E-GENTING
PROGRAMMING COMPETITION 2007

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.	Financial Statements	240
2.	The Bankcard Kid	160
3.	Merge and Sort	80
4.	Air Pollution Monitor	320

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 FINANCIAL STATEMENTS

The Trio Trading Company maintains transaction records in an SQL database. The transaction records are maintained in two tables, one that describes the accounts that the transactions affect and another that records individual transactions. The schema entries for the two tables are as follows:

```
create table accounts (
     accId          char(10) not null,
     accDescr       char(60) not null,
     accBalance     double precision not null
);
create table transactions (
     txDate         date not null,
     txNo           integer not null,
     txDescr        char(60) not null,
     txFromAccId    char(10) not null,
     txToAccId      char(10) not null,
     txValue        double precision not null
);
```

`accounts`
: is a table that contains one row for each financial account.

`accId`
: is an account identifier, for example, 'HL334527'. The account identifier uniquely identifies the account.

`accDescr`
: is an account description, for example 'Helical Lamps on Hand' or 'Cash at Bank'.

`accBalance`
: is the current balance of the account in units of cents. For example, if the balance of cash at the bank were $14,782.56, then the value stored in `accBalance` would be 1478256.0.

`transactions`
: is a table that contains one row for each transaction.

`txDate`
: is the date of the transaction.

`txNo`
: is a transaction number. The transaction number uniquely identifies the transaction.

`txDescr`
: is a description of the transaction, for example, 'Purchase of Helical Lamps from Euon Manufacturing'.

`txFromAccId`
: is the account identifier of the account from which the transaction transferred funds (i.e. the account that was credited). For example in the purchase of helical lamps, the from-account might be 'Cash at Bank'.

`txToAccId`
: is the account identifier of the account to which the transaction transferred funds (i.e. the account that was debited). For example in the purchase of helical lamps, the to-account might be 'Helical Lamps on Hand'.

`txValue`
: is the value of the transaction in units of cents. For example, if the purchase of helical lamps cost $3,426.82, then the value of the `txValue` column would be 342682.0.

Your task is to create a flexible reporting program that summarises the balances of Trio Trading's accounts as at some particular date in the past. An example of the required report is presented in Figure 1.

```
22-09-07 11:23     FINANCIAL STATEMENTS              PAGE   1
                      as at 31-08-07

Line
No     Description                                     Balance

0100   Helical Lamps on Hand                          58,267.20
0110   Spiral Lamps on Hand                          142,816.85
0120   Flamingo Lamp Shades on Hand                    5,628.50
0130   Assorted Electrical Fittings on Hand            1,252.10
0180                                                 ----------
0190   Total value of stock on hand                  207,964.65
0200
0210   Cash at Bank, Antopolis Commercial Bank        22,865.12
0220   Cash at Bank, Bank of Trioville                71,102.78
0280                                                 ----------
0290   Total cash at bank                             93,967.90
0300
0310   Trade creditors                               102,781.25
0320   Mortgage on warehouse                          50,215.12
0380                                                 ----------
0390   Total liabilities                             152,996.37
0400                                                 ----------
0410   Net worth                                     148,936.18
0420                                                 ==========
```

Figure 1 – Sample Financial Statements

In order to accomplish the flexibility that Trio Trading needs, the lines in the financial statements are not to be hard-coded, but are to be defined using three database tables. The schema entries that define the three tables are presented below:

```
create table lines (
     lineNo          smallint not null,
     lineDescr       char(40) not null,
     lineType        smallint not null
);
create table elements (
     eleLineNo       smallint not null,
     eleSign         smallint not null,
     eleAccId        char(10) not null
);
create table references (
     refThisLineNo   smallint not null,
     refSign         smallint not null,
     refOtherLineNo smallint not null
);
```

`lines`
: is a table that contains one row for each line on the financial statements.

`lineNo`
: is a line number that uniquely identifies each line in the financial statements. The line number is shown in the left column of the financial statements.

`lineDescr`
: is a line description. The line description is shown in the middle column of the financial statements.

`lineType`
: is a line type code. The line type codes are defined in Table 1.

`elements`
: is a table that contains one row for each account to be added to or subtracted from the balance for the line shown in the right column of the financial statements (the 'line balance').

`eleLineNo`
: is a line number in the same domain as `lines.lineNo` that identifies the report line to which the element relates. The `elements` table may contain several rows with the same `eleLineNo`, each row identifying an account balance to be included in the line balance.

`eleSign`
: is +1 if the account balance is to be added to the line balance or –1 if the account balance is to be subtracted from the line balance.

`eleAccId`
: is the account identifier of the account whose balance is to be included in the line balance.

`references`
: is a table that contains one row for each other line whose balance is to be added to or subtracted from the line balance.

`refThisLineNo`
: is a line number in the same domain as `lines.lineNo` that identifies the report line to which the reference relates. The `references` table may contain several rows with the same `refThisLineNo`, each row identifying another line balance to be included in this line balance.

`refSign`
: is +1 if the other line balance is to be added to this line balance or –1 if the other line balance is to be subtracted from this line balance.

`refOtherLineNo`
: is a line number in the same domain as `lines.lineNo` that identifies the other line balance to be included in this line balance.

Table 1 – Line Type Codes

Line Type Code	Meaning
0	The balance column in the financial statements must be the line balance for the line as defined in Equation 1.
1	The balance column in the financial statements must be blank.
2	The balance column in the financial statements must show a row of minus (‘–’) signs.
3	The balance column in the financial statements must show a row of equals (‘=’) signs.

Line type codes 1, 2 and 3 allow blank lines, single lines and double lines to be included in the report. For example, in Figure 1, the blank line at line number 0200 would have line type 1, the single line at line number 0180 would have line type 2 and the double line at line number 0420 would have line type 3.

Line type 0 includes account balances in the report. The mathematical formula for the line balance to be shown for line type 0 is presented in Equation 1. In layman’s terms, the balance can include backdated account balances and the line balances of other lines. The `elements` table defines the account balances and the `references` table defines the line balances of other lines. The `eleSign` and `refSign` columns indicate whether the balance is to be added to or subtracted from the other components of the line balance.

