E-GENTING
PROGRAMMING COMPETITION 2008

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.	Table Load Profile	280
2.	Runner Status Display	330
3.	Denomination/Quantity Data Entry Block	220
4.	Data Stream Decoder	40

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 TABLE LOAD PROFILE

The management of the Ocean Treasure Restaurant have noticed that on occasion they have had to turn away large groups of guests because smaller groups occupy all their large tables. They suspect that their table mix might not be right and perhaps they should have a larger number of medium sized tables and less large tables.

To decide whether or not this would be a good idea, they would like to analyse their database to obtain an estimate of the number of groups of different sizes that occupy the restaurant on an hour-by-hour basis. Your task is to write a computer program that performs this analysis and produces a report, the 'Table Load Profile' that sumarises the results of the analysis.

The Ocean Treasure order entry and accounting system maintains several database tables that might assist the analysis.

Each time an order is taken from a table, the order is recorded in an orders table. The structure of the orders table is as follows:

```
create table orders (
     orderNo        integer not null,
     orderDate      date not null,
     orderTime      timestamp not null,
     tableNo        char(4) not null,
     covers         integer not null,
     waiterCode     char(8) not null
);
```

`orders`
: is a table that contains one row for each new order.

`orderNo`
: is a number that uniquely identifies the order.

`orderDate`
: is the trading day date of the order.

`orderTime`
: is the time when the order was taken.

`tableNo`
: is a code that uniquely identifies the table from which the order was taken.

`covers`
: for a new group of guests, is the number of guests seated at the table; for the second and subsequent orders from the same group of guests, it is zero.

`waiterCode`
: is a code that uniquely identifies the waiter that took the order.

When a group of guests settles their account, the settlement is recorded in a settlements table. The structure of the settlements table is as follows:

```
create table settlements (
     tableNo        char(4) not null,
     settleDate     date not null,
     settleTime     timestamp not null,
     settleMode     smallint not null,
     billAmount     double precision not null,
     tipAmount      double precision not null,
     refCode        char(32)
);
```

`settlements`
: is a table that contains one row for each settlement event.

`tableNo`
: is a code in the same domain as orders.tableNo that uniquely identifies the table where the guests sat.

`settleDate`
: is the trading day date of the settlement.

`settleTime`
: is the time at which the account was settled.

`settleMode`
: is a code that identifies the way in which the account was settled. 0 means cash, 1 means Visa, 2 means Mastercard and so on.

`billAmount`
: is the total amount of the bill that was settled.

`tipAmount`
: is the amount of any tip that the patrons gave.

`refCode`
: is a reference code, which has a different meaning depending on the settlement mode. If the settlement mode involved a credit card, it is usually the reference number printed on the credit card slip.

The Ocean Treasure Restaurant ordinarily opens around 11am, but may not close until 2 or 3am on the following morning, depending on customer demand. A single open session between 11am and 2am the next morning is referred to as a ‘trading day’. The Ocean Treasure Restaurant accounts for its trading activities in trading days and not real-time days.

The managers of the Ocean Treasure Restaurant would like to be able to generate a report similar to the example in Figure 1.

```
01-09-08 19:49      TABLE LOAD PROFILE              PAGE   1
                  for MON, TUE, WED, THU
              between 01-01-08 and 30-06-08

Hourly                Peak Concurrently Open Tables
Segment             1-2       3-4       5-6    7-or-more
                  Avg Max   Avg Max   Avg Max   Avg Max
11:00 to 11:59      1   3     0   2     0   3     0   1
12:00 to 12:59      4  10     1   5     2   7     0   3
13:00 to 13:59     12  22     5  12     5  10     0   3
14:00 to 14:59     13  23     5  13     5  12     1   4
15:00 to 15:59      9  16     3   8     4  11     0   4
16:00 to 16:59      4  10     1   4     1   5     0   3
17:00 to 17:59      3   9     1   5     1   5     0   2
18:00 to 18:59      7  14     3   8     3  10     0   3
19:00 to 19:59     18  29    16  28    16  30     2   7
20:00 to 20:59     26  39    24  37    24  36     4  11
21:00 to 21:59     27  39    23  34    26  38     5  11
22:00 to 22:59     21  33    18  32    23  34     4  11
23:00 to 23:59     14  25    11  22    15  26     2   7
00:00 to 00:59      9  17    10  20    11  19     1   3
01:00 to 01:59      6  17     9  20     9  17     0   2
02:00 to 02:59      1   7     4  11     4  10     0   2
03:00 to 03:59      0   2     0   5     0   6     0   1
```

Figure 1 – Sample Table Load Profile

The managers would like to be able to produce the report for a selectable analysis period, identified by trading day dates and in addition would like to be able to generate the report for selection of days of the week. For example, they would like to be able to generate the report for Mondays, Tuesdays, Wednesdays and Thursdays between 1 January 2008 and 30 June 2008.

The reporting program should be initiated by a command line similar to the example in Figure 2.

```
TableLoad –D 01-01-08 30-06-08 –W MON TUE WED THU
```

Figure 2 – Example Command Line

Specifically, the reporting program must accept a command line with the syntax defined in Figure 3. The from-date and to-date are inclusive. For example if the reporting period is defined by the option '–D 01-01-08 30-06-08', then the report must include the data for 1 January 2008 and 30 June 2008.

The reporting period, identified by '–D' is a mandatory parameter. The reporting program should reject requests that do not include the reporting period option.

The selected days of the week, identified by '–W' is an optional parameter. If the user does not include the '-W' parameter in the command line, the reporting program should generate the Table Load Profile for all days of the week.

```
command_line   : "TableLoad" option_list
               ;
option_list    : option_list option
               | /*empty*/
               ;
option         : "–D" from_date to_date
               | "–W" day_list
               ;
day_list       : day_list day
               | day
               ;
day            : "MON" | "TUE" | "WED" | "THU" | "FRI"
               | "SAT" | "SUN"
               ;
```

Where:

`command_line`
: is the command line submitted by the user.

`from_date`
: is the from-date of the report in DD-MM-YY format.

`to_date`
: is the to-date of the report in DD-MM-YY format.

Figure 3 – Command Line Syntax

The Table Load Profile must display the following information:

1. date and time the report was generated;
2. the codes ('MON', 'TUE', etc) for the selected days of the week or the words 'for all days' if all days of the week are selected;
3. the from-date and to-date;
4. for each hourly segment:
    a. for each group size segment (i.e. 1 to 2 guests, 3 to 4 guests, 5 to 6 guests and more than 7 guests):
        i. average peak load,
        ii. maximum peak load.

The peak load for an hourly segment on a particular trading day is the maximum number of concurrently open tables during the hourly segment. For example, if one group of guests arrived at 12:40 (as recorded in the orders table) and departed at 13:20 (as recorded in the settlements table) and another group of guests arrived at 13:30 and departed at 14:50, then the peak load between 13:00 and 13:59 would be 1 group, but if the second group arrived at 13:10 instead of 13:30, then the peak load would be 2 groups.

The average peak load is the sum of the peak loads for each trading day selected by the user divided by the number of trading days.

The maximum peak load is the highest peak load during the trading days selected by the user.

All dates should be input and displayed in DD-MM-YY format.

