E-GENTING
PROGRAMMING COMPETITION 2005

General instructions:

1. Answer one or more of the questions.
2. The competition is an open book test.
3. The duration for 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 will be the sum of the credit you received 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 marks allocated for the questions are as follows:

No	Name	Marks
1.	Mytel Mobile	250
2.	Project Scheduling Program	350
3.	Failsafe Storage	100
4.	Vending Machine Multiplexer	500

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 MYTEL MOBILE

Mytel Mobile operates a mobile telephone service across a number of Southeast Asian nations. It has a large customer base of around two million subscribers, many of whom roam from one country to another.

For around five years, Mytel has been operating a loyalty programme in which subscribers earn points for making telephone calls over the mobile service. The subscribers can use their points to buy a variety of goods and services.

Mytel generally purchases the goods and services from other businesses. Mytel endeavours to set up agreements with the other businesses that allow Mytel to buy the goods and services at discounted prices. Mytel argues that it is entitled to a discount because it is helping to promote the other business.

Mytel wants to set up such an agreement with one of a number of airlines that are major carriers in the regions that Mytel covers.

Mytel now needs to convince the airlines that a significant proportion of its subscribers are regular travellers and that those regular travellers have enough points to buy a significant number of tickets.

Your task is to write a reporting program that analyses Mytel's call history database and displays statistics that the airlines might find interesting.

The reporting program must accept a reporting period expressed as a from-date and a to-date and generate a report that has the following contents:

1. date and time the report was generated;
2. reporting period (from-date and to-date);
3. for each point balance range:
    a. the point balance range;
    b. the total number of subscribers on the database with a point balance in the range that registered one or more calls in the reporting period ('the selected scribers');
    c. for each major city in the Mytel coverage area (the 'first city'):
        ~~i. the number of selected subscribers who were based in the first city;~~
        ii. for each other major city in the Mytel coverage area (the 'other city'):
            1. the number of times the selected subscribers who were based in the first city and visited the other city.
    d. the total number of trips made by the selected subscribers;
4. totals for all point balance ranges:
    a. the total number of subscribers who registered calls in the reporting period (the 'active subscribers');
    b. for each major city in the Mytel coverage area (the 'first city'):
        ~~i. the number of active subscribers who were based in the first city;~~
        ii. for each other major city in the Mytel coverage area (the 'other city'):

1. the number of times active subscribers who were based in the first city and visited the other city.

c. the total number of trips made by all active subscribers.

The reporting program must accept the from-date and to-date in DD-MM-YY format. Years between 70 and 99 must be assumed to be in the 20th century and years between 00 and 69 must be assumed to be in the 21st century.

The reporting program should be initiated by a command line similar to:

```
MytelRep 01-01-05 30-06-05
```

Where '`01-01-05`' is the from-date and '`30-06-05`' is the to-date.

The reporting period must include the from-date and to-date. For example, the reporting period for the preceding command line is 00:00 on the 1 January 2005, until 23:59 on 30 June 2005 inclusive.

The point balance ranges are 0 to 99,999 points, 100,000 to 299,999 points and 300,000 points or more.

The major cities are Kuala Lumpur, Singapore and Bangkok. The cities are identified by their airport codes: KUL, SIN and BKK respectively.

The report must be formatted in accordance with the following layout:

```
         1         2         3         4         5         6         7         8
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0....5....0
DD-MM-YY HH:MM          INTER-CITY TRIP ANALYSIS REPORT                   PAGE X
                           for XX-XX-XX to XX-XX-XX

                     Number of   Base          Number of trips to         Total no
Point range        subscribers   city       KUL         SIN         BKK   of trips

0 to 99,999          X,XXX,XXX   KUL            0  X,XXX,XXX  X,XXX,XXX  X,XXX,XXX
                                 SIN  X,XXX,XXX            0  X,XXX,XXX
                                 BKK  X,XXX,XXX  X,XXX,XXX            0

100,000 to 299,999   X,XXX,XXX   KUL            0  X,XXX,XXX  X,XXX,XXX  X,XXX,XXX
                                 SIN  X,XXX,XXX            0  X,XXX,XXX
                                 BKK  X,XXX,XXX  X,XXX,XXX            0

300,000 or more      X,XXX,XXX   KUL            0  X,XXX,XXX  X,XXX,XXX  X,XXX,XXX
                                 SIN  X,XXX,XXX            0  X,XXX,XXX
                                 BKK  X,XXX,XXX  X,XXX,XXX            0

Total                X,XXX,XXX   KUL            0  X,XXX,XXX  X,XXX,XXX  X,XXX,XXX
                                 SIN  X,XXX,XXX            0  X,XXX,XXX
                                 BKK  X,XXX,XXX  X,XXX,XXX            0
```

The reporting program must garner the information in the report from Mytel's call register.

The call register is an SQL database with the following schema:

```
create table subscribers (
     subId          integer not null,
     subName        char(40) not null,
     subBal         integer not null
);
create unique index subIdInd on
     subscribers (subId);

create table callRegister (
     crSubId        integer not null,
     crFromCity     char(3) not null,
     crCalledNo     char(20) not null,
     crConnectTime  integer not null,
     crDisconTime   integer not null
);
create index callRegTimeInd on
     callRegister (crConnectTime);
```

`subscribers`
: is a table that contains one row for each subscriber.

`subId`
: is a subscriber identifier that uniquely identifies each subscriber.

`subName`
: is the subscriber's name.

`subBal`
: is the subscriber's point balance.

`callRegister`
: is a table that contains one row for each telephone call made by each subscriber.

`crSubId`
: is the subscriber identifier of the subscriber who made the call.

`crFromCity`
: is the airport code of the city from which the call was made.

`crCalledNo`
: is the telephone number that was called.

`crConnectTime`
: is the time the connection was made in seconds since 00:00 UTC on 1 January 1970.

`crDisconTime`
: is the time the connection was broken in seconds since 00:00 UTC on 1 January 1970.

The call register is a huge table. It contains over 45 billion rows! It is too big for the database management system to sort. It can only be read in connection time order.

Any attempt to randomly select a row from the call register results in an unacceptable execution time. For example, software engineers at Mytel have calculated that it would take several years to execute the following SQL statement for every customer on the database.

```
select    crConnectTime, crFromCity
from      callRegister
where     crSubId = :subId and
          crConnectTime between :fromDate and :toDate
order     by crConnectTime;
```

In practice, the call register can only be scanned sequentially in connection time order.

Nevertheless, the computer that will run the reporting program is quite large. The reporting program can use up to 150Mb of RAM.

The reporting program must determine each subscriber's base city by assuming that the subscriber's base city is the city from which he made the greatest number of calls.

The reporting program must determine the number of trips made by each subscriber by scanning the call register and counting a new trip whenever a call is made from a new city other than the base city. For example if KUL is the subscriber's base city and calls are made from KUL, SIN, KUL, BKK and then KUL, the reporting program must count two trips, one to Singapore and the other to Bangkok.

