorFile`
merupakan nama fail bagi fail di mana mesej ralat akan dituliskan

`machines`
merupakan objek yang melaksanakan API

Figure 2 – Pengisytiharan Modul

C++

```
enum AccessoryType {
        RIM     = 1,
        TYRE    = 2,
        AUDIO   = 3,
        GPS     = 4,
};
class Accessory {
public:
        AccessoryType  accessoryType;         // Accessory type
        const char     skuId[16];             // SKU identifier
};
class IMachines {
public:
        virtual int GetServiceableStations(int *stations) = 0;
                                        // Get list of serviceable assembly stations
        virtual bool IsStationReady (int stationId) = 0;
                                        // Test whether the assembly station is ready
                                        // for the next request
        virtual bool GetCarNumber (int stationId, const char *carSerNo) = 0;
                                        // Get the serial number of the car currently
                                        // at the assembly station
        virtual int GetAccessories (int stationId,
```

```
                Accessory *accessoryArray) = 0;
                                        // Get a list of the accessories installed in
                                        // the car currently at the assembly station
        virtual bool IsValidAccessory (AccessoryType accessoryType,
                const char *skuId) = 0;
                                        // Test whether a SKU identifier is valid
        virtual void InstallAccessory (int stationId,
                AccessoryType accessoryType, const char *skuId) = 0;
                                        // Install an accessory in the car currently at
                                        // the assembly station
        virtual void RemoveAccessory (int stationId,
                AccessoryType accessoryType) = 0;
                                        // Remove an accessory from the car currently at
                                        // the assembly station
        virtual void Eject (int stationId) = 0;
                                        // Eject the car from the assembly station and move
                                        // the next car in the queue to the assembly station
};
```

Java

```
public enum AccessoryType {
        RIM,
        TYRE,
        AUDIO,
        GPS,
}
public class Accessory {
        public AccessoryType   accessoryType;         // Accessory type
        public String          skuId;                 // SKU identifier
}
public interface Machines {
        int[] getServiceableStations();
                                        // Get list of serviceable assembly stations
        boolean isStationReady (int stationId);
                                        // Test whether the assembly station is ready
                                        // for the next request
        String getCarNumber (int stationId);
                                        // Get the serial number of the car currently
                                        // at the assembly station
        Accessory [] getAccessories (int stationId);
                                        // Get a list of the accessories installed in
                                        // the car currently at the assembly station
        boolean isValidAccessory (AccessoryType accessoryType, String skuId);
                                        // Test whether a SKU identifier is valid
        void installAccessory (int stationId, AccessoryType accessoryType,
                String skuId);
                                        // Install an accessory in the car currently at
                                        // the assembly station
        void removeAccessory (int stationId, AccessoryType accessoryType);
                                        // Remove an accessory from the car currently at
                                        // the assembly station
        void eject (int stationId);
                                        // Eject the car from the assembly station and move
                                        // the next car in the queue to the assembly station
}
```

C#

```
public enum AccessoryType {
        RIM     = 1,
        TYRE    = 2,
        AUDIO   = 3,
        GPS     = 4,
}
public class Accessory {
        public AccessoryType   AccessoryType;         // Accessory type
        public string          SkuId;                 // SKU identifier
}
public interface Machines {
        int[] GetServiceableStations();
                                        // Get list of serviceable assembly stations
        bool IsStationReady (int stationId);
                                        // Test whether the assembly station is ready
                                        // for the next request
        string GetCarNumber (int stationId);
                                        // Get the serial number of the car currently
                                        // at the assembly station
        Accessory [] GetAccessories (int stationId);
                                        // Get a list of the accessories installed in
                                        // the car currently at the assembly station
        bool IsValidAccessory (AccessoryType accessoryType, string skuId);
                                        // Test whether a SKU identifier is valid
        void InstallAccessory (int stationId, AccessoryType accessoryType,
                string skuId);
                                        // Install an accessory in the car currently at
                                        // the assembly station
        void RemoveAccessory (int stationId, AccessoryType accessoryType);
                                        // Remove an accessory from the car currently at
                                        // the assembly station
        void Eject (int stationId);
                                        // Eject the car from the assembly station and move
                                        // the next car in the queue to the assembly station
}
```

Penerangan

`GetServiceableStations`

Memulangkan satu tata susunan ID stesen pemasangan yang mengenalpasti stesen-stesen pemasangan yang boleh beroperasi. Versi C++ menulis ID stesen pemasangan sebagai satu tata susunan di lokasi yang dirujuk oleh petunjuk `stations` dan memulangkan bilangan elemen yang ditulis.