The Table Load Profile should be laid out in accordance with the layout in Figure 4.

```
         1         2         3         4         5         6
1...5....0....5....0....5....0....5....0....5....0....5....0
DD-MM-YY HH:MM      TABLE LOAD PROFILE              PAGE X-X
                    for X-X, X-X, X-X
               between XX-XX-XX and XX-XX-XX


Hourly                Peak Concurrently Open Tables
Segment             1-2       3-4       5-6    7-or-more
                  Avg Max   Avg Max   Avg Max   Avg Max
11:00 to 11:59    X-X X-X   X-X X-X   X-X X-X   X-X X-X
12:00 to 12:59    X-X X-X   X-X X-X   X-X X-X   X-X X-X
13:00 to 13:59    X-X X-X   X-X X-X   X-X X-X   X-X X-X
  :         :       :   :     :   :     :   :     :   :
  :         :       :   :     :   :     :   :     :   :
```

Figure 4 – Table Load Profile Layout

Programmers writing in C or C++ may make use of the date conversion functions described in Figure 5. Java and C-sharp programmers should use the date conversion functions provided by the Java and C-sharp language libraries.

```
#include "dtime.h"
                        // Include declarations of double time
                        // functions.
typedef double Dtime_t;
                        // Double time data type.  Dates and times are
                        // stored in seconds since 00:00 on
                        // 1 January 1970.  Dates are stored as seconds
                        // since 00:00 on 1 January 1970 until 00:00 on
                        // the stored date.
static const Dtime_t NULL_DTIME = -(65536.0*65536.0*65536.0*65536.0);
                        // Reserved double time value (before the big
                        // bang) used to represent an invalid time.
Dtime_t PackDtime (int yr, int mo, int dy, int hr, int mi, int se);
                        // Pack a double time value from its
                        // components.  yr is a 4-digit year (e.g.
                        // 2008).  mo is the calendar month number
                        // (e.g. 1 for January).  dy is the day of the
                        // month (e.g. 1 for the first of the month).
                        // hr, mi and se are the hour, minute and
                        // second components of a 24-hour clock.
                        // PackDtime returns NULL_DTIME if the
                        // components do not represent a valid date
                        // and time.
void UnpackDtime (Dtime_t dtime, int *yr, int *mo, int *dy,
      int *hr, int *mi, int *se);
                        // Unpack a double time value into its
                        // components.
Dtime_t DtimeFromSql (const char *sqlTime);
                        // Convert an SQL date or timestamp string into
                        // double time format.  DtimeFromSql returns
                        // NULL_DTIME if the SQL date or timestamp
                        // string is invalid.
char *SqlDateFromDtime (char *sqlDate, Dtime_t dtime);
                        // Load an SQL date string from a double time
                        // value.  SqlDateFromDtime returns sqlDate.
char *SqlTimestampFromDtime (char *sqlTimestamp, Dtime_t dtime);
                        // Load an SQL timestamp string from a double
                        // time value.  SqlTimestampFromDtime returns
                        // sqlTimestamp.
Dtime_t DtimeFromUserDate (const char *userDate);
                        // Convert a user date in DD-MM-YY format into
                        // the double time format.
char *UserDateFromDtime (char *userDate, Dtime_t dtime);
                        // Convert the date components of a double time
                        // value into a user date string in DD-MM-YY
                        // format. UserDateFromDtime returns userDate.
char *UserTimeFromDtime (char *userTime, Dtime_t dtime);
                        // Convert the time components of a double time
                        // value into a user time string in HH:MM:SS
                        // format. UserTimeFromDtime returns userTime.
```

Figure 5 – Double Time Functions

2 RUNNER STATUS DISPLAY

Cash-in-a-Flash is a money-changing business that operates a team of runners who pick up and deliver bundles of local and foreign cash from clients. There have been a few incidents in which runners have been mugged and have been overdue for a long time before the management realised something was wrong. The management of Cash-in-a-Flash has decided to install a computer system to monitor the movements of the runners so that these situations are highlighted more quickly.

The idea is that when a customer telephones in with a request, the telephone operator will key the request into the computer system.

The computer system will then show the request on a dynamic Runner Status Display. The runner manager will then see the request on the Runner Status Display and allocate a runner to the request. When the runner leaves Cash-in-a-Flash's office, the runner manager will click the request on the Runner Status Display, which will show a dialog box to receive the runner id of the runner. On submission of the dialog box, the computer system will change the colour of the request to indicate that a runner has been dispatched.

When the runner arrives at the customer's premises and changes the money, he will make an entry in a hand-held terminal, which will signal to the computer system to change the colour of the request to indicate that the runner has changed the money.

When the runner returns to Cash-in-a-Flash's office, the runner manager will click the Runner Status Display again to signal that the received money has been returned. This final operation will drop the request from the Runner Status Display.

If the runner becomes overdue, either for changing the money or returning to the office, the computer system will change the colour of the request to red to signal to the runner manager that there is a problem.

The proposal is that the computer system will be a client-server system that uses HTTP to commnicate between the clients and the server. Your task is to program the client module that shows the Runner Status Display.

The Runner Status Display should be laid out as shown in Figure 6.

Runner Status Display					
Address	Amount	Runner	Request	Dispatch	Change
54 Jalan Damar Minyak	3000	18	10:12	10:23	10:47
27 Jalan Robson	1500	27	10:32	10:45	
7 Jalan Mesra	1200		10:37		
38 Jalan Bukit Ceylon	1800		10:49		

Figure 6 – Runner Status Display Layout

The client module obtains the data to be shown on the display by sending the following URL to the server:

http://cashinaflash.com/RunnerStatusData

The server responds by sending a file similar to the example in Figure 7 to the client in the HTTP response.

```
1889297
7787<t>2<t>0<t>54 Jalan Damar Minyak<t>3000<t>18<t>10:12<t>10:23<t>10:47
7788<t>1<t>0<t>27 Jalan Robson<t>1500<t>27<t>10:32<t>10:45<t>
7789<t>0<t>1<t>7 Jalan Mesra<t>1200<t><t>10:37<t><t>
```

Figure 7 – Runner Status Data Format

The first line of the runner status data returned in the response contains a serial number that identifies the version of the runner status data. Each time the runner status data changes, the server increments the serial number.

The second and subsequent lines contain data for each outstanding request. Each line has 7 columns of data, each separated by a tab character (ASCII code 9, represented by '<t>' in the example). The columns contain the following information:

1. request number;
2. request status code;
3. overdue status code;
4. address;
5. transaction amount;
6. runner id;
7. request time;
8. dispatch time;
9. change time.

The runner id, dispatch time and change time columns may be empty if the event that loads a value into the column has not yet occurred.

The request and overdue status codes are described in Table 1. The client module must display the request in the colour corresponding to the request's request and overdue status codes.

When the client module is initiated, it must send a RunnerStatusData request to the server to obtain the initial state of the Runner Status Display.

Once the client module has obtained the initial state of the Runner Status Display, it must send the following HTTP request to obtain updates to the display:

http://cashinaflash.com/RunnerStatusData?fromSerNo=1889297

Where 1889297 is the serial number in the last runner status data file received by the client module.

Table 1 – Request and Overdue Status Codes

Request Status Code	Overdue Status Code	Meaning	Colour Components		
			Red	Green	Blue
0	0	Requested	255	204	153
1	0	Dispatched	0	204	255
2	0	Money changed	255	255	0
As above	1	Overdue	255	0	0

When the server receives a request with a fromSerNo parameter, it compares the serial number in the request to the latest serial number. If the serial number in the request is greater than or equal to the latest serial number, the server does not respond immediately, but holds the request until it processes another request that updates the runner status data. When another request updates the runner status data, the server increments the latest serial number and then sends a response containing the latest serial number back to the original client. If the serial number in the request is less than the latest serial number, the server responds immediately in the same way as it does for a request without the 'fromSerNo=' parameter.

The client module must refresh the Runner Status Display by executing the following cycle:

1. Send a RunnerStatusData request without any parameter to the server.
2. Receive and display the initial runner status data.
3. Loop:
    a. Send the server a RunnerStatusData request with a 'fromSerNo=' parameter containing the last serial number received from server.
    b. Receive and display the updated runner status data.

In effect, the normal state of the thread in the client module that updates the Runner Status Display is to be waiting to receive an updated runner status data file from the server.

Concurrently with receiving runner status data updates from the server, the client module must also be monitoring the mouse. If the user clicks the mouse over a request line with request status 0 (requested), the client module must display a dialog box similar to the example in Figure 8.