If a subscriber makes a trip from one non-base city to another non-base city, the leg must be counted as a trip. For example, if a subscriber based in Singapore made calls from SIN, KUL, BKK, KUL and then SIN, the reporting program must count three trips, one to Kuala Lumpur, one to Bangkok and then another back to Kuala Lumpur. The leg back to Singapore must not be counted as a trip.

Although, Mytel primarily operates in Kuala Lumpur, Singapore and Bangkok, subscribers occasionally make calls from other cities. If the reporting program encounteres a city code other than 'KUL', 'SIN' or 'BKK' on the database, it should quietly ignore the call register row with the unrecognised city code.

2 PROJECT SCHEDULING PROGRAM

Your company requires a program to schedule the phases of projects involving multiple programmers. Your task is to write the scheduling program.

The scheduling program must accept an input file name as an argument, read and analyse the input file and generate a project schedule listing.

A typical input file might look like this:

```
start 30-08-05;          # Project start date
holidays {               # Public holidays
     31-08-05,                # Hari merdeka
     01-11-05                 # Deepavali
}
tasks {                  # Task list
     func      "Functional specification"    MDJ  10;
     design    "Internal design"             AB   12
          needs func;
     screens   "Program data entry screens"  JOJ  8
          needs design;
     reports   "Program reports"             AB   10
          needs design;
     tests     "Acceptance tests"            MDJ  12
          needs screens, reports;
}
```

Text starting with a hash-character (#) and continuing to the end of the line is a comment. The scheduling program must ignore comments in the input file.

Except for comments, the content of the input file is divided into tokens.

There are five types of tokens. The token types are keywords such as '`start`' and '`holidays`', dates such as '`30-08-05`', identifiers such as '`screens`', strings such as "`Program reports`" and numbers.

Zero or more white space characters (i.e. space, tab and new-line) may separate consecutive tokens. If two consecutive tokens would be indistinguishable were it not for white space between them (for example 'needs' followed by 'design' in the above example) the input file is guaranteed to contain at least one white space character between the tokens.

Dates are in DD-MM-YY format.

Identifiers contain only upper and lower-case letters, digits and the underscore and dash characters. The scheduling program is not required to process identifiers more than 10 characters long.

Identifiers, numbers and dates are indistinguishable at the lexical level. They cannot be distinguished from each other until the parser recognises their positions in the syntactic sequence.

Strings are enclosed in double-quotes. Inside a string, the escape sequences '`\"`' and '`\\`' represent a double-quote and a back-slash respectively. The scheduling program is not required to process strings longer than 40 characters. Strings may not contain new-line characters.

The format of the input file in BNF is:

input_file:
    start_stmt holidays_stmt tasks_stmt

start_stmt:
    `start` *start_date* `;`

holidays_stmt:
    `holidays { `*holiday_list*$_{opt}$` }`

holiday_list:
    holiday_date
    holiday list `,` *holiday_date*

tasks_stmt:
    `tasks { `*task_list*$_{opt}$` }`

task_list:
    task_definition
    task_list task_definition

task_definition:
    task_name task_description person duration needs_clause$_{opt}$ `;`

needs_clause:
    `needs` *needs_list*

needs_list:
    task_name
    needs_list `,` *task_name*

The *start_date* is the date of the first day on which work is to be done, subject to it not being a holiday.

A *holiday_date* is the date of a public holiday. The scheduling program must not schedule work to be done on public holidays, nor must it schedule work to be done on Saturday or Sunday. If the *start_date* is a Saturday, Sunday or public holiday, the scheduling program must not schedule work to be done until the next day that is not a Saturday, Sunday or public holiday.

A *task_name* is an identifier. A *task_name* before a *task_description* labels the task so that it can be referred to in a *needs_clause*. A *task_name* in a *needs_clause* indicates that the scheduling program must not schedule the task associated with the *needs_clause* until after the task labeled by the *task_name* is completed.

A *task_description* is a string that describes the task to be completed.

A *person* token is an identifier that contains the initials of the person who will undertake the task. The scheduling program is not required to process *person* tokens more than three characters long. The scheduling program must schedule tasks assigned to the same person in the order in which they appear in the input file. In effect, each task other than

the first task assigned to a person has an implied *needs_clause* that requires the previous task assigned to the person to be completed before the new task is started.

duration is the number of days the person is to be given to complete the task.

If a person must wait for another person to complete a task, the scheduling program must imply a filler task for the first person. The filler task must start on the day that the person starts waiting and finish on the day before the person starts the next task. Each filler task must be listed in the project schedule listing along with the tasks defined in the input file. The task name of a filler task must be 'SPACE' and the task description must be 'Waiting for dependency'.

If the scheduling program detects an error in the input file, it should display a message indicating the input file line number at which the error was detected and the nature of the error. For example:

```
Error: line 142: invalid date
```

The project schedule listing generated by the scheduling program must contain the following information:

1. input file name;
2. for each task, including both defined tasks and filler tasks, in task completion date order:
    a. task name;
    b. task description;
    c. the initials of the person who is to undertake the task;
    d. task duration;
    e. task start date;
    f. task end date.
3. project duration.

The task start date is the first day on which work on the task will be done and the task end date is the last day on which work on the task will be done. I.e., the task is expected to start at the beginning-of-business on the task start date and finish before close-of-business on the task end date.

The project duration is the total number of working days the project will take to complete. It is not a simple sum of the task durations because tasks done by different people may be done in parallel.

The project schedule listing must be formatted in accordance with the following layout:

```
          1         2         3         4         5         6         7         8
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0....5....0
DD-MM-YY HH:MM             PROJECT SCHEDULE LISTING                      PAGE X
                           for X------------------X

Task name  Description                               Person Days  Start    Finish

X--------X X---------------------------------------X X-X   X-X  XX-XX-XX XX-XX-XX
X--------X X---------------------------------------X X-X   X-X  XX-XX-XX XX-XX-XX
X--------X X---------------------------------------X X-X   X-X  XX-XX-XX XX-XX-XX
                                                          ----
Project duration                                          X--X
```

The input file name must be displayed after the word ‘for’.

The column entitled ‘Person’ must contain the initials of the person who is to undertake the task.

The column entitled ‘Days’ must contain the task duration.

The column entitled ‘Start’ must contain the task start date.

The column entitled ‘Finish’ must contain the task end date.

3 FAILSAFE STORAGE

An application programmer needs a small amount of failsafe storage. The nature of the failsafe requirement is that the storage must be either completely updated by a write operation or completely unchanged. If a power failure occurs during a write operation, there must be a way to reliably roll back to the previous version of the data.

Your job is to write a Java or C++ class that achieves this objective, the ‘failsafe storage class’.

The failsafe storage class must maintain two copies of the stored data in separate disk files. Each time the data is updated, the class must write the new version to the alternate disk file, so that the previously updated disk file is not subject to corruption by the write operation. For example, if the two disk files are called ‘A’ and ‘B’, the first update might be written to file ‘A’. The second update must then be written to file ‘B’, the third back to file ‘A’ and so on.

Along with the data to be stored, the failsafe storage class must also store a 32-bit serial number and a 16-bit CRC of the serial number and the stored data.