Equation 1 – Formula for Balance of Line Type 0

$$\text{lineBal}_i = \sum_{j=1}^{n} \text{eleSign}_j * \text{backBal}_j + \sum_{k=1}^{m} \text{refSign}_k * \text{lineBal}_k$$

Where:

lineBal_i	is the line balance of the i-th line.
n	is the number of `elements` table rows with `eleLineNo` equal to the line number of the i-th line.
eleSign_j	is the value of `eleSign` in the j-th `elements` table row with `eleLineNo` equal to the line number of the i-th line.
backBal_j	is the back-dated balance of the account identified by `eleAccountId` in the j-th `elements` table row with eleLineNo equal to the line number of the i-th line.
m	is the number of `references` table rows with a `refLineNo` equal to the line number of the i-th line.
refSign_k	is the value of `refSign` the k-th `references` table row with refThisLineNo equal to the line number of the i-th line.
lineBal_k	is the line balance of the other line identified by `refOtherLineNo` in the k-th references table row with `refThisLineNo` equal to the line number of the i-th line.

The reporting program must accept a single parameter, the date of the report. The reporting program must roll back the effect of the transactions that occurred after the date of the report to calculate the backdated account balances used to calculate the line balances shown on the report.

The reporting program should endeavour to identify self-referencing `references` table entries. For example if line 0210 was the sum of lines 0220 and 0230, and line 0230 was the sum of lines 0210 and 0240, the formula for the evaluation of lines 0210 and 0230 could potentially recur indefinitely, because line 0210 includes the value of line 0230 and line 0230 includes the value of line 0210. The financial statements reporting program should endeavour to identify such situations and send an appropriate message to the user instead of recurring indefinitely.

The financial statements must display the following information:

1. date and time the report was generated;
2. page number at the top of each page;
3. date of the report;
4. for each line defined in the lines table:

a. line number;
b. line description;
c. balance column depending on the line type as defined in Table 1.

The lines in the financial statements must be shown in line number order.

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

The financial statements should be laid out in accordance with the layout in Figure 2.

```
         1         2         3         4         5         6
1...5....0....5....0....5....0....5....0....5....0....5....0
DD-MM-YY HH:MM     FINANCIAL STATEMENTS              PAGE X-X
                      as at XX-XX-XX

Line
No     Description                                    Balance

XXXX   X--------------------------------------X  XXX,XXX.XX
XXXX   X--------------------------------------X  XXX,XXX.XX
XXXX   X--------------------------------------X  ----------
XXXX   X--------------------------------------X  XXX,XXX.XX
XXXX   X--------------------------------------X  ==========
```

Figure 2 – Financial Statements Layout

2 THE BANKCARD KID

A British entrepreneur by the name of Peter Burns is endeavouring to do for the retail banking industry, what various others have done for the airline industry. He has figured out a low-cost model for retail banking services such as savings accounts, payroll distribution, credit cards and cash dispensation that extinguishes bank fees and significantly increases the interest paid to savings account holders.

The problem is that his touch-screen ATMs, which he had programmed cheaply by an overseas software house, occasionally and unpredictably throw up a message that says 'atmruntime.exe has encountered a problem and needs to close. We are sorry for the inconvenience…'. While the fault has not yet caused an account to be debited without paying out the correct amount of cash, it happens with sufficient frequency for a daily tabloid to have published the headline 'Burnt Again'.

The problem has now become a focus point for the full-service banks to attack Burns's low-cost alternative. They are planning marketing campaigns based on the slogan 'who wants to get Burnt?'

Despite significant efforts on the part of Burns's in-house programming team to identify the cause of the problem, the efforts have been to no avail because it is almost impossible to exercise the fault by operating the ATM in a testing environment. What the software engineers think is needed is a program they call the 'Bankcard Kid' that emulates

intensive use of the ATM so that they can exercise the program overnight. Your task is to write the Bankcard Kid.

The Bankcard Kid must execute the following cycle:

1. repeat until terminated:
    a. randomly select a card from the available alternatives;
    b. emulate the insertion of the card into the ATM;
    c. wait for between 3 and 7 seconds;
    d. check that the card was accepted;
    e. emulate the entry of a PIN, waiting for between 0.05 and 1.0 seconds between each emulated touch of the screen;
    f. wait for between 1.5 and 3 seconds;
    g. check that the PIN was accepted;
    h. pseudo-randomly select an account (cheque, savings or credit);
    i. wait for between 1.5 and 3 seconds;
    j. check that the account selection was accepted;
    k. emulate entry of an amount to be withdrawn, waiting for between 0.05 and 1.0 seconds between each emulated touch of the screen;
    l. wait for 5.0 seconds;
    m. check that the correct amount is paid out;

In the above pseudo-code, the expression 'wait for between A and B seconds' means pseudo-randomly select a waiting time in milliseconds between limits A and B inclusive and then wait for the pseudo-randomly selected waiting time.

The ATM infrastructure and device drivers provide the following functions that can be called from C or C++:

```
void EmulateCardIn (const char *cardNumber);
                    // Emulate the insertion of a card
                    // into the card reader
void ReadScreenPixels (long *pixelArray);
                    // Read the pixels shown on the
                    // screen into pixelArray
void ReadScreenDump (const char *fileName,
     long *pixelArray);
                    // Read pixels from a screen dump
                    // file
void EmulateTouch (int x, int y);
                    // Emulate a user touching the
                    // touch screen at co-ordinates
                    // (x, y)
int GetAmountPaid ();
                    // Get the amount paid out by
                    // the bill dispenser
```

`cardNumber`
is a pointer to a null-terminated character string containing a card number.

`pixelArray`
: is a pointer to an array to be loaded with the RGB colour components of each pixel on the ATM screen. `pixelArray` must have a capacity of at least 307,200 32-bit integers.

`fileName`
: is a pointer to a null-terminated character string containing the name of a screen dump file from which the RGB colour components of a previous screen image are to be extracted.

`x`
: is the distance from the left edge of the screen to the point on the screen where the touching of the screen is to be emulated.

`y`
: is the distance from the top edge of the screen to the point on the screen where the touching of the screen is to be emulated.

`return value of GetAmountPaid`
: is the amount paid out by the bill dispenser since the last call to `GetAmountPaid` in whole dollars. For example, if the bill dispenser had paid out $50 since the last time `GetAmountPaid` was called, then `GetAmountPaid` would return 50.