If the user enters a runner id and clicks the OK button, the client module must send the following HTTP request to the server.

http://cashinaflash.com/Dispatch?reqNo=7789&runId=12

Where '7789' is the request number selected by the mouse click and '12' is the runner identifier entered by the user.

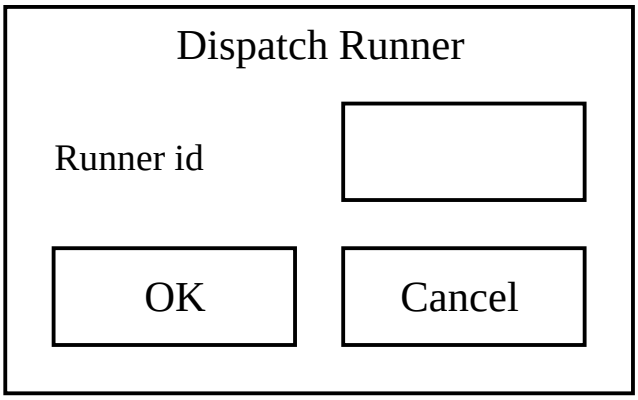

Figure 8 – Dispatch Runner Dialog Box

If the server accepts the Dispatch request, it returns an empty response. If it rejects the Dispatch request, it returns an HTTP response containing a status code that represents a client error.

If the user clicks the cancel button, the client module must close the dialog box and return to its idle state.

If the user clicks the mouse over a request line with request status 2 (money changed), the client module must display a dialog box similar to the example in Figure 9.

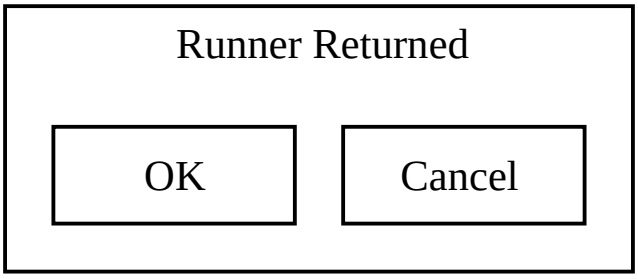

Figure 9 – Runner Returned Dialog Box

If the user clicks the OK button, the client module must send the following HTTP request to the server.

http://cashinaflash.com/Return?reqNo=7789

Where ‘7789’ is the request number selected by the mouse click.

If the server accepts the Return request, it returns an empty response. If it rejects the Return request, it returns an HTTP response containing a status code that represents a client error.

If the user clicks the cancel button, the client module must close the dialog box and return to its idle state.

The client module may have a limit of 12 concurrent requests, so that all the requests can be displayed on the screen at the same time without scrolling.

3 DENOMINATION/QUANTITY DATA ENTRY BLOCK

A banking application requires a breakdown of banknotes of various denominations to be entered into several different data entry screens. For example, a denomination breakdown is to be entered into the Transfer Funds data entry screen, to record the breakdown of bank notes to be transferred between one teller and another; and into the Close Teller data entry screen, to record the breakdown of banknotes at a teller's station when the station is closed.

In all cases the data entry fields for receiving a denomination breakdown from the user must be laid out in the manner shown in Figure 10.

Denomination	Quantity	Value
100	26	2600
50	87	4350
10	96	960
5	17	85
1	145	145
Total		8140

Figure 10 – Denomination Breakdown Block

When a user enters a value into a quantity field, the application must automatically update the corresponding value field and the total at the bottom of the block.

The intended layout of the Transfer Funds data entry screen is presented in Figure 11.

Although the denomination breakdown block will appear in several data entry screens, it must not be a separate data entry screen or dialog box in its own right. In the Transfer Funds data entry screen, the user must be able to press the tab key to move from the from-teller field to the to-teller field and then again to move from the to-teller field to the quantity of $100 bills and then the quantity of $50 bills and so on.

To avoid duplication, the chief software engineer would like to have a single program module that creates and manages a denomination breakdown block and have the applications, such as the Transfer Funds data entry screen, make use of the program module to create and manage the denomination breakdown blocks in the individual data entry screens. Your task is to design and program the program module that creates and manages a denomination breakdown block and to create a prototype Transfer Funds program that demonstrates the use of the denomination breakdown module that you have designed and programmed.

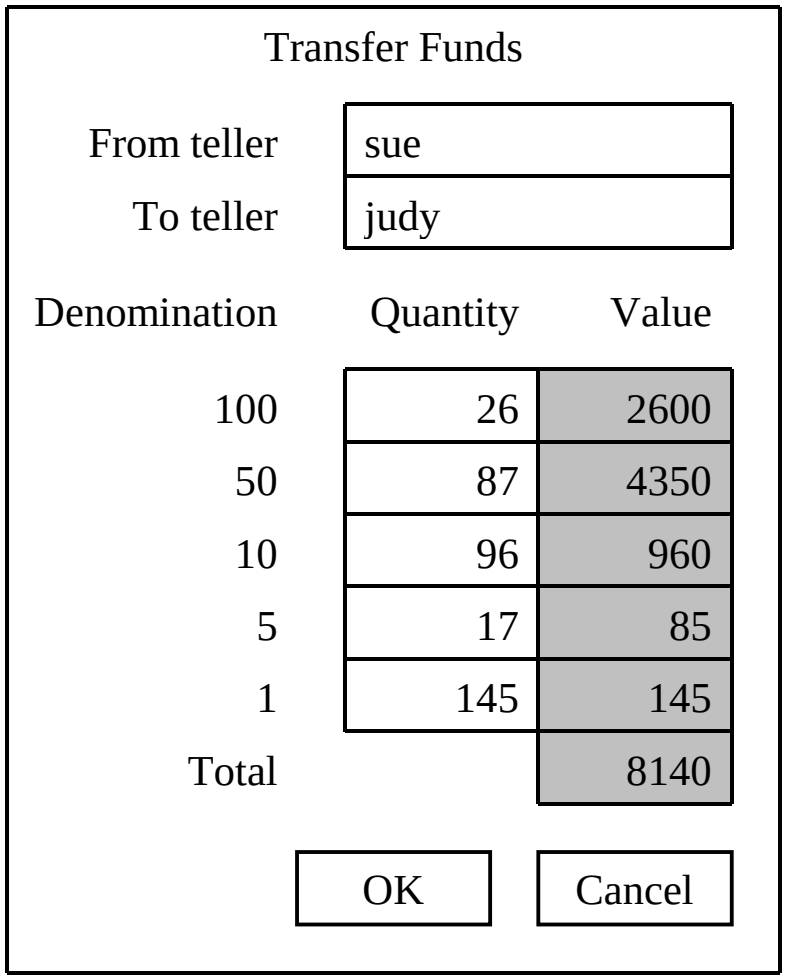

Figure 11 – Transfer Funds Data Entry Screen

The denomination breakdown module must display the static text fields in the denomination breakdown block and create the quantity fields and the display-only value fields. It must process keystrokes entered into the quantity fields and automatically update the value and total fields as values are entered into the quantity fields. It must be designed in a manner that will allow it to be used by multiple applications.

The denomination breakdown module must have a means for the application to retrieve an array of denomination/quantity tuplets that contains the denominations with non-zero quantities and the corresponding quantities.

If the user clicks the OK button on the prototype Transfer Funds data entry screen, the data entry screen program must log the entered values to an output that can be viewed by the user. If you are programming in Java, the program could log the values to System.out via, say, System.out.println. If your preferred programming language does not have an equivalent to Java's System.out, the program could write the values to a disk file.

If the user clicks the Cancel button, the data entry screen program should quietly exit.

4 DATA STREAM DECODER

A stream of data coming from a serial interface consists of blocks of data of the form shown in Figure 12.