The failsafe storage class must increment the 32-bit serial number each time it updates the stored data.

When the failsafe storage class retrieves the stored data, it must validate the checksums in both the disk files. If one of the checksums is valid, but not the other, the class must return the data with the valid checksum to the application.

If both the checksums are valid, the class must return the data with the more recent serial number. The class must determine the more recent serial number by executing the following algorithm:

1. Let the retrieved serial numbers be A and B.
2. If A = B, then there is a fault.
3. If $((A - B) \bmod 2^{32}) < 2^{31}$, then A is the more recent serial number, otherwise B is the more recent serial number.

If neither of the checksums is valid, the class must return a failed status to the application.

The method to calculate the CRC has already been written by another programmer. Its calling syntax in Java is:

```
class Checksums {
     // …
     static short crc16 (byte [] data, int length);
     // …
}
```

The calling syntax in C is:

```
unsigned short Crc16 (const void *data, int length);
```

`data`
: is the byte array for which the CRC is to be calculated.

`length`
: is the number of bytes over which the CRC is to be calculated.

`return value`
: is the 16-bit checksum.

If the CRC calculated by the above functions is appended to the data in little-endian byte order, the CRC of the concatenated data and CRC is zero.

If you write in Java, the failsafe storage class must be a concrete class that implements the following interface:

```
interface FailsafeInterface {
     byte [] read () throws IOException;
                          // Retrieve the data in the
                          // failsafe storage
     void write (byte [] data, int length)
          throws IOException;
                          // Update the contents of the
                          // failsafe storage
};
```

`read`
: must retrieve the latest data from the failsafe storage and return it as a byte array. The read method must throw an IOException if neither of the two disk files can be successfully read.

`write`
: must update the failsafe storage. The write method must throw an IOException if the disk file to be updated could not be written.

`data`
: is a reference to an array that contains the data to be stored in the failsafe storage.

`length`
: is the number of bytes to be stored in the failsafe storage.

If you write in C++ the failsafe storage class must have the following declaration:

```
class FailsafeStorage_c {
     // Class instance variables and local methods
     // as required.
public:
     int Read (void *buffer, int capacity);
                          // Retrieve the data in the
                          // failsafe storage
     int Write (const void *data, int length);
                          // Update the contents of the
                          // failsafe storage
};
```

`Read`
: must retrieve the latest data from the failsafe storage and load it into the buffer pointed to by the 'buffer' argument. If the data is successfully retrieved, Read must return the number of bytes actually read. If neither disk file could be successfully read, Read must return –1.

`buffer`
: is a pointer to a memory buffer to be loaded with the retrieved data.

`capacity`
: is the maximum number of bytes that Read may load into the buffer. If more than 'capacity' bytes are stored in the failsafe storage, Read must load the first 'capacity' bytes into the buffer and then return the number of bytes actually loaded.

`Write`
: must update the failsafe storage. If Write successfully updates the failsafe storage it must return the number of bytes written to the storage. If Write cannot update the failsafe storage, it must return –1.

`data`
: is a pointer to the data to be written to the failsafe storage.

`length`
: is the number of bytes to be written to the failsafe storage.

4 VENDING MACHINE MULTIPLEXER

To mitigate the effect of infectious diseases, a national health service has installed several hundred vending machines to supply condoms and clean syringes to drug addicts and other users.

Usage of the machines has been well below expectations. The health service has discovered that one of the reasons is because the users are having difficulty changing notes into coins. The shops around the vending machines are reluctant to change notes into coins because the users are a rather rough lot and the shopkeepers are worried that they will scare away paying customers.

To overcome this difficulty, the health service would like to install an electronic purse interface into each of the vending machines. However, the micro-controller of the vending machine has only one port for connecting to either a coin acceptor or an e-purse interface. The health service is reluctant to remove the coin acceptors because a significant number of users do not yet carry the e-purse cards.

Your task is to program a multiplexer that will allow both a coin acceptor and an e-purse interface to be used by the vending machine.

The existing coin acceptors communicate with the vending machine controller over a local TCP/IP network. Both the coin acceptor and the e-purse interface communicate with the vending machine controller using the same protocol.

The protocol is a client-server protocol in which the coin acceptor or e-purse interface is the client and the vending machine controller is the server. The underlying transport layer is TCP.

When a coin is successfully inserted into the coin acceptor, or an e-purse card comes within range of the e-purse reader/writer, the client sends a request message to the vending machine over the TCP connection. The request message is an 8-byte packet with the following structure.

Offset (bytes)	Length (bytes)	Contents
0	4	Serial number.
4	4	Value of the coin or total amount in the purse.

When the vending machine controller receives this message, it checks to see if the serial number is the same as the previous serial number received from the client. If the serial number is different to the previous serial number, the vending machine controller processes the request and returns a response back to the client. The response is an 8-byte packet with the following structure.

Offset (bytes)	Length (bytes)	Contents
0	4	Serial number.
4	4	Amount consumed.

The amount consumed is the amount actually spent by the user. The coin acceptor refunds the difference between the value of the coin and the amount consumed by dispensing lower-denomination coins back to the user. The e-purse interface deducts the amount consumed from the user's e-purse balance.

If the vending machine controller receives a request with a serial number equal to the serial number of the previous request, it assumes that the message is a repetition of the previous request and returns the last response again without reprocessing the request.

If the connection between a client and the vending machine controller is broken while the client is waiting for a response, the client attempts to re-establish a connection and resend the request. If the vending machine controller detects a broken connection, it abandons any response that is ready to be transmitted and listens for a new connection from the client.

All the numbers in the request and response messages are 32-bit binary quantities sent in 4-byte packets in big-endian order.

The multiplexer, that you are to program, must accept connections from both the coin acceptor and the e-purse interface, each of which has its own serial number sequence that may overlap or interlace with the serial number sequence of the other. The multiplexer must appear to both the coin acceptor and the e-purse interface to be a vending machine controller and it must appear to the vending machine controller to be a single coin acceptor or e-purse interface.

On receipt of a request from either the coin acceptor or the e-purse interface, the multiplexer must renumber the request so as to avoid incorrectly sending a duplicate serial number and then relay the request to the vending machine controller. When the vending machine controller returns a response, the multiplexer must relay the response back to the original client, converting the serial number back into that of the original request.

The multiplexer must respond to broken connections and repeated requests in the same way as the network components that it emulates.

For the purposes of initial development and testing, the network components are to communicate from the IP addresses and ports in the table below. A different IP addresses may be assigned to the vending machine controller at a later date. The multiplexer program should allow the IP addresses and port numbers to be conveniently changed by the use of symbolic constants.

Function	IP address	Port
The vending machine controller listens for connections from clients on:	127.0.0.1	2251
The multiplexer must listen for connections from clients on:	Not known	2252
The multiplexer must connect to the controller from:	Not known	2253
The coin acceptor connects to the multiplexer from:	Not known	2254
The e-purse interface connects to the multiplexer from:	Not known	2255

The multiplexer should be able to correctly process repeated requests even when the repetition is received after the multiplexer has been restarted as a consequence of a power failure.