`IsStationReady`

Memulangkan nilai benar jika stesen pemasangan yang dikenali oleh ID tersebut adalah bersedia untuk memproses arahan seterusnya.

`GetCarNumber`

Memulangkan nombor siri kereta yang berada pada stesen pemasangan yang dikenalpasti oleh ID stesen pemasangan tersebut. Jika semua kereta yang diatur kepada stesen tersebut sudah siap diproses, null akan dipulangkan. Dalam fungsi versi C++, jika sesebuah kereta berada pada stesen pemasangan, fungsi tersebut memuatkan nombor siri kereta ke dalam rentetan yang dirujuk oleh pentunjuk

`carSerNo` dan memulangkan benar. Jika semua kereta dalam giliran telah diproses, ia memulangkan tidak benar.

`GetAccessories`
: Memulangkan satu tata susunan aksesori yang telah dipasang pada kereta yang berada dalam stesen pemasangan yang dikenalpasti dengan ID stesen pemasangan tersebut. Versi C++ menulis aksesori ke dalam tata susunan yang dirujuk oleh petunjuk `accessoryArray` dan memulangkan bilangan aksesori yang telah ditulis.

`IsValidAccessory`
: Memulangkan nilai benar jika ID SKU adalah ID yang sah untuk jenis aksesori yang diberi.

`InstallAccessory`
: Memasang aksesori dalam kereta yang berada pada stesen pemasangan yang dikenalpasti oleh ID yang diberi.

`RemoveAccessory`
: Menanggal aksesori dalam kereta yang berada pada stesen pemasangan yang dikenalpasti oleh ID yang diberi.

`Eject`
: Mengeluarkan kereta daripada stesen pemasangan yang dikenalpasti oleh ID yang diberi dan menggerakkan kereta seterusnya dalam giliran (jika ada) ke stesen pemasangan.

`stationId`
: merupakan integer 32-bit yang mengenalpasti sesebuah stesen pemasangan secara unik.

`accessoryType`
: merupakan nilai integer yang mengenalpasti jenis aksesori.

`skuId`
: merupakan satu rentetan yang mengenalpasti SKU.

Figure 3 – Antaramuka Pengaturcaraan Aplikasi Mesin

2 SEMBUNYI DAN CARI

Sebuah agensi maklumat mempunyai petugas yang perlu membawa maklumat sensitif. Agensi tersebut inginkan satu produk yang membolehkan petugas-petugasnya membawa maklumat tanpa dikesan dengan mudah. Penyulitan maklumat adalah langkah yang baik tetapi fail yang penuh dengan maklumat tersulit dalam pemacu USB bukan sesuatu yang mudah diterangkan.

Mereka berpendapat bahawa adalah mungkin untuk menyembunyi maklumat dalam format tersulit pada ruang yang tidak digunakan oleh sistem fail dan kemudian memulihkan maklumat tersebut dengan mengimbas media tersebut dari mula hingga akhir. Dengan cara ini, jika seseorang petugas ditahan, beliau boleh menafi dan menegaskan maklumat rawak pada ruang yang tidak digunakan oleh sistem fail adalah kesan daripada proses pemadaman sistem fail untuk memastikan tiada data yang dibawa.

Agensi tersebut mengehendaki anda mengaturcarakan 2 aturcara, satu dipanggil ‘sembunyi’ dan satu lagi dipanggil ‘cari’ untuk mencapai objektif ini.

Aturcara ‘sembunyi’ mesti menerima satu aliran teks biasa daripada aliran input piawainya. Sebelum menyulit teks biasa dan menulis teks rahsia ke sistem fail, aturcara ‘sembunyi’ mesti mencipta satu fail sementara dan mengisi fail tersebut dengan data rawak sehingga sistem fail telah penuh. Selepas itu, bagi setiap muka surat teks biasa yang tersulit, ia mesti menulis 8 salinan teks rahsia pada lokasi rawak dalam fail tersebut di mana lokasi tersebut tidak digunakan untuk menyimpan muka surat teks rahsia yang lain. Setiap muka surat teks rahsia mesti mempunyai struktur seperti dalam Figure 4. 8 salinan muka surat tersebut mengurangkan risiko di mana sistem fail menulis ganti semua salinan bagi sesuatu muka surat. Nombor salinan yang berbeza menghalang salinan-salinan untuk sesuatu muka surat menghasilkan text rahsia yang sama, di mana ia akan memberi petunjuk kepada penganalisa penyulitan. Apabila aturcara sembunyi tersebut habis menulis ke fail sementara, ia mesti memadam fail tersebut dari sistem fail.