Length byte 1	Length byte 2	Data bytes

Figure 12 – Data Stream Block Format

The two length bytes contain the number of data bytes in the block. Length byte 1 contains the high-order 8 bits and length byte 2, the low-order 8 bits.

Each data block, except the last, contains exactly 1022 data bytes. The last block contains between 0 and 1021 data bytes. The last data block signals the end of the data stream.

Your task is to write a function that allows an application to read an arbitrary number of bytes from the input stream without regard to the lower-level blocking arrangements.

If you are writing in C or C++, your function must have the following application interface:

```
int SerialRead (void *buf, int len);
```

`buf`
: is a pointer to a buffer that is to be loaded with the received data.

`len`
: is the number of bytes to be loaded into the buffer.

`return value`
: is the number of bytes that were actually loaded into the buffer.

If you are writing in C or C++, your function may make use of the following function to receive data from the serial interface:

```
int SerialRecv (void *buf, int len);
```

`buf`
: is a pointer to a buffer that is to be loaded with the received data.

`len`
: is the number of bytes to be received.

`return value`
: is the number of bytes that were actually received.

If you are writing in Java, your function must have the following application interface:

```
public class SerialReader {
     // ...
     public static byte[] serialRead (int len);
     // ...
}
```

`len`
: is the number of bytes to be returned.

`return value`
: is a byte array containing the bytes retrieved from the input stream.

If you are writing in Java, your function may make use of the following function to receive data from the serial interface:

```
public class SerialReceiver {
     // ...
     public static byte[] serialRecv (int len);
     // ...
}
```

`len`
: is the number of bytes to be received.

`return value`
: is a byte array containing the bytes received from the serial interface.

If the ‘len’ parameter passed to SerialRead is less than or equal to zero, SerialRead must quitely return zero without reading any data.

If the ‘len’ parameter is greater than zero, and there are more than ‘len’ data bytes remaining in the input stream, SerialRead must return exactly ‘len’ data bytes. If the ‘len’ parameter is greater than zero, and there are less than ‘len’ data bytes remaining in the input stream, SerialRead must return the number of data bytes remaining in the input stream.

Repeated calls to SerialRead when no data bytes remain in the input stream must return zero.

The SerialRecv function may return between 1 and ‘len’ data bytes. No meaning is to be attached to SerialRecv returning less than ‘len’ data bytes. If SerialRecv is called and there are data bytes in the input ring buffer, SerialRecv returns the data bytes in the input ring buffer without waiting. If the input ring buffer is empty, SerialRecv blocks until one or more data bytes are loaded into the input ring buffer and then returns the data bytes to its caller.

The SerialRead function may make use of static memory to store data between successive calls. It is not required to be re-enterant (i.e. thread-safe).

For optimal efficiency, SerialRead should read data from SerialRecv in blocks of 16 characters.

PERTANDINGAN PENGATURCARAAN E-GENTING 2008

Arahan-arahan am:

1. Jawab satu atau lebih soalan yang diberikan.
2. Pertandingan ini adalah satu ujian di mana peserta-peserta dibenarkan merujuk kepada buku-buku atau bahan-bahan rujukan.
3. Masa yang diperuntukan untuk pertandingan ini adalah 8 jam.
4. Perbincangan sesama peserta tidak dibenarkan.
5. Untuk menerima kredit dalam 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. Sekiranya jawapan anda merupakan penyelesaian yang munasabah, anda akan menerima kredit untuk hasilan methodical seperti gambar rajah aliran data, gambar rajah peralihan keadaan, jadual dan sebagainya.
7. Jumlah markah anda adalah jumlah kredit yang anda perolehi bagi setiap penyelesaian yang munasabah.
8. Program-program anda akan dinilai berdasarkan kepada betapa mudah ianya boleh dibaca dan difahami.
    - Takukan mestilah bersih dan 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 diperuntukan kepada setiap soalan adalah seperti berikut:

No	Tajuk	Markah
1.	Profil Pengunaan Meja	280
2.	Paparan Status Penghantar	330
3.	Blok Input Denominasi dan Kuantiti	220
4.	Penyahkod Data	40

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

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

1 PROFIL PENGGUNAAN MEJA

Pihak pengurusan Restoran Ocean Treasure mendapati bahawa kadang-kala mereka terpaksa menolak pelanggan berkumpulan besar hanya kerana meja besar mereka telah diduduki oleh pelanggan berkumpulan kecil. Mereka mengesyaki kombinasi meja-meja mereka adalah tidak berkesan dan merasakan mereka harus meletakkan lebih banyak meja bersaiz sederhana dan mengurangkan meja bersaiz besar.

Untuk menentukan samada ini merupakan idea yang baik, mereka ingin menganalisis pangkalan data mereka untuk menentukan anggaran bilangan kumpulan pelanggan yang mengunjungi restoran tersebut pada setiap jam. Tugas anda ialah mengaturcarakan satu aturcara komputer yang akan melakukan analisis tersebut dan menghasilkan satu laporan, “Profil Penggunaan Meja” yang merumuskan keputusan analisis tersebut.

Terdapat beberapa jadual pangkalan data dalam sistem pesanan dan akaun Ocean Treasure yang mungkin dapat membantu dalam analisis tersebut.

Setiap kali satu pesanan diambil dari sesebuah meja, pesanan tersebut akan direkodkan ke satu jadual pesanan. Struktur jadual pesanan adalah seperti berikut:

```
create table orders (
     orderNo        integer not null,
     orderDate      date not null,
     orderTime      timestamp not null,
     tableNo        char(4) not null,
     covers         integer not null,
     waiterCode     char(8) not null
);
```

`orders`
: merupakan satu jadual yang mengandungi satu rekod untuk setiap pesanan baru.

`orderNo`
: merupakan satu nombor unik yang mengenalpasti sesuatu pesanan.

`orderDate`
: merupakan tarikh urus niaga bagi pesanan.

`orderTime`
: merupakan masa apabila pesanan diambil.

`tableNo`
: merupakan kod unik yang mengenalpasti meja di mana pesanan diambil.

`covers`
: bagi kumpulan pelanggan yang baru, ia merupakan bilangan pelanggan yang menduduki meja tersebut; bagi pesanan seterusnya yang datang dari kumpulan pelanggan yang sama, nilainya ialah sifar (0).

`waiterCode`
: merupakan kod unik yang mengenalpasti pelayan yang mengambil pesanan tersebut.

Apabila sekumpulan pelanggan membayar bil, bayaran tersebut direkodkan di satu jadual bayaran. Struktur jadual bayaran adalah seperti berikut:

```
create table settlements (
     tableNo        char(4) not null,
     settleDate     date not null,
     settleTime     timestamp not null,
     settleMode     smallint not null,
     billAmount     double precision not null,
     tipAmount      double precision not null,
     refCode        char(32)
);
```

`settlements`
: merupakan satu jadual yang mengandungi satu rekod untuk setiap bayaran yang dibuat.

`tableNo`
: merupakan kod yang berada dalam domain yang sama dengan orders.tableNo yang mengenalpasti meja di mana pelanggan duduk.

`settleDate`
: merupakan tarikh urus niaga bagi bayaran tersebut.

`settleTime`
: merupakan masa apabila bayaran dibuat.

`settleMode`
: merupakan kod yang mengenalpasti cara bayaran dibuat. 0 bermaksud tunai, 1 bermaksud Visa, 2 bermaksud Mastercard dan sebagainya.

`billAmount`
: merupakan jumlah amaun bagi bil yang dibayar.

`tipAmount`
: merupakan amaun tip yang diberi oleh pelanggan.

`refCode`
: merupakan kod rujukan, yang mempunyai makna yang berbeza bergantung kepada cara bayaran. Jika cara bayaran melibatkan kad kredit, ia merupakan nombor rujukan yang dicetak pada slip kredit kad.