The multiplexer may use the failsafe storage class described in the previous question.

PERTANDINGAN PENGATURCARAAN E-GENTING 2005

Arahan-arahan am:

1. Jawab satu atau lebih soalan yang diberikan.
2. Pertandingan ini adalah satu ujian dimana calon-calon boleh merujuk kepada buku-buku atau bahan-bahan rujukan.
3. Masa yang diperuntukkan untuk pertandingan ini adalah lapan jam.
4. Perbincangan di antara peserta-peserta adalah tidak dibenarkan.
5. Untuk menerima kredit bagi menjawab sesuatu soalan, jawapan anda mestilah merupakan penyelesaian yang munasabah. Penyelesaian yang munasabah ialah jawapan yang menyelesaikan masalah tersebut atau jawapan yang mungkin akan menyelesaikan masalah tersebut dengan sedikit usaha tambahan.
6. Bagi jawapan untuk penyelesaian yang munasabah, anda akan menerima kredit untuk hasilan methodikal seperti gambar rajah aliran data, gambar rajah peralihan keadaan, jadual dan sebagainya.
7. Jumlah markah anda adalah daripada jumlah kredit yang diperuntukkan kepada setiap soalan yang dijawab.
8. Program-program anda akan dinilai berdasarkan kepada betapa mudah ianya boleh dibaca dan difahami.
   - Takukan mestilah bersih and sejajar.
   - Nama-nama pembolehubah harus menggambarkan isi kandungan pembolehubah-pembolehubah berkenaan.
   - Pasangan (*coupling*) di antara modul-modul mestilah nyata.
   - Setiap modul harus melakukan satu perkara dengan baik
9. Markah yang diperuntukkan kepada setiap soalan adalah seperti berikut:

No	Nama	Markah
1.	Mobil Mytel	250
2.	Aturcara Penjadualan Projek	350
3.	Storan Selamat	100
4.	Pemultipleks Mesin Penjualan	500

10. Selain dinyatakan, program anda boleh ditulis menggunakan mana-mana bahasa pengaturcaraan utama di bawah mana-mana sistem operasi utama.
11. Selain dinyatakan, anda boleh menggunakan semua fungsi piawaian perpustakaan dalam bahasa dan sistem operasi yang anda pilih.
12. Perkataan-perkataan ‘must’, ‘must not’, ‘required’, ‘should’, ‘should not’, dan ‘may’ adalah ditafsirkan seperti yang diterangkan dalam RFC 2119[2].
13. . Anda TIDAK dijangka menjawab semua soalan

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

1 MOBIL MYTEL

Mytel Mobile adalah pembekal perkhidmatan telekomunikasi untuk telefon mudah alih yang beroperasi di beberapa buah negara di Asia Tenggara. Ia mempunyai lebih kurang dua juta pelanggan, di mana kebanyakan daripada mereka sering berkunjung ke luar negara.

Mytel telah mengendalikan "Program Pelanggan Setia" selama lima tahun, di mana di bawah program ini, pelanggan boleh mendapat mata ganjaran apabila membuat panggilan telefon melalui perkhidmatan telekomunikasi Mytel. Pelanggan boleh menggunakan mata ganjaran tersebut untuk membeli pelbagai barangan dan perkhidmatan.

Mytel berusaha untuk memeterai persetujuan dengan syarikat-syarikat lain supaya dibenarkan untuk membeli barangan dan perkhidmatan dengan mata ganjaran dan juga pada harga diskaun. Mytel berpendapat bahawa dengan pemberian diskaun ianya dapat membantu dalam mempromosikan produk syarikat-syarikat tersebut dan sekaligus terus melanggan pada masa akan datang.

Mytel berhajat untuk memeterai persetujuan sedemikian dengan salah sebuah syarikat penerbangan yang merupakan syarikat penerbangan utama di rantau ini yang diliputi perkhidmatan Mytel.

Mytel kini perlu meyakinkan syarikat penerbangan tersebut bahawa sebahagian besar daripada pelanggannya adalah mereka yang kerap melancong, dan mereka mempunyai mata ganjaran yang cukup untuk membeli bilangan tiket yang ketara.

Tugas anda adalah menulis satu aturcara yang menjanakan laporan. Aturcara tersebut akan menganalisa pangkalan data Mytel yang menyimpan rekod panggilan dan menunjukkan statistik yang akan menarik minat syarikat penerbangan tersebut.

Aturcara tersebut mesti menerima satu tempoh laporan yang diungkapkan sebagai satu "tarikh-dari" dan satu "tarikh-sehingga". Laporan yang dijanakan harus mengandungi:

5. Tarikh dan masa laporan itu dijanakan.
6. Tempoh laporan ("tarikh-dari" dan "tarikh-sehingga")
7. Bagi setiap julat baki imbangan mata ganjaran:
    a. Julat baki imbangan mata ganjaran;
    b. Jumlah pelanggan dalam pangkalan data dengan baki imbangan mata ganjaran yang berada dalam julat, dan telah menbuat satu atau lebih panggilan dalam tempoh laporan ("Pelanggan terpilih").
    c. Bagi setiap bandar utama berada dalam kawasan liputan Mytel ("bandar pertama"):
        i. bilangan pelanggan terpilih yang menetap di bandar pertama
        ii. Bagi setiap bandar utama lain yang berada dalam kawasasn liputan Mytel ("bandar lain"):
            1. Kekerapan pelanggan terpilih yang menetap di bandar pertama berkunjung ke bandar lain.
    d. Jumlah kunjungan yang dibuat oleh pelanggan terpilih.
8. Jumlah bagi semua julat baki imbangan mata ganjaran:
    a. Jumlah bilangan pelanggan yang membuat panggilan dalam tempoh laporan ("pelanggan aktif").

b. Bagi setiap bandar utama yang berada dalam kawasan liputan Mytel ("bandar pertama");
    i. Bilangan pelanggan aktif yang menetap di bandar pertama;
    ii. Bagi setiap bandar utama yang berada dalam kawasan liputan Mytel ("bandar lain"):
        1. Kekerapan kunjungan ke bandar lain yang dilakukan oleh pelanggan aktif yang menetap di bandar pertama.
c. Jumlah bilangan kunjungan yang dibuat oleh semua pelanggan aktif.

Aturcara tersebut mesti menerima "tarikh-dari" dan "tarikh-sehingga" dalam format DD-MM-YY (di mana DD adalah hari, MM adalah bulan dan YY adalah tahun). Tahun yang berada dalam julat antara 70 sehingga 99 mesti dianggap berada dalam abad ke-20 dan tahun yang berada dalam julat 00 sehingga 69 mesti dianggap berada dalam abad ke-21.

Aturcara tersebut harus dimulakan melalui "command line" seperti di bawah:

```
MytelRep 01-01-05 30-06-05
```

Di mana '01-01-05' adalah "tarikh-dari" dan '30-06-05' adalah "tarikh-sehingga".