Aturcara ‘cari’ tersebut mesti membuka media sistem fail seolah-olah media tersebut adalah satu fail, membaca setiap muka surat dan cuba mengesahkan tandatangan digital muka surat tersebut. Jika tandatangan digital tersebut berjaya disahkan, aturcara ‘cari’ tersebut harus menganggap muka surat tersebut adalah sebahagian daripada dokumen yang sedang dipulihkan. Ia harus memulihkan struktur dalam Figure 5 dan meletakkan teks biasa tersebut pada kedudukan yang betul dalam dokumen yang dipulihkan itu.

Apabila dokumen tersebut dipulihkan sepenuhnya, aturcara ‘cari’ tersebut harus memaparkan dokumen tersebut ke aliran output piawainya.

Aturcara ‘cari’ tersebut harus mengenalpasti muka surat yang hilang dan memaparkan amaran yang mengandungi nombor muka surat tersebut.

Kedudukan Bait	Kandungan
0 hingga 991	Nilai disulit bagi struktur dalam Figure 5
992 hingga 1003	Data Rawak
1004 hingga 1023	Tandatangan Digital

Figure 4 – Struktur Muka Surat yang Tersulit

Kedudukan Bait	Kandungan
0 hingga 3	Nombor Muka Surat
4 hingga 5	Nombor Salinan
6 hingga 7	Saiz teks biasa dalam bait
8 hingga 975	Teks Biasa, diisi dengan data rawak jika perlu

Figure 5 – Bahagian Teks Biasa Sebelum Disulit

Nombor muka surat dalam bahagian teks biasa yang belum disulit mesti bermula dari sifar dan meningkat mengikut urutan bagi setiap muka surat teks biasa.

Ambil perhatian bahawa bahagian teks biasa yang belum disulit adalah sepanjang 976 bait manakala panjang selepas disulit adalah 992 bait. Nilai disulit mempunyai panjang tambahan sebanyak 16 bait untuk menampung satu vector permulaan rawak yang digunakan dalam proses penyulitan CBC (cipher block chaining).

Seperti disebut sebelum ini, nombor salinan mesti mempunyai nilai berbeza bagi setiap salinan disulit untuk sesuatu muka surat teks biasa. Untuk menggagalkan serangan teks biasa yang diketahui, aturcara ‘sembunyi’ tersebut harus memilih nombor-nombor salinan yang rawak dan bukan nombor-nombor yang mengikut urutan teks biasa.

Saiz teks biasa dalam bait merupakan bilangan bait dalam medan teks biasa yang sebenarnya digunakan. Bagi semua muka surat kecuali muka surat terakhir dalam dokumen, saiz teks biasa ialah 968. Untuk muka surat terakhir, saiz teks biasa merupakan bilangan bait dalam muka surat terakhir yang sebenarnya digunakan. Aturcara ‘sembunyi’ tersebut mesti mengisi ruang baki dengan data rawak.

Aturcara ‘sembunyi’ dan ‘cari’ tersebut mesti menggunakan fungsi-fungsi dalam Figure 6 untuk melakukan operasi penyulitan dan tandatangan digital.

Aturcara-aturcara tersebut mesti menerima satu kunci penyulitan yang digunakan oleh operasi penyulitan dan tandatangan digital. Kunci penyulitan tersebut merupakan satu rentetan teks. Aturcara-aturcara ‘sembunyi’ dan ‘cari’ harus menerima hanya satu kunci

penyulitan daripada pengguna and memberi kunci tersebut kepada kedua-dua operasi penyulitan dan tandatangan digital.

Untuk kegunaan projek ini, aturcara ‘cari’ boleh mencapai media sistem fail dengan membuka fail pseudo ‘wholedisk’.