Restoran Ocean Treasure dibuka pada kira-kira 11 pagi, dan tutup di antara 2 hingga 3 pagi pada keesokan hari, bergantung kepada keperluan pelanggan. Sesi perniagaan tersebut yang berada di antara 11 pagi hingga 2 pagi keesokan hari dirujuk sebagai satu hari urus niaga. Restoran Ocean Treasure mengira activiti urus niaganya dalam hari urus niaga dan bukan hari sebenar.

Pengurus Restoran Ocean Treasure ingin menjanakan satu laporan seperti dalam Figure 13.

```
01-09-08 19:49      TABLE LOAD PROFILE              PAGE   1
                  for MON, TUE, WED, THU
              between 01-01-08 and 30-06-08

Hourly                Peak Concurrently Open Tables
Segment             1-2       3-4       5-6    7-or-more
                  Avg Max   Avg Max   Avg Max   Avg Max
11:00 to 11:59      1   3     0   2     0   3     0   1
12:00 to 12:59      4  10     1   5     2   7     0   3
13:00 to 13:59     12  22     5  12     5  10     0   3
14:00 to 14:59     13  23     5  13     5  12     1   4
15:00 to 15:59      9  16     3   8     4  11     0   4
16:00 to 16:59      4  10     1   4     1   5     0   3
17:00 to 17:59      3   9     1   5     1   5     0   2
18:00 to 18:59      7  14     3   8     3  10     0   3
19:00 to 19:59     18  29    16  28    16  30     2   7
20:00 to 20:59     26  39    24  37    24  36     4  11
21:00 to 21:59     27  39    23  34    26  38     5  11
22:00 to 22:59     21  33    18  32    23  34     4  11
23:00 to 23:59     14  25    11  22    15  26     2   7
00:00 to 00:59      9  17    10  20    11  19     1   3
01:00 to 01:59      6  17     9  20     9  17     0   2
02:00 to 02:59      1   7     4  11     4  10     0   2
03:00 to 03:59      0   2     0   5     0   6     0   1
```

Figure 13 – Contoh Profail Penggunaan Meja

Pengurus tersebut ingin menjana laporan tersebut untuk tempoh tertentu, yang dikenalpasti dengan menggunakan tarikh urus niaga dan juga hari minggu. Contohnya, beliau ingin menjana laporan tersebut untuk Hari Isnin, Selasa, Rabu dan Khamis di antara 1 Januari 2008 hingga 30 Jun 2008.

Laporan tersebut harus dijana melalui antara muka perintah seperti dalam Figure 14 di bawah:

```
TableLoad –D 01-01-08 30-06-08 –W MON TUE WED THU
```

Figure 14 – Contoh antara muka perintah

Secara khususnya, aturcara laporan tersebut mesti menerima antara muka perintah dengan sintaks yang didefinasikan dalam Figure 15. Tarikh-dari dan tarikh-hingga adalah terangkum. Contohnya, jika tempoh laporan diberi sebagai ‘–D 01-01-08 30-06-08’, maka laporan tersebut mesti merangkumi data bagi 1 Januari 2008 dan 30 Jun 2008.

Tempoh laporan, dikenalpasti sebagai ‘-D’ adalah parameter wajib. Aturcara laporan tersebut harus menolak permintaan yang tidak mengandungi tempoh laporan.

Hari didalam minggu yang dikenalpasti dengan ‘-W’ adalah parameter pilihan. Jika parameter ‘-W’ tidak diberikan, aturcara laporan tersebut harus menjana Profail Penggunaan Meja bagi semua hari minggu.

```
command_line   : "TableLoad" option_list
               ;
option_list    : option_list option
               | /*empty*/
               ;
option         : "–D" from_date to_date
               | "–W" day_list
               ;
day_list       : day_list day
               | day
               ;
day            : "MON" | "TUE" | "WED" | "THU" | "FRI"
               | "SAT" | "SUN"
               ;
```

Where:

`command_line`
: merupakan perintah yang dihantar oleh pengguna.

`from_date`
: merupakan tarikh-dari bagi laporan tersebut dalam format DD-MM-YY.

`to_date`
: merupakan tarikh-hingga bagi laporan tersebut dalam format DD-MM-YY.

Figure 15 – Sintaks Antara muka perintah

Profail Penggunaan Meja mesti memaparkan maklumat berikut:

1. Tarikh dan masa laporan tersebut dijanakan;
2. Kod (‘MON’, ‘TUE’, dan sebagainya) bagi hari minggu yang terpilih atau frasa ‘for all days’ jika semua hari minggu terpilih;
3. tarikh-dari dan tarikh-hingga;
4. bagi setiap jam segmen:
    a. bagi setiap saiz kumpulan segmen (iaitu 1 hingga 2 pelanggan, 3 hingga 4 pelanggan, 5 hingga 6 pelanggan dan lebih 7 pelanggan):
        i. muatan puncak purata,
        ii. muatan puncak maximum.

Muatan puncak bagi sesuatu jam segmen pada sesuatu hari urus niaga ialah bilangan maximum meja yang dibuka sejajar pada jam segmen tersebut. Contohnya, jika satu kumpulan pelanggan tiba pada 12:40 (seperti yang direkodkan dalam jadual pesanan) dan meninggal pada 13:20 (seperti yang direkodkan dalam jadual bayaran) dan satu

kumpulan pelanggan lagi tiba pada 13:30 dan meninggal pada 14:50, maka muatan puncak di antara 13:00 hingga 13:59 ialah 1 kumpulan, tetapi jika kumpulan kedua tiba pada 13:10 dan bukan 13:30, maka muatan puncak ialah 2 kumpulan.

Purata muatan puncak dikira sebagai jumlah muatan puncak bagi setiap hari urus niaga yang terpilih oleh pengguna dibahagi dengan bilangan hari urus niaga.

Muatan puncak maksimum ialah muatan puncak yang tertinggi di antara semua hari urus niaga yang terpilih oleh pengguna.

Semua tarikh harus diinput dan dipaparkan dalam format DD-MM-YY.

Profail Penggunaan Meja tersebut harus dipaparkan seperti dalam Figure 16.

```
         1         2         3         4         5         6
1...5....0....5....0....5....0....5....0....5....0....5....0
DD-MM-YY HH:MM      TABLE LOAD PROFILE              PAGE X-X
                     for X-X, X-X, X-X
               between XX-XX-XX and XX-XX-XX


Hourly                 Peak Concurrently Open Tables
Segment              1-2       3-4       5-6    7-or-more
                   Avg Max   Avg Max   Avg Max   Avg Max
11:00 to 11:59     X-X X-X   X-X X-X   X-X X-X   X-X X-X
12:00 to 12:59     X-X X-X   X-X X-X   X-X X-X   X-X X-X
13:00 to 13:59     X-X X-X   X-X X-X   X-X X-X   X-X X-X
  :        :        :   :     :   :     :   :     :   :
  :        :        :   :     :   :     :   :     :   :
```