The ATM infrastructure also provides the following equivalent functions that can be called from Java:

```
public class ATMInfra {
     //...
     static public void emulateCardIn (
          String cardNumber) {/*...*/}
                    // Emulate the insertion of a card
                    // into the card reader
     static public int [] readScreenPixels ()
          {/*...*/}
                    // Read the pixels shown on the
                    // screen into pixelArray
     static public int [] readScreenDump (
          const char *fileName) {/*...*/}
                    // Read pixels from a screen dump
                    // file
     static public void emulateTouch (
          int x, int y) {/*...*/}
                    // Emulate a user touching the
                    // touch screen at co-ordinates
                    // (x, y)
```

```
        static public int getAmountPaid ()
             {/*...*/}
                        // Get the amount paid out by
                        // the bill dispenser
        //...
    }
```

`cardNumber`
: is a string containing a card number.

`return value of readScreenPixels`
: is a reference to an array of integers that contains the RGB colour components of the pixels on the screen.

`fileName`
: is a string containing the name of a screen dump file from which the RGB colour components of a previous screen image are to be extracted.

`return value of readScreenDump`
: is a reference to an array of integers that contains the RGB colour components of the pixels in the screen dump file.

`return value of getAmountPaid`
: is the amount paid out by the bill dispenser since the last call to `getAmountPaid` in whole dollars. For example, if the bill dispenser had paid out $50 since the last time `GetAmountPaid` was called, then `GetAmountPaid` would return 50..

Burns's software engineers have approximately 50 sample card numbers and PINs in a file called 'TestCards.txt'. The card numbers are 16 digits long and the PINs are 6 digits long. Each card number-PIN pair is stored on a separate line, with the card number and PIN separated by a single space character. The first 10 lines of TestCards.txt are shown in Figure 3.

```
0856669126311954 212384
4996552050476648 806909
9764925646670808 089694
7570032704671097 693010
1516410663483838 565200
6611851186922681 140063
0969351051899105 097351
4928296487649896 272250
6505872582625350 543119
2906025111421829 668369
```

Figure 3 – TestCards.txt

The Bankcard Kid must pseudo-randomly select a new card number-PIN pair from the data in TestCards.txt for each test cycle.

Each of the test accounts have been pre-loaded with a large amount of money that should never be exhausted in a normal run of the Bankcard Kid.

To test whether or not a card was accepted, a PIN was accepted or an account selection was accepted, the Bankcard Kid must compare the display shown on the screen with a screen dump taken from a successful run. Sample screen dumps from previous successful runs are stored in the files listed in Table 2.

Table 2 – Screen Dump File Names

Screen Dump File Name	Contents
AcceptedCard.rgb	A screen dump of the display after a card is successfully inserted.
AcceptedPIN.rgb	A screen dump of the display after a PIN is successfully keyed in.
AcceptedAccount.rgb	A screen dump of the display after an account has been successfully selected.

Each screen dump file contains exactly 307,200 32-bit words, which contain the RGB colour components of each pixel on the display. The Bankcard Kid should use the `ReadScreenDump` function to extract the 32-bit words from the screen dump files. Similarly, the `ReadScreenPixels` function also returns exactly 307,200 32-bit words.

Because a cursor flashes on the screen, the screen pixels of a successful operation may not exactly match the contents of the corresponding screen dump file. The Bankcard Kid should treat the arrays returned by `ReadScreenDump` and `ReadScreenPixels` as matching if no more than 200 pixels are different.

The Bankcard Kid must emulate entry of a PIN by using the `EmulateTouch` function to emulate the user touching the touch-screen keypad. The ATM automatically processes the PIN when all six digits are entered. The Bankcard Kid must wait for a pseudo-random period of between 0.05 and 1.0 seconds between clicking each button. The layout of the PIN entry screen is presented in Figure 4. The dimensions are in pixel units that can be passed directly to `EmulateTouch`.

The screen for selecting an account is presented in Figure 5. The Bankcard Kid must pseudo-randomly select the cheque, savings or credit account.

The screen for entering the amount to be withdrawn is presented in Figure 6. The Bankcard Kid must enter a pseudo-randomly selected amount between $10 and $1000. The amount must be a multiple of $10. The Bankcard Kid must enter the whole dollar amount followed by a decimal point and then two zero digits and then the OK button. The Bankcard Kid must wait for a pseudo-random period of between 0.05 and 1.0 seconds between clicking each button.