C++

```
void Encrypt(const char *key, const void *plaintext, void *ciphertext);
                                        // Encrypt the plaintext into ciphertext
void Decrypt(const char *key, const void *ciphertext, void *plaintext);
                                        // Decrypt the ciphertext into plaintext
void GetSignature(const char *key, const void *data, void *signature);
                                        // Derive digital signature
unsigned char GetRandomByte ();
                                        // Get a cryptographically secure random byte
```

Java

```
public class Util {
        static public void Encrypt(String key, byte[] plaintext, byte[] ciphertext);
                                        // Encrypt the plaintext into ciphertext
        static public void Decrypt(String key, byte[] ciphertext, byte[] plaintext);
                                        // Decrypt the ciphertext into plaintext
        static public void GetSignature(String key, byte[] data, byte[] signature);
                                        // Derive digital signature
        static public byte GetRandomByte ();
                                        // Get a cryptographically secure random byte
}
```

C#

```
public class Util {
        static public void Encrypt(string key, byte[] plaintext, byte[] ciphertext);
                                        // Encrypt the plaintext into ciphertext
        static public void Decrypt(string key, byte[] ciphertext, byte[] plaintext);
                                        // Decrypt the ciphertext into plaintext
        static public void GetSignature(string key, byte[] data, byte[] signature);
                                        // Derive digital signature
        static public byte GetRandomByte ();
                                        // Get a cryptographically secure random byte
}
```

Penerangan

`Encrypt`
: menyulit `plaintext` ke `ciphertext`.

`Decrypt`
: Memulih `ciphertext` ke `plaintext`.

`GetChecksum`
: Mengira checksum untuk `data` dan simpan dalam `checksum`.

`GetRandomByte`
: Menjana dan mengembalikan satu bait rawak yang selamat dari segi penyulitan.

Figure 6 – Fungsi-fungsi Penyulitan

3 LAPORAN STATUS PESANAN

Satu kilang minuman ringan menggunakan satu sistem untuk menjejaki pesanan pelanggan dan satu sistem yang lain untuk menjejaki stok. Tugas anda ialah untuk mengaturcara satu aturcara laporan yang mengabungkan data dari dua sistem ini ke dalam satu Laporan Status Pesanan.

Laporan Status Pesanan tersebut mesti mengandungi maklumat seperti berikut:

1. Bahagian 'Committed Stock' mengandungi:
    a. bagi setiap pelanggan yang mempunyai sekurang-kurangnya satu pesanan yang boleh dipenuhi sepenuhnya dari stok semasa, tetapi masih belum dihantar:
        i. nombor pelanggan;
        ii. bagi setiap pesanan yang boleh dipenuhi sepenuhnya dari stok semasa, tetapi masih belum dihantar:
            1. nombor pesanan;
            2. bagi setiap jenis perisa yang dipesan:
                a. nombor jenis perisa;
                b. kuantiti dipesan;
2. Bahagian 'Backorders' mengandungi:
    a. bagi setiap pelanggan yang mempunyai sekurang-kurangnya satu pesanan yang tiada stok yang cukup:
        i. nombor pelanggan;
        ii. bagi setiap pesanan yang tiada stok yang cukup:
            1. nombor pesanan;
            2. bagi setiap jenis perisa yang tidak cukup stok:
                a. nombor jenis perisa;
                b. kuantiti yang dipesan;
                c. kuantiti dan peratusan yang sedia ada;
                d. kuantiti dan peratusan yang tidak cukup;
3. Bahagian 'Shortages' mengandungi:
    a. bagi setiap jenis perisa yang tidak cukup:
        i. nombor jenis perisa;
        ii. kuantiti yang tidak cukup;
    b. jumlah kuantiti yang tidak cukup;
4. Bahagian 'Surplus Stock' mengandungi:
    a. bagi setiap jenis perisa dengan lebihan:
        i. nombor jenis perisa;
        ii. bagi setiap kumpulan pengeluaran dengan lebihan untuk jenis perisa tersebut:
            1. nombor kumpulan;
            2. kuantiti lebihan;
        iii. jika lebih daripada satu kumpulan pengeluaran dengan lebihan untuk jenis perisa tersebut:
            1. jumlah kuantiti lebihan;
    b. jumlah besar kuantiti lebihan;