Tempoh laporan mesti mengandungi "tarikh-dari" dan "tarikh-sehingga". Sebagai contoh, tempoh laporan untuk "command line" di atas adalah pukul 00:00 pada 1hb Januari 2005, sehingga dan termasuk pukul 23:59 pada 30hb Jun 2005.

Julat baki imbangan poin adalah 0 hingga 99,999 poin, 100,000 hingga 299,999 poin dan 300,000 poin atau lebih.

Bandar-bandar utama adalah Kuala Lumpur, Singapura dan Bangkok. Bandar-bandar tersebut dikenal pasti dengan kod lapangan terbang: KUL, SIN, BKK masing-masing.

Laporan tersebut mesti diformat mengikut susun atur seperti berikut:

```
         1         2         3         4         5         6         7         8
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0....5....0
DD-MM-YY HH:MM          INTER-CITY TRIP ANALYSIS REPORT                   PAGE X
                           for XX-XX-XX to XX-XX-XX

                     Number of   Base          Number of trips to         Total no
Point range        subscribers   city       KUL         SIN         BKK   of trips

0 to 99,999          X,XXX,XXX   KUL          0  X,XXX,XXX  X,XXX,XXX  X,XXX,XXX
                                 SIN  X,XXX,XXX          0  X,XXX,XXX
                                 BKK  X,XXX,XXX  X,XXX,XXX          0

100,000 to 299,999   X,XXX,XXX   KUL          0  X,XXX,XXX  X,XXX,XXX  X,XXX,XXX
                                 SIN  X,XXX,XXX          0  X,XXX,XXX
                                 BKK  X,XXX,XXX  X,XXX,XXX          0

300,000 or more      X,XXX,XXX   KUL          0  X,XXX,XXX  X,XXX,XXX  X,XXX,XXX
                                 SIN  X,XXX,XXX          0  X,XXX,XXX
                                 BKK  X,XXX,XXX  X,XXX,XXX          0

Total                X,XXX,XXX   KUL          0  X,XXX,XXX  X,XXX,XXX  X,XXX,XXX
                                 SIN  X,XXX,XXX          0  X,XXX,XXX
                                 BKK  X,XXX,XXX  X,XXX,XXX          0
```

Aturcara tersebut mesti mengumpul maklumat yang berada dalam laporan daripada pendaftar panggilan Mytel.

Pendaftar panggilan tersebut ialah sebuah pangkalan data SQL dengan skema seperti berikut:

```
create table subscribers (
     subId          integer not null,
     subName        char(40) not null,
     subBal         integer not null
);
create unique index subIdInd on
     subscribers (subId);

create table callRegister (
     crSubId        integer not null,
     crFromCity     char(3) not null,
     crCalledNo     char(20) not null,
     crConnectTime  integer not null,
     crDisconTime   integer not null
);
create index callRegTimeInd on
     callRegister (crConnectTime);
```

`subscribers`
: ialah sebuah jadual yang mengandungi satu pelanggan untuk setiap baris.

`subId`
: ialah pengecam pelanggan yang mengenal pasti setiap pelanggan secara unik.

`subName`
: ialah nama pelanggan.

`subBal`
: ialah baki imbangan mata ganjaran pelanggan.

`callRegister`
: ialah sebuah jadual yang mengandungi satu panggilan dibuat oleh pelanggan untuk setiap baris.

`crSubId`
: ialah pengecam pelanggan kepada pelanggan yang membuat panggilan.

`crFromCity`
: ialah kod lapangan terbang bandar di mana panggilan itu dibuat.

`crCalledNo`
: ialah nombor telefon yang dipanggil.

`crConnectTime`
: ialah masa ketika sambungan telefon dibuat dalam saat sejak 00:00 UTC pada 1hb Januari 1970.

`crDisconTime`
: ialah masa ketika sambungan telefon diputuskan dalam saat sejak 00:00 UTC pada 1hb Januari 1970.

Pendaftar panggilan adalah satu jadual yang besar. Ia mengandungi lebih daripada 45 bilion baris! Ia adalah terlalu besar untuk disusun oleh system pangkalan data. Ia hanya boleh dibaca mengikut tertib masa sambungan.

Sebarang cubaan untuk memilih satu baris secara rawak dari pendaftar panggilan mengambil masa pelaksanaan yang panjang. Sebagai contoh, kiraan yang dibuat oleh jurutera perisian di Mytel menunjukkan bahawa beberapa tahun diperlukan untuk melaksanakan pernyataan SQL berikut untuk setiap pelanggan dalam pangkalan data.

```
select    crConnectTime, crFromCity
from      callRegister
where     crSubId = :subId and
          crConnectTime between :fromDate and :toDate
order     by crConnectTime;
```

Secara praktik, pendaftar panggilan hanya boleh diimbas secara berturut dalam tertib masa sambungan.

Walaupun demikian, komputer yang akan melaksanakan aturcara tersebut adalah agak besar. Aturcara tersebut boleh menggunakan RAM sehingga 150 Mb.

Aturcara tersebut mesti menentukan bandar penetapan setiap pelanggan dengan anggapan bahawa bandar penetapan pelanggan adalah bandar di mana paling banyak panggilan telah dibuat dari bandar tersebut oleh pelanggan tersebut.

Aturcara tersebut mesti menentukan bilangan kunjungan yang dibuat oleh setiap pelanggan dengan mengimbas pendaftar panggilan dan mengira kunjungan baru apabila terdapat panggilan dibuat dari bandar yang bukan merupakan bandar penetapan pelanggan. Sebagai contoh, jika KUL adalah bandar penetapan pelanggan dan panggilan telah dibuat dari KUL, SIN, KUL, BKK, dan kemudian KUL, aturcara tersebut mesti mengira dua kunjungan, satu ke Singapura dan satu lagi ke Bangkok.

Jika seorang pelanggan membuat kunjungan dari satu bandar yang bukan merupakan bandar penetapan ke satu lagi bandar yang juga bukan bandar penetapan, pusingan itu mesti dikira sebagai satu kunjungan. Sebagai contoh, jika seorang pelanggan yang menetap di Singapura membuat panggilan dari SIN, KUL, BKK, KUL kemudian SIN, aturcara tersebut mesti mengira 3 kunjungan, satu ke Kuala Lumpur, satu ke Bangkok dan kemudian kembali ke Kuala Lumpur. Pusingan balik ke Singapura tidak dikira sebagai satu kunjungan.

Walaupun lokasi utama Mytel beroperasi adalah di Kuala Lumpur, Singapura dan Bangkok, pelanggan kadang-kala membuat panggilan dari bandar yang lain. Jika aturcara tersebut menemui kod bandar yang selain daripada ‘KUL’, ‘SIN’ atau ‘BKK’ dalam pangkalan data, ia harus mengabaikan baris pendaftar panggilan tersebut yang mengandungi kod bandar yang tidak dikenali.

2 ATURCARA PENJADUALAN PROJEK

Syarikat anda memerlukan satu aturcara untuk menjadualkan fasa-fasa projek yang melibatkan beberapa orang pengaturcara. Tugas anda adalah menulis aturcara penjadualan tersebut.