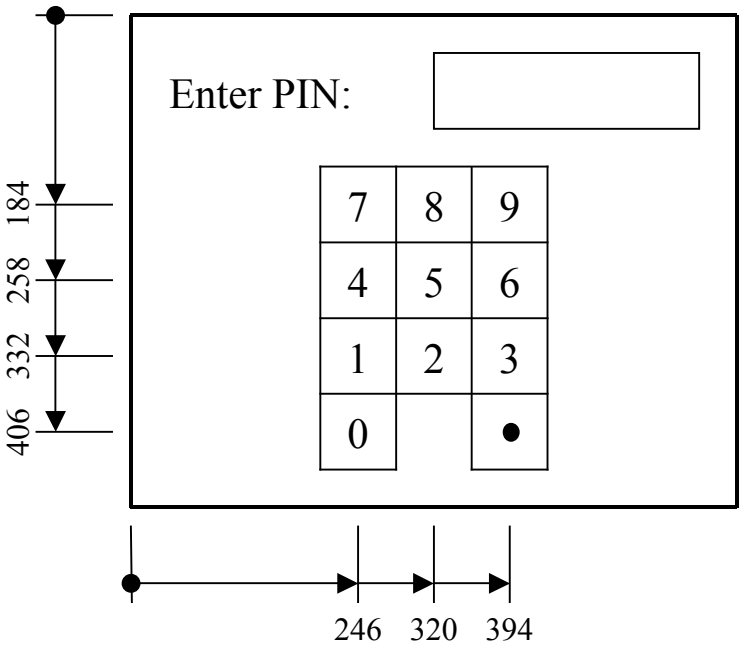

Figure 4 – PIN Screen Layout

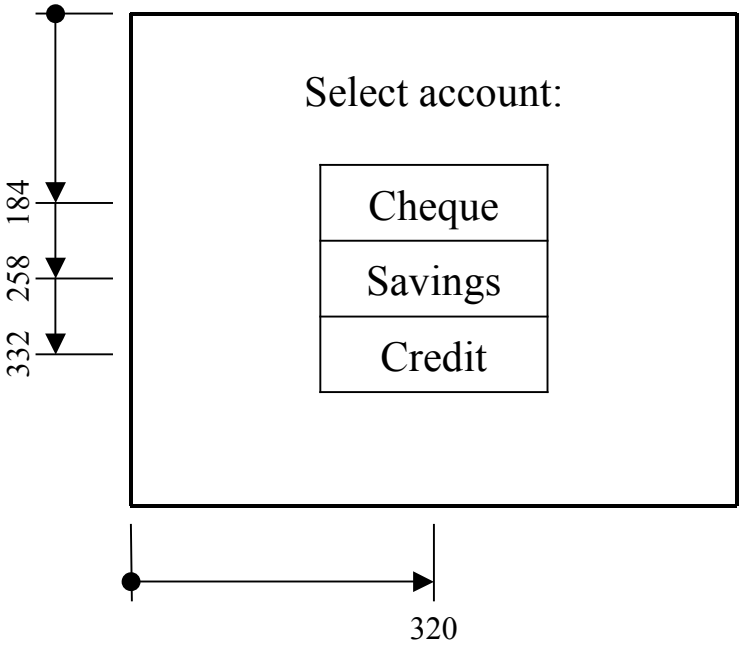

Figure 5 – Account Selection Screen Layout

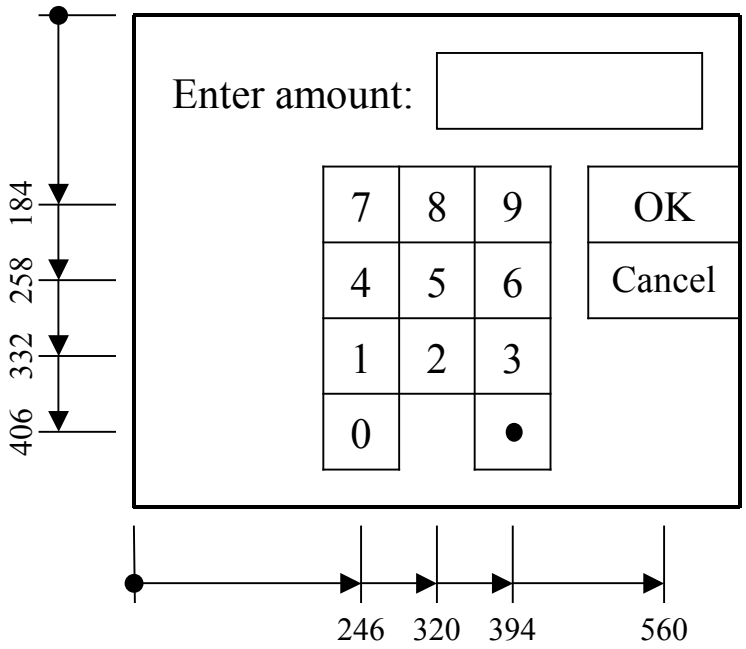

Figure 6 – Amount Screen Layout

The ATM dispenses approximately five banknotes per second, so it may take several seconds to dispense a large amount of money. After keying in the amount to be withdrawn, the Bankcard Kid must wait for five seconds and then call `GetAmountPaid` once per second accumulating the amount dispensed until `GetAmountPaid` returns zero. When `GetAmountPaid` returns zero the dispensing function has finished and the accumulated amount is the total amount dispensed.

The Bankcard Kid must check that the correct amount is dispensed.

When the Bankcard Kid is first started, it must call GetAmountPaid to reset any left over amount from any previous tests.

If the Bankcard Kid detects a fault in the operation of the ATM it must write a message that describes the nature of the fault to standard output and then exit.

3 MERGE AND SORT

E-Genting's source code control systems generate source file change listings in the form shown in Figure 7. Each listing contains changes to multiple source files. The changes to each source file start with a header line. The header line starts with a series of 24 equals signs ('=') followed by a space, a file name, then a space, a change version ('37.2' in Figure 7) and then another space and a series of 24 equals signs.

During the testing phase of the software development life cycle, several files of change listings of the type shown in Figure 7 might be handed to a tester for testing. The files might contain multiple changes to the same source file, each with a different version.

```
======================== drs/GuiComp.cpp 37.2 ========================
14a15
> // 24-08-07   JS      Changed 'comp issue loc date' to 'shift date'
145c146
<       frm.FedText (COL_WIDTH*2, l, "Comp issue loc date");
---
>       frm.FedText (COL_WIDTH*2, l, "Shift date");
======================== drs/GuiIncd.cpp 37.2 ========================
14a15
> // 24-08-07   JS      Changed 'comp issue loc date' to 'shift date'
146c147
<       frm.FedText (COL_WIDTH*2, l, "Comp issue loc date");
---
>       frm.FedText (COL_WIDTH*2, l, "Shift date");
```

Figure 7 – Source File Change Listing

The testing team has requested a program that merges the changes from multiple change listings into a single output, with the changes ordered by file name and then version.

Your task is to write a program that accepts a list of file names from a command line, each identifying a file containing a change listing of the type shown in Figure 7, sorts the changes in the change listings by file name and then version and then emits the changes, in the same format as the input files to the program's standard output.

In the change listings, a line beginning with an equals sign identifies a header line. The source file changes themselves never contain lines that start with an equals sign.

File names do not contain embedded blanks.

In comparing the version numbers, the sorting function should treat the version as two separate numbers separated by a decimal point, a release number and a level number. It should sort changes to the same file by release and then level. For example, version 37.5 should come before version 37.12.

4 AIR POLLUTION MONITOR

In an effort to make the public and industry more conscious of the causes of air pollution, the Department of Environment has installed a large number of air pollution sensors in and around the Klang Valley. These sensors automatically transmit information on their location, obtained from GPS receivers inside the sensors, along with the value of the air pollution index (API) at the location, to a central server every 10 minutes.

Experience has shown that this raw information is difficult to interpret. What the Department of Environment would like is an on-line dynamic map that can be accessed over the Internet that shows areas of higher and lower air pollution in different colours, like hypsometric tints on a topographical map.