Figure 16 – Paparan Profail Penggunaan Meja

Pengaturcara yang mengaturcara dalam C atau C++ boleh menggunakan fungsi-fungsi penukaran tarikh dalam Figure 17. Pengaturcara Java dan C-sharp boleh menggunakan fungsi-fungsi penukaran tarikh yang terkandung dalam perpustakaan Java dan C-sharp.

```
#include "dtime.h"
                        // Include declarations of double time
                        // functions.
typedef double Dtime_t;
                        // Double time data type.  Dates and times are
                        // stored in seconds since 00:00 on
                        // 1 January 1970.  Dates are stored as seconds
                        // since 00:00 on 1 January 1970 until 00:00 on
                        // the stored date.
static const Dtime_t NULL_DTIME = -(65536.0*65536.0*65536.0*65536.0);
                        // Reserved double time value (before the big
                        // bang) used to represent an invalid time.
Dtime_t PackDtime (int yr, int mo, int dy, int hr, int mi, int se);
                        // Pack a double time value from its
                        // components.  yr is a 4-digit year (e.g.
                        // 2008).  mo is the calendar month number
                        // (e.g. 1 for January).  dy is the day of the
                        // month (e.g. 1 for the first of the month).
                        // hr, mi and se are the hour, minute and
                        // second components of a 24-hour clock.
                        // PackDtime returns NULL_DTIME if the
                        // components do not represent a valid date
                        // and time.
void UnpackDtime (Dtime_t dtime, int *yr, int *mo, int *dy,
      int *hr, int *mi, int *se);
                        // Unpack a double time value into its
                        // components.
Dtime_t DtimeFromSql (const char *sqlTime);
                        // Convert an SQL date or timestamp string into
                        // double time format.  DtimeFromSql returns
                        // NULL_DTIME if the SQL date or timestamp
                        // string is invalid.
char *SqlDateFromDtime (char *sqlDate, Dtime_t dtime);
                        // Load an SQL date string from a double time
                        // value.  SqlDateFromDtime returns sqlDate.
char *SqlTimestampFromDtime (char *sqlTimestamp, Dtime_t dtime);
                        // Load an SQL timestamp string from a double
                        // time value.  SqlTimestampFromDtime returns
                        // sqlTimestamp.
Dtime_t DtimeFromUserDate (const char *userDate);
                        // Convert a user date in DD-MM-YY format into
                        // the double time format.
char *UserDateFromDtime (char *userDate, Dtime_t dtime);
                        // Convert the date components of a double time
                        // value into a user date string in DD-MM-YY
                        // format. UserDateFromDtime returns userDate.
char *UserTimeFromDtime (char *userTime, Dtime_t dtime);
                        // Convert the time components of a double time
                        // value into a user time string in HH:MM:SS
                        // format. UserTimeFromDtime returns userTime.
```

Figure 17 – Fungsi-fungsi tarikh dan masa

2 PAPARAN STATUS PENGHANTAR

Cash-in-a-Flash merupakan sebuah perniagaan penukaran mata wang yang beroperasi dengan sepasukan penghantar yang mengutip dan menghantar mata wang tempatan dan asing daripada pelanggan. Terdapat beberapa insiden di mana penghantar telah dirompak lalu mengakibatkan kelewatan yang panjang sebelum pihak pengurusan mendapati mengenalpasti isu tersebut. Pihak pengurusan Cash-in-a-Flash mengambil keputusan untuk memasang satu sistem komputer untuk mengawasi gerakan penghantar agar situasi sebegini dapat dilaporkan dengan lebih cepat.

Ideanya ialah apabila seseorang pelanggan menelefon dengan satu permintaan, operator telefon akan memasukkan permintaan tersebut ke dalam sistem komputer.

Sistem komputer tersebut kemudiannya akan memaparkan permintaan tersebut pada satu Paparan Status Penghantar dinamik. Pengurus penghantar akan melihat permintaan pada Paparan Status Penghantar dan memperuntukkan seorang penghantar. Apabila penghantar tersebut meninggalkan pejabat Cash-in-a-Flash, pengurus penghantar akan memilih permintaan tersebut pada Paparan Status Penghantar, yang akan memaparkan petak dialog untuk menerima id penghantar bagi penghantar tersebut. Apabila input itu dihantar, sistem komputer tersebut akan menukar warna permintaan itu untuk menunjukkan bahawa penghantar telah dihantar.

Apabila penghantar tiba di premis pelanggan dan menukar mata wang, penghantar tersebut akan mengemaskini melalui satu terminal mudah-alih, yang akan menghantar isyarat kepada sistem komputer supaya menukar warna permintaan untuk menunjukkan bahawa penghantar telah melakukan penukaran mata wang.

Apabila penghantar tersebut kembali ke pejabat Cash-in-a-Flash, pengurus penghantar akan menggunakan Paparan Status Pengguna untuk memberi isyarat bahawa wang yang diterima telah dikembalikan. Operasi terakhir ini akan menggugurkan permintaan tersebut daripada Paparan Status Penghantar.

Jika penghantar tersebut lewat, sama ada untuk penukaran wang atau kembali ke pejabat, sistem komputer tersebut akan menukar warna permintaan ke merah untuk mengingatkan pengurus penghantar bahawa terdapat masalah dengan permintaan tersebut.

Cadangan yang dikemukakan ialah sistem komputer tersebut merupakan sistem klien-pelayan yang menggunakan HTTP untuk berkomunikasi di antara klien dan pelayan. Tugas anda ialah mengaturcara modul klien tersebut yang memaparkan Paparan Status Penghantar.

Paparan Status Penghantar harus diatur seperti dalam Figure 18.

Modul klien mencapai data yang akan dipaparkan pada paparan dengan menghantar URL seperti di bawah kepada pelayan:

http://cashinaflash.com/RunnerStatusData

Pelayan tersebut membalas dengan menghantar satu fail seperti dalam Figure 19 kepada klien dalam respon HTTP.

Runner Status Display					
Address	Amount	Runner	Request	Dispatch	Change
54 Jalan Damar Minyak	3000	18	10:12	10:23	10:47
27 Jalan Robson	1500	27	10:32	10:45	
7 Jalan Mesra	1200		10:37		
38 Jalan Bukit Ceylon	1800		10:49		

Figure 18 – Aturan Paparan Status Penghantar

```
1889297
7787<t>2<t>0<t>54 Jalan Damar Minyak<t>3000<t>18<t>10:12<t>10:23<t>10:47
7788<t>1<t>0<t>27 Jalan Robson<t>1500<t>27<t>10:32<t>10:45<t>
7789<t>0<t>1<t>7 Jalan Mesra<t>1200<t><t>10:37<t><t>
```

Figure 19 – Format Data Status Penghantar

Baris pertama data status penghantar yang dikembalikan dalam respon mengandungi satu nombor siri yang mengenalpasti versi data status penghantar tersebut. Setiap kali data status penghantar berubah, pelayan menambah nombor siri tersebut.

Baris kedua dan baris berikutnya mengandungi data bagi setiap permintaan yang belum digugur. Setiap baris mempunyai 7 komponen data, setiapnya dipisah oleh satu aksara tab (kod ASCII 9, diwakili oleh ‘<t>’ dalam contoh). Komponen-komponen data tersebut ialah:

1. nombor permintaan;
2. kod status permintaan;
3. kod status lewat;
4. alamat;
5. amaun transaksi;
6. id penghantar;
7. masa permintaan;
8. masa menghantar;
9. masa menukar.

Id penghantar, masa menghantar dan masa menukar boleh dikosongkan jika situasi yang memuatkan nilai tersebut belum lagi berlaku.

Permintaan dan kod status lewat dihuraikan dalam Table 2. Modul klien tersebut mesti memaparkan permintaan dalam warna bersepadan dengan perkembangan permintaan dan kod status lewat.

Apabila modul klien dimulakan, ia mesti menghantar satu permintaan RunnerStatusData kepada pelayan untuk mencapai keadaan awal Paparan Status Penghantar.

Setelah modul klien berjaya mencapai keadaan awal Paparan Status Penghantar tersebut, ia mesti menghantar permintaan HTTP seperti berikut untuk mendapat kemaskini kepada paparan itu:

http://cashinaflash.com/RunnerStatusData?fromSerNo=1889297

Di mana 1889297 merupakan nombor siri dalam fail data status penghantar terakhir yang diterima oleh modul klien.