Aturcara tersebut mesti menerima satu nama fail input sebagai “argument”, membaca dan menganalisa fail input tersebut dan menjanakan senarai jadual projek.

Kandungan fail input yang tipikal adalah seperti berikut:

```
start 30-08-05;          # Project start date
holidays {               # Public holidays
     31-08-05,                # Hari merdeka
     01-11-05                 # Deepavali
}
tasks {                  # Task list
     func      "Functional specification"    MDJ  10;
     design    "Internal design"             AB   12
          needs func;
     screens   "Program data entry screens"  JOJ  8
          needs design;
     reports   "Program reports"             AB   10
          needs design;
     tests     "Acceptance tests"            MDJ  12
          needs screens, reports;
}
```

Teks yang bermula dengan karakter hash (#) dan bersambungan hingga ke penghujung baris tersebut adalah komen. Aturcara tersebut mesti mengabaikan komen dalam fail input.

Selain komen, kandungan fail input adalah dibahagikan kepada token.

Terdapat lima jenis token. Jenis-jenis token adalah kata kunci seperti '`start`' dan '`holidays`', tarikh seperti '`30-08-05`', pengecam seperti '`screens`', rentetan seperti "`Program reports`" dan nombor.

Sifar atau lebih ruang karakter (iaitu jarak, tab, dan new-line) boleh digunakan untuk mengasingkan token-token yang berturutan. Jika dua token berturutan tidak dapat dibezakan jika tiada ruang di antara mereka (contoh: 'needs' diikuti dengan 'design' dalam contoh di atas), fail input tersebut dijamin mengandungi sekurang-kurangnya satu karakter ruang di antara dua token itu.

Tarikh adalah dalam format DD-MM-YY.

Pengecam hanya mengandungi huruf besar, huruf kecil, digit, karakter 'underscore' dan 'dash'. Aturcara tersebut tidak perlu memproses pengecam yang lebih panjang daripada 10 karakter.

Pengecam, nombor dan tarikh tidak dapat dibezakan pada aras lexical. Mereka tidak dapat dibezakan antara satu sama lain sehingga penghurai mengenal kedudukan mereka dalam susunan sintaks.

Rentetan dirangkum dengan double-quotes. Dalam rentetan, simbol '`\"`' dan '`\\`' mewakili "double-quotes" dan "back-slash" masing-masing. Aturcara tersebut tidak perlu memproses rentetan yang lebih panjang daripada 40 karakter. Rentetan boleh tidak mengandungi karakter "new-line".

Format fail input dalam BNF ialah:

input_file:
    start_stmt holidays_stmt tasks_stmt

start_stmt:
    `start` *start_date* `;`

holidays_stmt:
    `holidays { `*holiday_list*$_{opt}$` }`

holiday_list:
    holiday_date
    holiday list `,` *holiday_date*

tasks_stmt:
    `tasks { `*task_list*$_{opt}$` }`

task_list:
    task_definition
    task_list task_definition

task_definition:
    task_name task_description person duration needs_clause$_{opt}$ `;`

needs_clause:
    `needs` *needs_list*

needs_list:
    task_name
    needs_list `,` *task_name*

start_date adalah tarikh hari pertama di mana kerja perlu dilakukan, dengan syarat hari tersebut bukan hari cuti.

holiday_date adalah tarikh cuti am. Aturcara tersebut mestilah tidak menjadualkan tugas pada cuti am, dan juga Hari Sabtu dan Hari Ahad. Jika *start_date* adalah Hari Sabtu, Hari Ahad atau cuti am, aturcara tersebut mestilah tidak menjadualkan tugas sehingga hari seterusnya yang bukan merupakan Hari Sabtu, Hari Ahad atau cuti am.

task_name adalah pengecam. *task_name* sebelum *task_description* melabelkan tugas tersebut supaya ia boleh dirujuk di dalam *needs_clause. task_name* dalam *needs_clause* menunjukkan bahawa aturcara tersebut mestilah tidak menjadualkan tugas berkait dengan *needs_clause* sehingga selepas tugas yang dilabel dengan *task_name* diselesaikan.

task_description adalah satu rentetan yang menerangkan tugas yang perlu di selesaikan.

Token *person* ialah satu pengecam yang mengandungi parap individu yang akan menjalankan tugas tersebut. Aturcara tersebut tidak perlu memproses token *person yang* mempunyai panjang lebih daripada tiga karakter. Aturcara tersebut mesti menjadualkan tugas yang diberikan kepada individu yang sama dengan tertib yang sama seperti mana ia wujud dalam fail input. Kesannya, setiap tugas selain tugas pertama yang diberi kepada

seseorang individu mempunyai *needs_clause* tersirat yang memerlukan tugas sebelumnya diselesaikan sebelum memulakan tugas baru.

duration ialah bilangan hari yang diberikan kepada seseorang individu untuk melengkapkan tugas yang diberi.

Jika seseorang individu perlu menunggu seseorang individu yang lain melengkapkan tugas, aturcara tersebut mesti memberikan tugas pengisi kepada individu pertama tersebut. Tugas pengisi tersebut mesti bermula pada hari seseorang individu itu mula menunggu dan tamat pada hari sebelum individu tersebut memulakan tugas seterusnya. Setiap tugas pengisi perlu disenaraikan dalam senarai jadual projek bersama dengan tugas yang didefinasi dalam fail input. Nama tugas untuk satu tugas pengisi mestilah 'SPACE' dan penerangan tugas mestilah 'Waiting for dependency'.

Jika aturcara tersebut mengesan sesuatu ralat dalam fail input, ia harus memaparkan mesej untuk menunjukkan nombor baris dalam fail input di mana ralat itu dikesan dan jenis ralat tersebut. Contoh:

```
Error: line 142: invalid date
```

Senarai jadual projek yang dijanakan oleh aturcara tersebut mesti mengandungi maklumat berikut:

1. Nama fail input;
2. bagi setiap tugas, termasuk tugas yang didefinasi dan tugas pengisi, disusun dalam tertib tarikh siap:
    a. nama tugas;
    b. penerangan tugas;
    c. parap individu yang akan menjalankan tugas tersebut;
    d. jangka masa tugas
    e. tarikh mula tugas
    f. tarikh tamat tugas
3. jangka masa projek.

Tarikh mula tugas adalah hari pertama di mana kerja akan dilakukan ke atas tugas tersebut dan tarikh tamat tugas adalah hari terakhir di mana kerja akan dilakukan atas tugas tersebut. Dengan kata lain, tugas tersebut dijangka bermula ketika 'permulaan perniagaan' pada tarikh mula tugas dan tamat sebelum 'penutupan perniagaan' pada hari tamat tugas.

Jangka masa projek adalah jumlah bilangan hari bekerja yang diambil untuk menyiapkan projek tersebut. Ia bukanlah jumlah jangka masa semua tugas sebab tugas yang disiapkan oleh individu yang berlainan boleh dilakukan secara selari.