The raw data from the air pollution sensors can be obtained by downloading a text file over the Internet. For testing purposes, a mirror site has been set up at http://genting.com.my/rnd/airdata.txt. A sample of the data in airdata.txt is shown in Figure 8. The format of each line of airdata.txt is shown in Figure 9.

```
E10126033N00259537194
E10149349N00255301342
E10147289N00252330231
E10128254N00258029010
E10134282N00313547133
E10146512N00306312421
E10133274N00303149145
E10129093N00300315194
E10123597N00311052107
E10135371N00311513370
          :
          :
```

Figure 8 – Sample of Air Pollution Data in airdata.txt

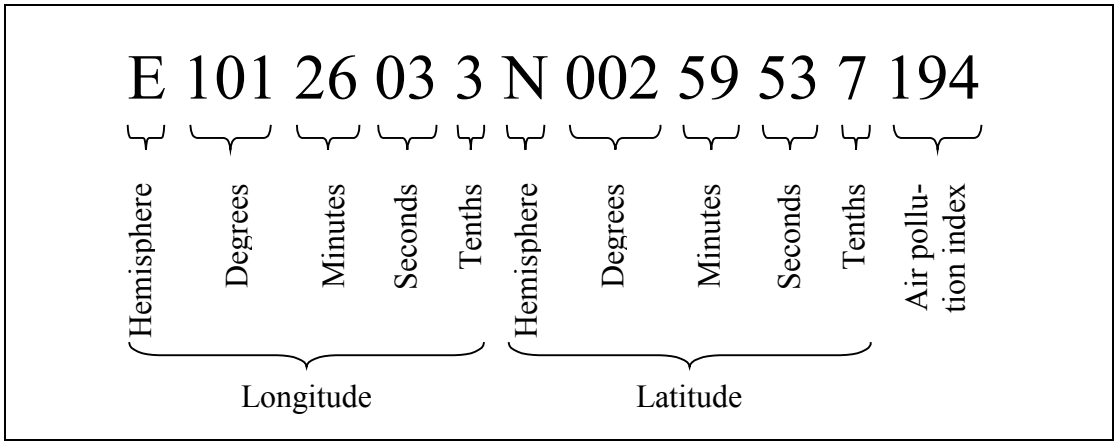

Figure 9 – Format of airdata.txt lines

The longitude hemisphere is either 'E' for east or 'W' for west. The latitude hemisphere is either 'N' for north or 'S' for south.

All the co-ordinates in airdata.txt lie between longitudes 101° 20' and 101° 50' east and latitudes 002° 50' and 003° 20' north.

The Klang Valley is sufficiently close to the equator for longitudes and latitudes to be treated as simple x-y co-ordinates without visible distortion.

The Department of Environment has approximately 500 air pollution sensors scattered around the Klang Valley, though the number changes from time to time as sensors are taken out of service for maintenance and then put back into service, perhaps in a different location.

Your task is to create a client program that can be downloaded over the Internet, as say a Java applet, that renders an air pollution map that displays areas with different levels of air pollution in different colours. It is not a requirement that the client be written in Java. You are free to write the client in any language that allows the program to be downloaded over the Internet and executed in a conventional web browser. An example of the required output is shown in Figure 10.

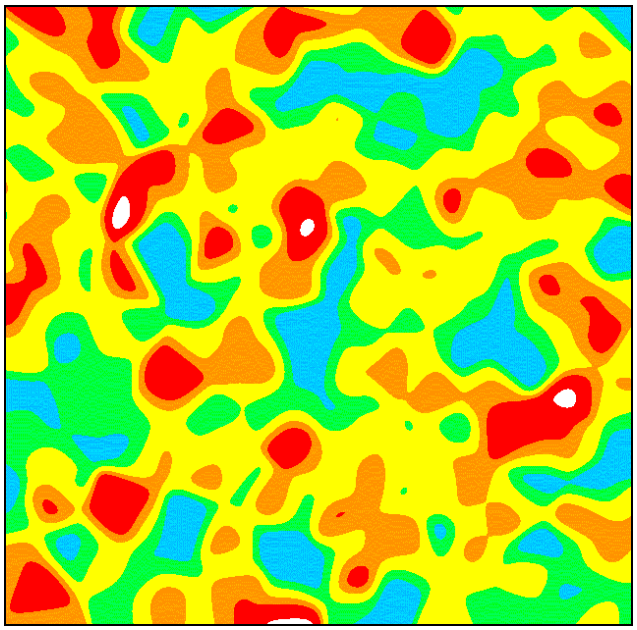

Figure 10 – Example Air Pollution Map

Air pollution levels in different areas must be colour-coded in accordance with the colours in Table 3.

Table 3 – Air Pollution Index Colour Coding

API	Status	RGB Colour Components		
		Red	Green	Blue
0-50	Good	0	204	255
51-100	Moderate	0	255	64
101-200	Unhealthy	255	255	0
201-300	Very unhealthy	255	153	0
301-500	Hazardous	255	0	0
Above 500	Emergency	255	255	255

The air pollution map should be rendered at a scale of approximately 1:330,000. For your reference, a minute of longitude at the equator (a nautical mile) is 1852 metres or 72913 inches.

Equation 2 provides a means for estimating the API at any point (x, y) within the limits of the co-ordinates in airdata.txt. The client program should use Equation 2 or an algorithm that closely approximates it to determine the colour to be shown by each pixel in the air pollution map.

Equation 2 – Interpolating the Air Pollution Index

$$api(x,y) = \frac{\sum_{i=1}^{n} ad_i \, w(x,y,x_i,y_i)}{\sum_{i=1}^{n} w(x,y,x_i,y_i)}$$

Where:

api(x,y)	is the estimated API at point (x,y).
x	is a longitude in minutes.
y	is a latitude in minutes
n	is the number of sensors.
ad_i	is the API reported by sensor i in airdata.txt.
xi	is the longitude in minutes of sensor i.
yi	is the latitude in minutes of sensor i.
$w(x,y.x_i,y_i)$	is the weighting function defined below.

Equation 3 – Weighting Function

$$w(x,y.x_i,y_i) = e^{-\left(\frac{((x_i - x)^2 + (y_i - y)^2)}{A^2}\right)}$$

Where:

$w(x,y.x_i,y_i)$	is the value of the weighting function.
x	is a longitude in minutes.
y	is a latitude in minutes
x_i	is the longitude in minutes of sensor i.
y_i	is the latitude in minutes of sensor i.
e	is Euler's number (approximately 2.7128)
A	is the constant 0.9 nautical miles.

Sensors with weighting function values less than 10^{-6} may be disregarded when calculating the API at any particular point (x,y).

Because the sensors are sending updated information to the server every 10 minutes, on average at least one line in airdata.txt changes every 1.2 seconds. For this reason, the client should obtain another copy of airdata.txt from the server and refresh the display every 30 seconds. To satisfy this requirement, the client program should be able to convert the sensor results into an air pollution map in less than 30 seconds. A conversion time of less than 5 seconds would be preferred. The client program should show the old air pollution map while it is generating a new air pollution map.

Previous attempts to program the air pollution monitor have shown that a naïve implementation that evaluates the weighting function for every sensor for every pixel takes approximately 51 seconds to generate a new image.

The credit for this question will be divided into two halves: half for making the air pollution monitor work at all and the other half for making it accomplish its performance goals.