Table 2 – Permintaan dan Kod Status Lewat

Request Status Code	Overdue Status Code	Meaning	Colour Components		
			Red	Green	Blue
0	0	Requested	255	204	153
1	0	Dispatched	0	204	255
2	0	Money changed	255	255	0
As above	1	Overdue	255	0	0

Apabila pelayan menerima satu permintaan dengan satu parameter fromSerNo, ia akan membanding nombor siri dalam permintaan tersebut dengan nombor siri terakhir. Jika nombor siri dalam permintaan adalah lebih besar atau bersamaan dengan nombor siri terakhir, pelayan tidak akan memberi respon dengan serta merta, tetapi akan menangguh permintaan tersebut sehingga ia memproses permintaan lain yang mengemaskini data status penghantar. Apabila permintaan lain mengemaskini data status penghantar, pelayan menokok nombor siri terakhir dan kemudian menghantar satu respon yang mengandungi nombor siri terakhir kepada klien asal. Jika nombor siri dalam permintaan adalah kurang daripada nombor siri terakhir, pelayan memberi respon serta-merta dalam cara yang sama apabila ia bertindak kepada permintaan tanpa parameter ‘fromSerNo=’.

Modul klien mesti mengemaskini Paparan Status Penghantar dengan melaksanakan kitar berikut:

1. Hantar satu permintaan RunnerStatusData tanpa sebarang parameter kepada pelayan.
2. Menerima dan memaparkan data status penghantar awal.
3. Ulang:
    a. Hantar kepada pelayan satu permintaan RunnerStatusData dengan parameter ‘fromSerNo=’ yang mengandungi nombor siri terakhir yang diterima dari pelayan.
    b. Terima dan memaparkan data status penghantar terkini.

Ini bermaksud keadaan normal bagi untai modul klien yang mengemaskini Paparan Status Penghantar ialah menunggu untuk menerima fail data status penghantar daripada pelayan.

Sejajar dengan menerima kemaskini data status penghantar daripada pelayan, modul klien mesti juga mengawasi tetikus. Jika pengguna klik pada satu permintaan yang mempunyai status 0 (Requested), modul klien mesti memaparkan satu kotak dialog seperti dalam Figure 20.

Jika pengguna tersebut menginputkan satu id penghantar dan klik butang OK, modul klien mesti menghantar permintaan HTTP berikut kepada pelayan:

http://cashinaflash.com/Dispatch?reqNo=7789&runId=12

Di mana '7789' merupakan nombor permintaan yang dipilih oleh klik tetikus dan '12' ialah id penghantar yang diinput oleh pengguna tersebut.

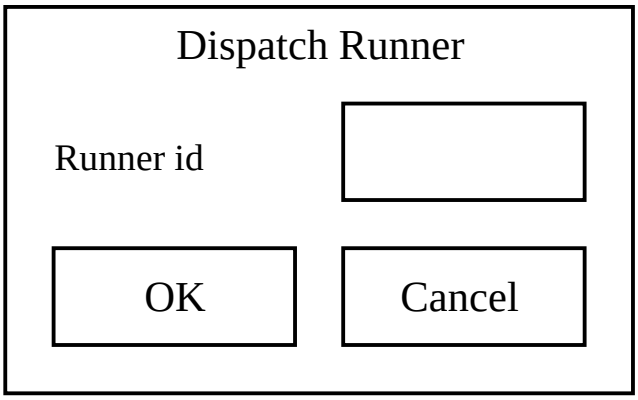

Figure 20 – Kotak Dialog Menghantar Penghantar

Jika pelayan menerima permintaan penghantaran (Dispatch) tersebut, ia akan mengembalikan satu respon kosong. Jika pelayan menolak permintaan penghantaran tersebut, ia akan mengembalikan satu respon HTTP yang mengandungi satu kod status yang mewakili satu kesilapan klien.

Jika pengguna klik pada butang cancel, modul klien tersebut mesti menutup kotak dialog tersebut dan kembali kepada keadaan leka.

Jika pengguna klik pada satu baris permintaan yang mempunyai status 2 (money changed), modul klien mesti memaparkan satu kotak dialog seperti dalam Figure 21.

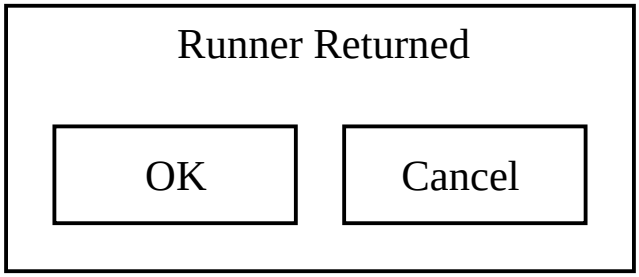

Figure 21 – Kotak Dialog Pengembalian Penghantar

Jika pengguna klik pada butang OK, modul klien mesti menghantar permintaan HTTP berikut kepada pelayan:

http://cashinaflash.com/Return?reqNo=7789

Di mana ‘7789’ merupakan nombor permintaan yang dipilih melalui klik tetikus.

Jika pelayan menerima permintaan pengembalian (Return) tersebut, ia akan mengembalikan satu respon kosong. Jika pelayan menolak permintaan pengembalian tersebut, ia akan mengembalikan satu respon HTTP yang mengandungi satu kod status yang mewakili satu kesilapan klien.

Jika pengguna klik pada butang cancel, modul klien tersebut mesti menutup kotak dialog tersebut dan kembali kepada keadaan leka.

Modul klien boleh dihadkan kepada 12 permintaan sejajar, supaya semua permintaan boleh dipaparkan pada skrin pada masa yang sama tanpa penatalan.

3 BLOK INPUT DENOMINASI DAN KUANTITI

Satu aplikasi bank memerlukan penghuraian wang kertas kepada berbagai denominasi dan dimasukkan ke beberapa skrin input data yang berbeza. Contohnya, penghuraian denominasi perlu dimasukkan ke skrin input data pemindahan wang bagi merekodkan huraian wang kertas yang perlu dipindah di antara satu juruwang dengan yang lain. Penghuraian denominasi juga perlu dimasukkan ke skrin input penutupan juruwang bagi merekodkan huraian wang kertas pada sesuatu stesen juruwang apabila stesen tersebut ditutup.

Dalam semua kes, skrin input untuk menerima huraian denominasi daripada pengguna mesti diaturkan seperti dalam Figure 22.

Denomination	Quantity	Value
100	26	2600
50	87	4350
10	96	960
5	17	85
1	145	145
Total		8140

Figure 22 – Blok Huraian Denominasi

Apabila pengguna memasukkan satu nilai ke dalam kotak kuantiti, aplikasi tersebut mesti mengemaskini kotak nilai dan kotak jumlah yang terletak di bawah secara automatik

Aturan bagi skrin pemindahan wang adalah seperti dalam Figure 23.

Walaupun blok huraian denominasi akan digunakan dalam beberapa skrin input data, ia tidak boleh merupakan skrin input atau kotak dialog yang berasingan. Dalam skrin input pemindahan wang, pengguna mesti dibenarkan untuk menggunakan kunci ‘tab’ untuk menggerakkan input dari ‘from-teller’ ke ‘to-teller’ dan kemudian dari ‘to-teller’ ke kuantiti wang kertas $100 dan kemudian ke kuantiti wang kertas $50 dan sebagainya.

Untuk mengelakkan pengulangan, ketua jurutera perisian berhasrat membina satu modul program yang menguruskan satu blok huraian denominasi dan aplikasi seperti skrin input data pemindahan wang akan menggunakan modul program tersebut untuk membina dan menguruskan blok huraian denominasi dalam skrin data input individu. Tugas anda ialah untuk mereka dan mengaturcara modul program tersebut yang membina dan mengurus blok huraian denominasi dan seterusnya membina satu prototaip pemindahan wang yang menunjukkan penggunaan modul huraian denominasi yang telah direka dan diaturcara.