‘Kuantiti lebihan’ ialah kuantiti stok yang tidak diperuntuk bagi pesanan. Kuantiti lebihan tidak termasuk stok yang akhirnya digunakan untuk memenuhi sesuatu pesanan yang sedia ada. ‘Kuantiti tidak cukup’ ialah kuantiti stok yang telah dipesan oleh seseorang pelanggan, tetapi tidak dapat dipenuhi dari stok yang sedia ada.
Aturcara laporan tersebut mesti memperuntukkan stok kepada pesanan awal terlebih dahulu. Memandangkan nombor-nombor pesanan adalah mengikut urutan dari yang awal ke yang kemudian, aturcara tersebut boleh menggunakannya untuk menentukan susunan pesanan. Aturcara laporan tersebut juga mesti memperuntukkan stok dari kumpulan awal terlebih dahulu. Aturcara laporan tersebut boleh menganggap bahawa kumpulan dengan nombor kumpulan yang lebih kecil adalah kumpulan yang dihasilkan dahulu.

```
ORDER STATUS REPORT

COMMITTED STOCK
------------------------------------------------------------------------------
CUSTOMER  ORDER     FLAVOUR    QUANTITY
123       456       789           1,000
          458       790           1,500
                    791           2,000
124       459       790           3,000


BACKORDERS
------------------------------------------------------------------------------
CUSTOMER  ORDER     FLAVOUR       ORDER    AVAILABLE       SHORTAGE
                                QUANTITY  QUANTITY PERC  QUANTITY PERC
124       457       789           1,000       200  20%       800  80%
                    788           1,000       200  20%       800  80%

SHORTAGES
------------------------------------------------------------------------------
FLAVOUR    QUANTITY
788             800
789             800
------------------
TOTAL         1,600

SURPLUS STOCK
------------------------------------------------------------------------------
FLAVOUR   BATCH      QUANTITY
790       556           2,800

791       557           1,000
          558           1,200
          ------------------
          TOTAL         2,200

----------------------------
GRAND TOTAL             5,000
```

Figure 7 – Contoh Laporan Status Pesanan

Laporan Status Pesanan harus disusun seperti dalam Figure 7. Nilai-nilai kuantiti dan peratusan harus dijajarkan ke kanan. Nilai-nilai kuantiti harus mempunyai koma untuk memisahkan ribuan. Contohnya, dua belas juta enam ratus harus dipaparkan sebagai 12,000,600. Nilai-nilai peratusan harus dibundarkan kepada integer dan memastikan

jumlah peratusan stok sedia ada dengan yang tidak cukup adalah 100 dan mengurangkan jumlah ralat pembundaran kuasa dua hingga minima.

Bahagian- bahagian ‘Committed Stock’ dan ‘Backorders’ harus disusun mengikut nombor pelanggan, nombor pesanan dan nombor jenis perisa. Bahagian ‘Shortages’ harus disusun mengikut nombor jenis perisa. Bahagian ‘Surplus Stock’ harus disusun mengikut nombor jenis perisa dan nombor kumpulan.

Aturcara laporan tersebut mesti mengekstrak maklumat yang diperlukan untuk menghasilkan Laporan Status Pesanan dari satu pangkalan data yang mengandungi jadual-jadual bagi kedua-dua sistem tersebut. Aturcara tersebut harus menggunakan indeks-indeks pangkalan data di mana yang sesuai supaya mengurangkan masa pelaksanaan aturcara laporan tersebut.

Sistem pesanan pelanggan menyimpan data dalam jadual-jadual berikut:

```
create table custOrder (
     orderNo        integer not null,
     orderCustNo    integer not null,
     delivered      smallint not null
);
create unique index orderNoInd on
     custOrder (orderNo);
create table orderItem (
     itemOrderNo    integer not null,
     itemFlavourNo  integer not null,
     itemQty        integer not null
);
create unique index orderNoFlavourNoInd on
     orderItem (itemOrderNo, itemFlavourNo);
```

`custOrder`
: ialah satu jadual yang menyimpan pesanan-pesanan pelanggan.

`custOrder.orderNo`
: ialah nombor pesanan untuk pesanan. Pesanan yang lebih awal mempunyai nilai yang lebih kecil.

`custOrder.delivered`
: ialah nilai yang menunjukkan sama ada pesanan telah dihantar kepada pelanggan. Jika pesanan telah dihantar, nilai ini bukanlah sifar.

`orderItem`
: ialah satu jadual yang menyimpan butir-butir yang dipesan oleh pelanggan-pelanggan.

`orderItem.itemOrderNo`
: ialah nombor pesanan bagi pesanan pelanggan yang mengandungi butir tersebut.

`orderItem.itemFlavourNo`
: ialah nombor jenis perisa bagi jenis perisa yang dipesan oleh pelanggan.

`orderItem.itemQty`
: ialah kuantiti jenis perisa yang dipesan oleh pelanggan.

Sistem stok menyimpan data dalam jadual-jadual berikut:

```
create table stock (
     stockBatchNo   integer not null,
     stockFlavourNo integer not null,
     stockQty       integer not null
);
create unique index batchNoInd on
     stock (stockBatchNo);
create index flavourNoInd on
     stock (stockFlavourNo);
```

`stock`
: ialah satu jadual yang menyimpan kuantiti stok bagi kumpulan pengeluaran.