PERTANDINGAN PENGATURCARAAN E-GENTING 2007

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. Pertandingan ini membenarkan peserta membuat rujukan dari mana-mana 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.	Penyata Kewangan	240
2.	Pengolok Kad Bank	160
3.	Gabung dan Susun	80
4.	Pemantau Pencemaran Udara	320

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 (semestinya) menjawab semua soalan.

---

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

5 PENYATA KEWANGAN

Syarikat Perdagangan Trio mengekalkan rekod-rekod urus niaga dalam satu pangkalan data SQL. Rekod-rekod urus niaga dikekalkan dalam dua jadual, satu jadual yang menyimpan akaun-akaun yang dipengaruhi oleh urus niaga, dan satu lagi jadual yang mencatat urus niaga individu. Skema untuk kedua-dua jadual tersebut adalah seperti berikut:

```
create table accounts (
     accId          char(10) not null,
     accDescr       char(60) not null,
     accBalance     double precision not null
);
create table transactions (
     txDate         date not null,
     txNo           integer not null,
     txDescr        char(60) not null,
     txFromAccId    char(10) not null,
     txToAccId      char(10) not null,
     txValue        double precision not null
);
```

`accounts`
: adalah satu jadual yang mengandungi satu baris untuk setiap akaun kewangan.

`accId`
: adalah satu pengecam akaun, contohnya 'HL334527'. Pengecam akaun mengenal pasti setiap akaun secara unik.

`accDescr`
: adalah huraian akaun, contohnya 'Helical Lamps on Hand' or 'Cash at Bank'.

`accBalance`
: adalah imbangan semasa untuk akaun tersebut dalam unit sen. Contohnya, jika imbangan tunai dalam bank ialah $14,782.56, maka nilai yang disimpan dalam `accBalance` ialah 1478256.0.

`transactions`
: adalah satu jadual yang mengandungi satu baris untuk setiap urusniaga.

`txDate`
: adalah tarikh urus niaga.

`txNo`
: adalah satu nombor urus niaga. Nombor urus niaga mengenal pasti sesuatu urus niaga secara unik.

`txDescr`
: adalah satu huraian urus niaga, contohnya, 'Purchase of Helical Lamps from Euon Manufacturing'.

`txFromAccId`

adalah pengecam akaun untuk akaun di mana urus niaga memindah wang daripadanya (iaitu akaun yang dikreditkan). Contohnya dalam pembelian helical lamps, akaun itu boleh merupakan 'Cash at Bank'.

`txToAccId`

adalah pengecam akaun untuk akaun di mana urus niaga memindah wang kepadanya (iaitu akaun yang didebitkan). Contohnya dalam pembelian helical lamps, akaun itu boleh merupakan 'Helical Lamps on Hand'.

`txValue`

adalah nilai urus niaga dalam unit sen. Contohnya jika pembelian helical lamps berharga $3,426.82, maka nilai lajur `txValue` merupakan 342682.0.

Tugas anda ialah menghasilkan satu aturcara yang menjana laporan secara fleksibel. Laporan tersebut merumuskan imbangan bagi akaun-akaun Syarikat Perdagangan Trio pada sesuatu tarikh yang lepas. Contoh laporan tersebut adalah seperti dalam Figure 1.

```
22-09-07 11:23     FINANCIAL STATEMENTS              PAGE   1
                      as at 31-08-07

Line
No     Description                                    Balance

0100   Helical Lamps on Hand                          58,267.20
0110   Spiral Lamps on Hand                          142,816.85
0120   Flamingo Lamp Shades on Hand                    5,628.50
0130   Assorted Electrical Fittings on Hand            1,252.10
0180                                                 ----------
0190   Total value of stock on hand                  207,964.65
0200
0210   Cash at Bank, Antopolis Commercial Bank        22,865.12
0220   Cash at Bank, Bank of Trioville                71,102.78
0280                                                 ----------
0290   Total cash at bank                             93,967.90
0300
0310   Trade creditors                               102,781.25
0320   Mortgage on warehouse                          50,215.12
0380                                                 ----------
0390   Total liabilities                             152,996.37
0400                                                 ----------
0410   Net worth                                     148,936.18
0420                                                 ==========
```

Figure 11 – Contoh Penyata Kewangan

Untuk memenuhi keperluan Syarikat Perdagangan Trio, baris-baris dalam penyata kewangan tidak boleh diaturcara secara terus, tetapi perlu ditakrif dengan menggunakan

tiga jadual pangkalan data. Skema yang menakrifkan ketiga-tiga jadual tersebut adalah seperti berikut:

```
create table lines (
     lineNo         smallint not null,
     lineDescr      char(40) not null,
     lineType       smallint not null
);
create table elements (
     eleLineNo      smallint not null,
     eleSign        smallint not null,
     eleAccId       char(10) not null
);
create table references (
     refThisLineNo  smallint not null,
     refSign        smallint not null,
     refOtherLineNo smallint not null
);
```

`lines`
: adalah satu jadual yang mengandungi satu barisan bagi setiap baris dalam penyata kewangan.

`lineNo`
: adalah satu nombor barisan yang mengenal pasti setiap barisan dalam penyata kewangan. Nombor barisan dipaparkan pada lajur kiri penyata kewangan.

`lineDescr`
: adalah satu huraian kepada barisan tersebut. Huraian tersebut dipaparkan pada lajur tengah penyata kewangan.

`lineType`
: adalah satu kod untuk jenis barisan. Kod-kod untuk jenis barisan ditakrif dalam Table 1.

`elements`
: adalah satu jadual yang mengandungi satu baris untuk setiap akaun yang akan ditambah kepada atau ditolak daripada imbangan bagi barisan tersebut yang terletak di lajur kanan dalam penyata kewangan (imbangan barisan).

`eleLineNo`
: adalah satu nombor barisan yang terletak dalam domain yang sama dengan `lines.lineNo` dan mengenal pasti barisan laporan di mana elemen itu dikaitkan. Jadual `elements` boleh mengandungi beberapa baris dengan `eleLineNo` yang sama, dengan setiap baris mengenal pasti satu imbangan akaun yang akan disertakan dalam imbangan barisan.

`eleSign`
: adalah +1 jika imbangan akaun akan ditambah kepada imbangan barisan atau –1 jika imbangan akaun akan ditolak daripada imbangan barisan.

`eleAccId`

adalah pengecam akaun bagi akaun di mana nilai imbangan akaunnya akan disertakan dalam imbangan barisan.