Senarai jadual projek mestilah diformat dalam susun atur seperti berikut:

```
         1         2         3         4         5         6         7         8
1...5....0....5....0....5....0....5....0....5....0....5....0....5....0....5....0
DD-MM-YY HH:MM             PROJECT SCHEDULE LISTING                      PAGE X
                           for X-----------------X

Task name  Description                              Person Days  Start    Finish

X--------X X--------------------------------------X X-X   X-X  XX-XX-XX XX-XX-XX
X--------X X--------------------------------------X X-X   X-X  XX-XX-XX XX-XX-XX
X--------X X--------------------------------------X X-X   X-X  XX-XX-XX XX-XX-XX
                                                          ----
Project duration                                          X--X
```

Nama fail input mestilah dipaparkan selepas perkataan ‘for’.

Lajur yang bertajuk ‘Person’ mesti mengandungi parap individu yang akan menjalankan tugas tersebut.

Lajur yang bertajuk ‘Days’ mesti mengandungi jangka masa tugas.

Lajur yang bertajuk ‘Start’ mesti mengandungi tarikh mula tugas.

Lajur yang bertajuk ‘Finish’ mesti mengandungi tarikh tamat tugas.

3 STORAN SELAMAT

Seorang pengaturcara aplikasi memerlukan sebilangan kecil storan yang selamat. Sifat storan tersebut ialah ia mesti dikemaskini oleh satu operasi tulis secara lengkap sama sekali atau tidak berubah sama sekali. Sekiranya gangguan elektrik berlaku ketika sesuatu operasi tulis berlangsung, satu cara yang boleh dipercayai mesti wujud untuk mengembalikan data asal.

Tugas anda ialah menulis satu kelas Java atau C++ untuk mencapai objektif kelas storan selamat ini.

Kelas storan selamat tersebut mesti mengekalkan dua salinan data yang tersimpan dalam fail disk yang berlainan. Setiap kali data dikemaskini, kelas tersebut mesti menulis versi baru data itu ke fail disk alternate, supaya fail disk yang dikemaskini sebelumnya tidak akan dipengaruhi oleh korupsi semasa operasi tulis tersebut. Sebagai contoh, jika dua fail disk tersebut dipanggil ‘A’ dan ‘B’, kemaskini pertama akan ditulis ke fail ‘A’. Kemaskini kedua mesti ditulis ke fail ‘B’, kemaskini ketiga ke file ‘A’ dan seterusnya.

Bersama dengan data yang akan disimpan, kelas storan selamat tersebut mesti juga menyimpan satu nombor siri 32-bit dan satu CRC 16-bit daripada nombor siri beserta data tersimpan.

Kelas storan selamat tersebut mesti menambah nombor siri 32-bit tersebut setiap kali mengemaskinikan data tersimpan.

Apabila kelas storan selamat tersebut mengesan semula data tersimpan, ia mesti mengesahkan ‘hasil tambah semak’ (checksum) dalam kedua-dua fail disk. Jika salah

satu hasil tambah semak adalah sah, tetapi tidak bagi yang satu lagi, kelas tersebut mesti mengembalikan data dengan hasil tambah semak yang sah kepada aplikasi.

Jika kedua-dua hasil tambah semak adalah sah, kelas tersebut mesti mengembalikan data dengan nombor siri yang terkini. Kelas tersebut mesti menentukan nombor siri yang lebih kini dengan melaksanakan algoritma berikut:

1. Katakan nombor-nombor siri yang tercapai ialah A dan B masing-masing.
2. Jika A = B, maka kesilapan berlaku.
3. Jika $((A - B) \bmod 2^{32}) < 2^{31}$, maka A adalah nombor siri yang terkini, jika tidak, B adalah nombor siri yang lebih terkini.

Jika tiada sebarang hasil tambah semak adalah sah, kelas tersebut mesti mengembalikan status gagal kepada aplikasi tersebut.

Metod untuk mengira CRC telah ditulis oleh seorang pengaturcara lain. Sintaks panggilan dalam Java ialah:

```
class Checksums {
     // …
     static short crc16 (byte [] data, int length);
     // …
}
```

Sintaks panggilan dalam C ialah:

```
unsigned short Crc16 (const void *data, int length);
```

`data`
: ialah tatasusunan bait di mana CRC akan dikira.

`length`
: ialah bilangan bait di mana CRC akan dikira.

`return value`
: ialah hasil tambah semak 16-bit

Jika CRC yang dikira dari fungsi di atas disambung kepada data dalam susunan byte "little endian", CRC untuk data yang bersambungan dengan CRC ialah sifar.

Jika anda menulis dalam Java, kelas storan selamat mestilah merupakan kelas kukuh yang mempunyai antaramuka berikut:

```
interface FailsafeInterface {
     byte [] read () throws IOException;
                         // Retrieve the data in the
                         // failsafe storage
     void write (byte [] data, int length)
          throws IOException;
                         // Update the contents of the
                         // failsafe storage
};
```

`read`
: mesti mencapai data terkini daripada storan selamat dan mengembalikannya sebagai satu tatasusunan bait. Metod read mesti membuang IOException jika kedua-dua fail disk tidak dapat dibaca.

`write`
: mesti kemaskini storan selamat. Metod write mesti membuang IOException jika fail disk yang akan dikemaskini tersebut tidak dapat ditulis.

`data`
: ialah satu rujukan kepada tatasusunan yang mengandungi data yang akan disimpan dalam storan selamat.

`length`
: ialah bilangan bait yang akan disimpan dalam storan selamat.

Jika anda menulis dalam C++, kelas storan selamat tersebut mesti mempunyai pengisytiharan berikut:

```
class FailsafeStorage_c {
     // Class instance variables and local methods
     // as required.
public:
     int Read (void *buffer, int capacity);
                         // Retrieve the data in the
                         // failsafe storage
     int Write (const void *data, int length);
                         // Update the contents of the
                         // failsafe storage
};
```

`Read`
: mesti mencapai data terkini dari storan selamat dan muatkan data tersebut ke dalam ‘buffer’ yang dituding oleh argumen ‘buffer’. Jika data berjaya dicapai, Read mesti mengembalikan bilangan bait yang sebenarnya dibaca. Jika kedua-dua fail disk tidak dapat dibaca, Read mesti mengembalikan –1.

`buffer`
: ialah satu penuding kepada penimbal memori yang akan dimuatkan dengan data yang dicapai.

`capacity`
: ialah bilangan bait maksimum yang Read akan muatkan ke dalam ‘buffer’. Jika bilangan bait yang tersimpan dalam storan selamat melebihi ‘capacity’, Read mesti memuatkan ‘capacity’ bytes pertama ke dalam ‘buffer’, kemudian mengembalikan bilangan bait yang sebenarnya dimuatkan.

`Write`
: mesti mengemaskini storan selamat tersebut. Jika Write berjaya mengemaskini storan selamat tersebut, ia mesti mengembalikan bilangan bait yang telah ditulis kepada storan. Jika Write tidak dapat mengemaskini storan selamat tersebut, ia mesti mengembalikan –1.