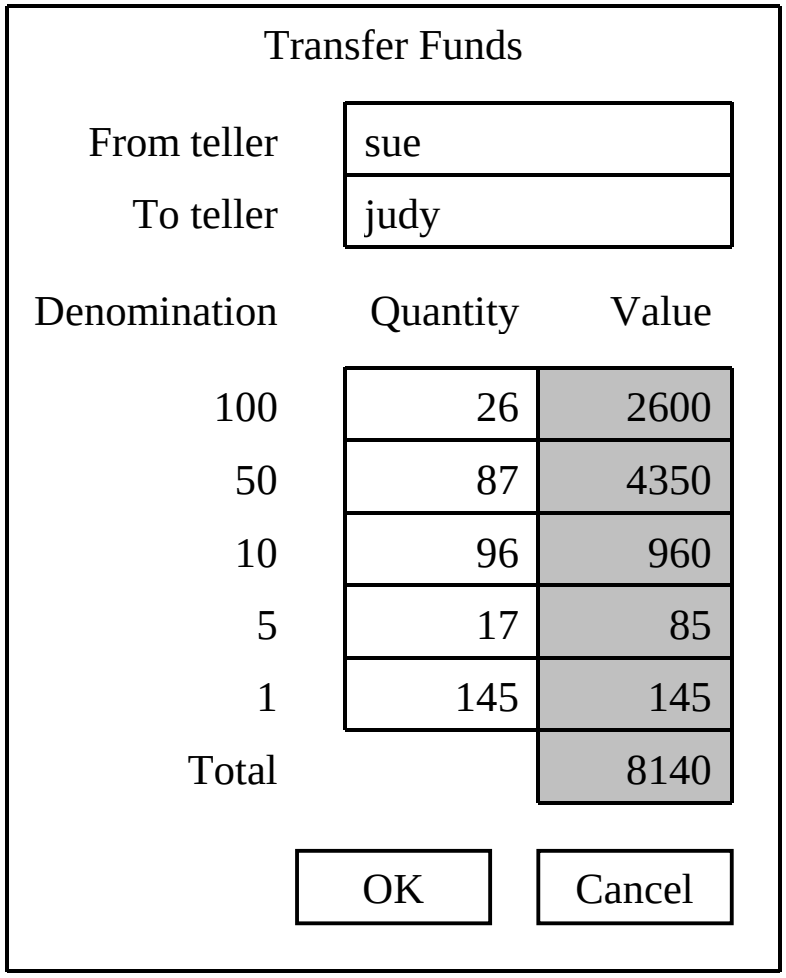

Figure 23 – Skrin Input Pemindahan Wang

Modul huraian denominasi tersebut mesti memaparkan teks statik dalam blok huraian denominasi dan membina kotak kuantiti dan kotak nilai untuk paparan sahaja. Ia mesti memproses input dalam kotak kuantiti dan mengemaskini kotak nilai dan jumlah secara automatik apabila nilai diinputkan ke kotak-kotak kuantiti. Ia mesti direka sedemikian supaya boleh digunakan oleh berbagai aplikasi.

Modul huraian denominasi tersebut mesti membekalkan cara supaya aplikasi boleh mencapai satu tatasusunan tuplet denominasi/kuantiti yang mengandungi denominasi dengan kuantiti bukan sifar.

Jika pengguna tersebut klik butang OK pada skrin input data bagi prototaip pemindahan wang, aturcara mesti mencatatkan nilai yang diinput pada satu output yang boleh dilihat oleh pengguna. Jika anda mengaturcara dalam Java, aturcara tersebut boleh mencatat nilai-nilai ke System.out melalui contohnya System.out.println. Jika bahasa pengaturcaraan yang digunakan tidak mempunyai cara yang sepadan dengan System.out dalam Java, aturcara tersebut boleh mencatat nilai-nilai ke satu fail.

Jika pengguna tersebut klik butang Cancel, aturcara skrin input data tersebut harus ditamatkan.

4 PENYAHKOD DATA

Satu aliran data yang berhubung melalui penyambungan bersiri mengandungi data blok dalam format seperti dalam Figure 24.

Length byte 1	Length byte 2	Data bytes

Figure 24 – Format Blok Aliran Data

Dua bait pertama mengandungi bilangan bait data dalam blok. Bait pertama mengandungi 8 bit tertib tinggi dan bait kedua mengandungi 8 bit tertib rendah.

Setiap blok data, kecuali yang terakhir, mengandungi tepat 1022 bait data. Blok terakhir mengandungi antara 0 hingga 1021 bait data. Blok data terakhir menandakan penamatan aliran data.

Tugas anda ialah mengaturcara satu fungsi yang membolehkan satu aplikasi untuk membaca secara wenang sebilangan bait daripada aliran input tanpa memerlukan pengaturan tentang halangan aras rendah.

> Jika anda mengaturcara dalam C atau C++, fungsi anda mesti mempunyai antaramuka aplikasi seperti berikut:
>
> ```
> int SerialRead (void *buf, int len);
> ```
>
> `buf`
>
> merupakan satu penuding kepada satu penimbal yang akan dimuatkan dengan data yang diterima.
>
> `len`
>
> merupakan bilangan bait yang akan dimuatkan ke penimbal.

`return value`

merupakan bilangan bait yang sebenarnya dimuatkan ke dalam penimbal.

Jika anda mengaturcara dalam C atau C++, anda boleh menggunakan fungsi berikut untuk menerima data dari antaramuka bersiri:

```
int SerialRecv (void *buf, int len);
```

`buf`

merupakan satu penuding kepada satu penimbal yang akan dimuatkan dengan data yang diterima.

`len`

merupakan bilangan bait yang akan diterima.

`return value`

merupakan bilangan bait yang sebenarnya diterima.

Jika anda mengaturcara dalam Java, fungsi anda mesti mempunyai antaramuka aplikasi seperti berikut:

```
public class SerialReader {
     // ...
     public static byte[] serialRead (int len);
     // ...
}
```

`len`

merupakan bilangan bait yang akan dikembalikan.

`return value`

merupakan satu tatasusunan bait yang mengandungi bait yang diterima dari aliran input.

Jika anda mengaturcara dalam Java, fungsi anda boleh menggunakan fungsi berikut untuk menerima data dari antaramuka bersiri:

```
public class SerialReceiver {
     // ...
     public static byte[] serialRecv (int len);
     // ...
}
```

`len`

merupakan bilangan bait yang akan diterima.

`return value`

merupakan satu tatasusunan bait yang mengandungi bait yang diterima dari penyambung bersiri.

Jika parameter ‘len’ yang dihantar kepada SerialRead kurang atau bersamaan sifar, SerialRead mesti mengembalikan sifar tanpa membaca sebarang data.

Jika parameter ‘len’ lebih besar daripada sifar, tetapi terdapat lebih daripada ‘len’ bait data tertinggal dalam aliran input, SerialRead mesti mengembalikan tepat ‘len’ bait data.

Jika parameter ‘len’ lebih besar daripada sifar, dan terdapat kurang daripada ‘len’ bait data tertinggal dalam aliran input, SerialRead mesti mengembalikan bilangan bait data tertinggal dalam aliran input.

Panggilan berulang kepada SerialRead apabila tiada bait data tertinggal dalam aliran input mesti mengembalikan sifar.

Fungsi SerialRecv boleh mengembalikan antara 1 dan ‘len’ bait data. SerialRecv yang mengembalikan kurang daripada ‘len’ bait data tidak membawa sebarang makna. Jika SerialRecv dipanggil dan terdapat bait data dalam lingkaran pemanpan input, SerialRecv mengembalikan bait data dalam lingkaran pemanpan input tanpa menunggu. Jika lingkaran pemanpan input tersebut adalah kosong, SerialRecv menunggu sehingga satu atau lebih bait data dimuatkan ke dalam lingkaran pemanpan input dan kemudian mengembalikan bait data tersebut kepada pemanggilnya.

Fungsi SerialRead boleh menggunakan ingatan statik untuk menyimpan data di antara setiap panggilan yang berjaya. Ia tidak perlu dijadikan selamat dalam operasi berbilang ulir.

Untuk keberkesanan yang optimal, SerialRead harus membaca data dari SerialRecv dalam blok 16 aksara.