`references`

adalah satu jadual yang mengandungi satu barisan bagi setiap baris lain yang di mana imbangannya perlu ditambah kepada atau ditolak daripada imbangan barisan.

`refThisLineNo`

adalah satu nombor barisan yang terletak dalam domain yang sama seperti `lines.lineNo` yang mengenal pasti barisan laporan yang dirujukkan. Jadual `references` boleh mengandungi beberapa baris dengan `refThisLineNo` yang sama, dengan setiap baris mengenal pasti satu imbangan barisan lain yang akan disertakan dalam imbangan barisan ini.

`refSign`

adalah +1 jika imbangan barisan lain akan ditambah kepada imbangan barisan ini atau –1 jika imbangan barisan lain akan ditolak daripada imbangan barisan ini.

`refOtherLineNo`

adalah satu nombor barisan yang terletak dalam domain yang sama seperti `lines.lineNo` yang mengenal pasti imbangan barisan lain yang akan disertakan dalam imbangan barisan ini.

Table 4 – Kod-kod Jenis Barisan

Kod Jenis Barisan	Makna
0	Lajur imbangan dalam penyata kewangan mestilah merupakan imbangan barisan bagi barisan yang ditakrif seperti dalam Equation 1.
1	Lajur imbangan dalam penyata kewangan mestilah dikosongkan.
2	Lajur imbangan dalam penyata kewangan mesti memaparkan satu baris tanda tolak (‘-‘).
3	Lajur imbangan dalam penyata kewangan mesti memaparkan satu baris tanda sama dengan (‘=).

Jenis barisan dengan kod 1, 2 dan 3 membenarkan barisan kosong, barisan dengan satu garis dan barisan dengan dua garis disertakan dalam laporan. Contohnya, dalam Figure 1, jenis barisan untuk barisan kosong yang terletak di nombor barisan 0200 adalah 1, jenis barisan untuk barisan dengan satu garisan yang terletak di nombor barisan 0180 adalah 2 dan jenis barisan untuk barisan dengan dua garisan yang terletak di nombor barisan 0420 adalah 3.

Barisan jenis 0 menyertakan imbangan akaun-akaun dalam laporan. Untuk barisan jenis 0, formula matematik bagi imbangan barisan yang akan dipaparkan adalah seperti dihurai

dalam Equation 1. Dengan kata lain, imbangan tersebut boleh berserta dengan imbangan akaun-akaun bertarikh terkebelakang dan imbangan barisan dari barisan-barisan lain. Jadual `elements` menakrifkan imbangan akaun-akaun dan jadual `references` menakrifkan imbangan barisan-barisan untuk barisan yang lain. Lajur `eleSign` dan `refSign` menunjukkan sama ada imbangan tersebut perlu ditambah kepada atau ditolak daripada komponen-komponen lain dalam imbangan barisan.

Equation 4 – Formula untuk Imbangan bagi Barisan Jenis 0

$$lineBal_i = \sum_{j=1}^{n} eleSign_j * backBal_j + \sum_{k=1}^{m} refSign_k * lineBal_k$$

Di mana:

$lineBal_i$	adalah imbangan barisan bagi barisan - i.
n	adalah bilangan baris dalam jadual `elements` dengan `eleLineNo` bersamaan dengan nombor barisan bagi barisan - i.
$eleSign_j$	adalah nilai `eleSign` baris ke-j dalam jadual `elements` dengan `eleLineNo` bersamaan dengan nombor barisan bagi barisan - i.
$backBal_j$	adalah imbangan terkebelakang bagi akaun yang dikenal pasti oleh `eleAccountId` pada baris - j dalam jadual `elements` dengan eleLineNo bersamaan dengan nombor barisan bagi barisan - i.
m	adalah bilangan baris dalam jadual `references` dengan `refLineNo` bersamaan dengan nombor barisan bagi barisan - i.
$refSign_k$	adalah nilai `refSign` bagi baris ke-k dalam jadual `references` dengan refThisLineNo bersamaan dengan nombor barisan bagi barisan - i.
$lineBal_k$	adalah imbangan barisan untuk barisan lain yang dikenal pasti melalui `refOtherLineNo` dalam baris - k dalam jadual `references` dengan `refThisLineNo` bersamaan dengan nombor barisan bagi barisan - i.

Aturcara penjanaan laporan tersebut mesti menerima satu parameter, iaitu tarikh laporan. Aturcara tersebut mesti mengabaikan kesan yang dihasilkan oleh urus niaga yang berlaku selepas tarikh laporan untuk menghitung imbangan akaun ter dahulu yang kemudiannya digunakan untuk menghitung imbangan barisan yang dipapar dalam laporan.

Aturcara penjanaan laporan tersebut harus cuba mengenal pasti rujukan balik kepada diri sendiri dalam jadual `references`. Contohnya, jika barisan 0210 adalah jumlah bagi barisan 0220 dan 0230, dan barisan 0230 adalah jumlah bagi barisan 0210 dan 0240,

maka formula untuk penghitungan barisan 0210 dan 0230 akan berulang tanpa berhenti, sebab barisan 0210 menyertakan nilai barisan 0230 dan barisan 0230 menyertakan nilai barisan 0210. Aturcara penjanaan laporan kewangan tersebut harus cuba mengenal pasti situasi sebegini dan menghantar mesej yang sesuai kepada pengguna dan bukannya berulang tanpa berhenti.

Penyata kewangan tersebut mesti memaparkan maklumat berikut:

1. tarikh dan masa laporan tersebut dijana;
2. nombor halaman di bahagian atas setiap halaman;
3. tarikh laporan;
4. bagi setiap barisan yang ditakrif dalam jadual `lines`:
    a. nombor barisan;
    b. huraian barisan;
    c. lajur imbangan bergantung kepada jenis barisan seperti ditakrif dalam Table 1.

Barisan-barisan dalam penyata kewangan mesti dipaparkan mengikut susunan nombor barisan.