`data`
: ialah satu penuding kepada data yang akan ditulis ke storan selamat.

`length`
: ialah bilangan bait yang akan ditulis ke storan selamat.

4 PEMULTIPLEKS MESIN PENJUALAN

Untuk mengurangkan kesan penyakit berjangkit, sebuah syarikat perkhidmatan kesihatan kebangsaan telah memasang beratus mesin penjualan untuk membekalkan kondom dan picagari bersih kepada penagih dadah dan pengguna yang lain.

Walaubagaimanapun kadar penggunaan mesin-mesin itu kurang memuaskan. Syarikat perkhidmatan kesihatan tersebut mendapati salah satu puncanya adalah kerana pengguna menghadapi kesukaran dalam menukar wang kertas ke wang syiling. Kedai-kedai di sekitar mesin-mesin penjualan enggan menawarkan penukaran wang kertas ke wang syiling kepada pengguna mesin tersebut kerana mereka bersikap agak kasar. Peniaga bimbang mereka akan menakutkan pelanggan-pelanggan kedai.

Untuk mengatasi masalah ini, syarikat tersebut ingin memasang perantara “dompet elektronik” pada mesin-mesin penjualan. Tetapi, pengawal-mikro pada mesin hanya mempunyai satu port untuk menyambung kepada sama ada satu penerima syiling atau satu antaramuka e-dompet.

Syarikat tersebut enggan menanggalkan penerima syiling kerana sebilangan besar penggunanya belum lagi mempunyai kad e-dompet.

Tugas anda adalah untuk memprogramkan sebuah pemultipleks yang akan membolehkan kedua-dua penerima syiling dan antaramuka e-dompet digunakan pada mesin penjualan.

Penerima syiling yang digunakan ketika ini berkomunikasi dengan pengawal mesin penjualan melalui satu rangkaian lokal TCP/IP. Kedua-dua penerima syiling dan antaramuka e-dompet berkomunikasi dengan pengawal mesin penjualan dengan menggunakan protokol yang sama.

Protokol tersebut adalah protokol klien-pelayan di mana penerima syiling atau antaramuka e-dompet adalah klien dan pengawal mesin penjualan adalah pelayan. Lapisan dasar untuk pengangkutan adalah TCP.

Apabila satu syiling berjaya dimasukkan ke dalam penerima syiling, atau sesekeping kad e-dompet mendekati alat pembaca/penulis e-dompet, klien menghantar satu mesej permintaan kepada mesin penjualan melalui sambungan TCP. Mesej permintaan adalah satu paket 8-bait yang mempunyai struktur seperti berikut:

Offset (bait)	Panjang (bait)	Kandungan
0	4	Nombor Siri.
4	4	Nilai syiling atau jumlah amaun dalam dompet.

Apabila mesin penjualan tersebut menerima mesej ini, ia akan memeriksa sama ada nombor siri adalah sama dengan nombor siri yang diterima dari klien sebelum itu. Jika nombor siri adalah berbeza dengan nombor siri sebelum itu, pengawal mesin penjualan memproses permintaan tersebut dan mengembalikan satu respon kepada klien. Respon tersebut ialah satu paket 8 bait dengan struktur seperti berikut:

Offset (bait)	Panjang (bait)	Kandungan
0	4	Nombor Siri.
4	4	Amaun yang telah digunakan.

Amaun yang telah digunakan ialah amoun yang sebenarnya dihabiskan oleh pengguna. Penerima syiling membayar balik perbezaan antara nilai syiling dengan amaun yang dihabiskan dengan mendispenkan syiling yang mempunyai denominasi yang lebih rendah kepada pengguna. Antaramuka e-dompet menolak amaun yang di habiskan daripada baki e-dompet pengguna.

Jika pengawal mesin penjualan menerima satu permintaan dengan nombor siri yang sama dengan nombor siri untuk permintaan sebelum itu, ia akan menganggap mesej itu adalah pengulangan permintaan sebelum itu dan mengembalikan respon terakhir sekali lagi tanpa memproses semula permintaan itu.

Jika sambungan antara klien dengan pengawal mesin penjualan terputus ketika klien sedang menunggu respon, klien akan cuba menjalin balik sambungan dan menghantar semula permintaan itu. Jika pengawal mesin penjualan mengesan suatu sambungan yang terputus, ia akan menggugurkan sebarang respon yang tersedia untuk dihantar dan menunggu sambungan baru dari klien.

Semua nombor dalam mesej permintaan dan respon adalah kuantiti dalam binari 32-bit dan dihantar dalam paket 4-bait dengan susunan big-endian.

Pemultipleks yang anda akan program, mesti menerima sambungan dari kedua-dua penerima syiling dan antaramuka e-dompet, di mana setiap satu mempunyai turutan nombor siri masing-masing, yang akan bertindih atau berjalin dengan turutan nombor siri yang lain. Pemultipleks mesti dilihat sebagai pengawal mesin penjualan oleh kedua-dua

penerima syiling dan antaramuka e-dompet. Di samping itu, pemultipleks juga mesti dilihat sebagai satu penerima syiling atau antaramuka e-dompet oleh pengawal mesin penjualan.

Ketika penerimaan satu permintaan dari sama ada penerima syiling atau anataramuka e-dompet, pemultipleks tersebut mesti menomborkan semula permintaan tersebut untuk mengelakkan kesilapan penghantaran nombor siri yang berulang yang akan kemudiannya menyampaikan permintaan tersebut kepada pengawal mesin penjualan. Apabila pengawal mesin penjualan mengembalikan satu respon, pemultipleks tersebut mesti menyampaikan respon itu kembali ke klien asal, dengan menukar balik nombor siri kepada nombor siri asal permintaan.

Pemultipleks tersebut mesti bertindak terhadap sambungan yang putus dan permintaan yang berulang dalam cara yang sama seperti komponen-komponen dalam rangkaian.

Untuk tujuan pembangunan dan pengujian permulaan, komponen-komponen rangkaian akan berkomunikasi dari alamat IP dan port-port seperti dalam jadual di bawah. Alamat-alamat IP yang berbeza akan diumpukkan kepada pengawal mesin penjualan pada masa yang akan dating. Program pemultipleks harus membenarkan alamat-alamat IP dan nombor-nombor port mudah diubah dengan penggunaan pemalar simbolik.

Fungsi	Alamat IP	Port
Pengawal mesin penjualan menunggu sambungan dari klien pada:	127.0.0.1	2251
Pemultipleks mesti menunggu sambungan dari klien pada:	Not known	2252
Pemultipleks mesti menyambung kepada pengawal dari:	Not known	2253
Penerima syiling menyambung ke pemultipleks dari:	Not known	2254
Antaramuka e-dompet menyambung ke pemultipleks dari:	Not known	2255

Pemultipleks harus berupaya untuk memproses permintaan yang berulang secara betul walaupun pengulangan itu diterima selepas pemultipleks dimulakan semula akibat gangguan elektrik.

Pemultipleks boleh menggunakan kelas storan selamat yang diterangkan pada soalan sebelum ini.