`stock.stockBatchNo`
: ialah nombor kumpulan bagi sesuatu kumpulan pengeluaran.

`stock.stockFlavourNo`
: ialah nombor jenis perisa bagi jenis perisa yang dihasilkan dalam kumpulan pengeluaran.

`stock.stockQty`
: ialah kuantiti daripada kumpulan pengeluaran yang masih dalam simpanan. Kuantiti stok berkurang selepas penghantaran dibuat.

4 SALAM DENGAN NAMA

Satu syarikat ingin menghantar kad ucapan Krismas kepada pelanggan-pelanggannya. Pelanggan-pelanggannya disenaraikan dalam satu fail teks. Setiap baris dalam fail tersebut mengandungi satu nama. Nama-nama tersebut tidak mempunyai huruf besar/kecil yang betul tetapi nasib baik semua nama hanya mengandungi aksara abjad Roman sahaja (A hingga Z). Nama keluarga dalam nama adalah dikurung dengan aksara kurungan persegi. Kurungan persegi tersebut dijamin dalam pasangan yang betul dan nama keluarga dijamin saling berdampingan tetapi mungkin mengandungi ruang kosong. Figure 8 menunjukkan beberapa contoh nama:

```
JOHN [SMITH]
pETER [RYAN] RObbin
christine [wong wang]
saram mikah [morni]
[Chee] ky li
```

Figure 8 – Contoh-contoh Nama Dengan Nama Keluarga Digariskan

Tugas anda ialah mengaturcara satu aturcara yang membetulkan huruf besar/kecil untuk nama-nama yang diberi, mengekstrak nama keluarga dan memaparkan dua baris teks bagi setiap nama. Aksara pertama bagi setiap perkataan mestilah dalam huruf besar manakala yang lainnya harus dalam huruf-huruf kecil.

Baris pertama bagi teks yang dipaparkan bagi setiap nama mesti bermula dengan "To:" diikuti dengan nama penuh tersebut. Baris kedua harus bermula dengan ucapan "Wishing you a Merry Christmas", diikuti dengan koma, kemudian satu tajuk dan diakhiri dengan nama keluarga tersebut. Tajuk tersebut ialah "Mr" jika nama tersebut nama lelaki, dan "Ms" jika nama tersebut nama perempuan. Jika sesuatu nama tidak mempunyai nama keluarga atau jantina pemilik nama tidak dapat dikenalpasti, aturcara tersebut mesti memaparkan nama penuh sebagai ganti kepada tajuk dan nama keluarga. Figure 9 menunjukkan satu contoh output:

```
To: Peter Ryan Robbin
Wishing you a Merry Christmas, Mr Ryan.

To: Saram Mikah Morni
Wishing you a Merry Christmas, Ms Morni.
```

Figure 9 – Contoh Output

Aturcara tersebut mesti menerima 3 parameters. Parameter pertama ialah nama fail bagi fail yang mengandungi senarai nama. Parameter kedua ialah nama fail bagi fail yang mengandungi nama yang biasanya diberi kepada lelaki. Parameter ketiga ialah nama fail bagi fail yang mengandungi nama yang biasanya diberi kepada perempuan.
Figure 10 menunjukkan satu contoh senarai nama yang biasanya diberi kepada lelaki.

Figure 11 menunjukkan satu contoh senarai nama yang biasanya diberi kepada perempuan. Aturcara tersebut boleh menganggap bahawa nama-nama dalam fail-fail tersebut adalah tidak bertindih dan mempunyai huruf besar/kecil yang betul.

```
John
Peter
Robbin
Ky
```

Figure 10 – Nama-nama yang Biasanya Diberi Kepada Lelaki

```
Christine
Sarah
Li
```

Figure 11 – Nama-nama yang Biasanya Diberi Kepada Perempuan

Aturcara tersebut mesti membandingkan nama-nama dalam fail pelanggan dengan senarai nama yang biasa diberi kepada lelaki dan perempuan untuk menentukan jantina pelanggan. Jika sebarang bahagian nama ditemui dalam senarai nama yang biasa diberi kepada lelaki tetapi tidak ditemui dalam senarai nama yang biasa diberi kepada perempuan, maka aturcara tersebut harus menganggap nama tersebut milik kepada seorang lelaki. Jika sebarang bahagian nama ditemui dalam senarai nama yang biasa diberi kepada perempuan tetapi tidak ditemui dalam senarai nama yang biasa diberi kepada lelaki, maka aturcara tersebut harus menganggap nama tersebut milik kepada seorang perempuan.