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

Penyata kewangan tersebut harus dipapar seperti dalam Figure 2.

```
          1         2         3         4         5         6
1...5....0....5....0....5....0....5....0....5....0....5....0
DD-MM-YY HH:MM     FINANCIAL STATEMENTS              PAGE X-X
                       as at XX-XX-XX

Line
No     Description                                    Balance

XXXX   X--------------------------------------X  XXX,XXX.XX
XXXX   X--------------------------------------X  XXX,XXX.XX
XXXX   X--------------------------------------X  ----------
XXXX   X--------------------------------------X  XXX,XXX.XX
XXXX   X--------------------------------------X  ==========
```

Figure 12 – Paparan Penyata Kewangan

6 PENGOLOK KAD BANK

Seorang usahawan British bernama Peter Burns cuba melakukan apa yang telah dilakukan oleh industri penerbangan ke atas industri perbankan. Beliau telah memikirkan satu model kos rendah untuk perkhidmatan perbankan seperti akaun simpanan, pengagihan gaji, kad kredit, dan pengeluaran tunai yang memansuhkan yuran bank dan juga meningkatkan bayaran faedah kepada pemegang akaun simpanan secara ketara.

Masalahnya sekarang ialah ATM sentuh skrinnya, yang diaturcarakan secara murah oleh sebuah syarikat perisian luar negara, sering kali dan juga di luar jangkaannya

memaparkan satu mesej yang menyatakan bahawa ‘atmruntime.exe menemui satu masalah dan terpaksa ditutup. Kami meminta maaf atas kesulitan…’. Walaupun masalah itu belum menyebabkan sesuatu akaun didebitkan tanpa membayar amaun tunai yang betul, kekerapan ia berlaku memadai untuk sesebuah tabloid harian mengumumkannya sebagai berita utama yang bertajuk ‘Burnt Again’.

Masalah ini telah menjadi titik fokus utama yang digunakan oleh bank-bank biasa untuk menyerang alternatif kos rendah Burn. Mereka merancang kempen pemasaran yang menggunakan slogan ‘who wants to get Burnt?’.

Walaupun pasukan pengaturcaraan Burn telah berusaha untuk mengenal pasti sebab kepada masalah itu, usaha itu tidak berhasil kerana dengan ATM yang beroperasi dalam tempoh pengujian , adalah hampir tidak mungkin untuk mengenal pasti kecacatan itu. Jurutera-jurutera perisian berpendapat bahawa apa yang diperlukan ialah satu aturcara yang mereka menggelarnya sebagai pengolok kad bank yang meniru penggunaan ATM tersebut secara intensif supaya mereka boleh melaksanakan aturcara tersebut semalaman. Tugas anda adalah mengaturcarakan pengolok kad bank tersebut.

Pengolok kad bank tersebut mesti melaksanakan kitar seperti berikut:

1. ulang sehingga dihentikan:
    a. memilih satu kad secara rawak daripada alternatif-alternatif yang sedia ada;
    b. meniru aksi pemasukan kad ke dalam ATM;
    c. tunggu di antara 3 hingga 7 saat;
    d. semak bahawa kad telah diterima;
    e. meniru kemasukan nombor PIN, dengan menunggu di antara 0.05 dan 1.0 saat bagi setiap aksi peniruan yang menyentuh skrin;
    f. tunggu di antara 1.5 hingga 3 saat;
    g. semak bahawa PIN telah diterima;
    h. pilih satu akaun (cek, simpanan atau kredit) secara pseudo-rawak;
    i. tunggu di antara 1.5 hingga 3 saat;
    j. semak bahawa pilihan akaun telah diterima;
    k. meniru aksi input sesuatu amaun yang ingin dikeluarkan, dengan menunggu di antara 0.05 hingga 1.0 saat bagi setiap aksi peniruan yang menyentuh skrin;
    l. tunggu selama 5.0 saat;
    m. semak bahawa amaun yang betul telah dibayar.

Dalam kod-pseudo di atas, ungkapan ‘tunggu di antara A hingga B saat’bermakna pilih satu masa menunggu secara pseudo-rawak dalam milisaat di antara had A dan B (termasuk A dan B) dan kemudian menunggu selama masa yang telah dipilih tersebut.

Infrastruktur ATM dan pemacu alatan membekalkan fungsi-fungsi berikut yang boleh dipanggil dari C atau C++:

```
     void EmulateCardIn (const char *cardNumber);
                         // Emulate the insertion of a card
                         // into the card reader
     void ReadScreenPixels (long *pixelArray);
                         // Read the pixels shown on the
                         // screen into pixelArray
     void ReadScreenDump (const char *fileName,
          long *pixelArray);
                         // Read pixels from a screen dump
                         // file
     void EmulateTouch (int x, int y);
                         // Emulate a user touching the
                         // touch screen at co-ordinates
                         // (x, y)
     int GetAmountPaid ();
                         // Get the amount paid out by
                         // the bill dispenser
```

`cardNumber`

adalah satu penuding kepada satu rentetan aksara dengan penamat null yang mengandungi nombor kad.

`pixelArray`

adalah satu penuding kepada satu tatasusunan yang akan dimuatkan dengan komponen warna RGB bagi setiap piksel atas skrin ATM. `pixelArray` mesti sekurang-kurangnya mempunyai kapasiti sebanyak 307,200 untuk integer 32-bit.

`fileName`

adalah satu petunjuk kepada satu rentetan aksara dengan penamat null yang mengandungi nama fail longgok skrin dari mana komponen-komponen warna RGB bagi imej skrin terdahulu perlu diekstrakkan.

`x`

adalah jarak dari pinggir kiri skrin ke titik pada skrin di mana aksi penyentuhan skrin akan ditirukan.

`y`

adalah jarak dari pinggir atas skrin ke titik pada skrin di mana aksi penyentuhan skrin akan ditirukan.

`nilai pulangan oleh GetAmountPaid`

adalah amaun yang dibayar oleh pengeluar bil sejak panggilan terakhir kepada `GetAmountPaid` dalam dolar bulat. Contohnya, jika pengeluar bil telah membayar $50 sejak masa terakhir `GetAmountPaid` dipanggil, maka `GetAmountPaid` akan memulangkan nilai 50.

Infrastruktur ATM juga membekalkan fungsi-fungsi setara yang berikut yang boleh dipanggil dari Java:

```
public class ATMInfra {
     //...
     static public void emulateCardIn (
          String cardNumber) {/*...*/}
                    // Emulate the insertion of a card
                    // into the card reader
     static public int [] readScreenPixels ()
          {/*...*/}
                    // Read the pixels shown on the
                    // screen into pixelArray
     static public int [] readScreenDump (
          const char *fileName) {/*...*/}
                    // Read pixels from a screen dump
                    // file
     static public void emulateTouch (
          int x, int y) {/*...*/}
                    // Emulate a user touching the
                    // touch screen at co-ordinates
                    // (x, y)
     static public int getAmountPaid ()
          {/*...*/}
                    // Get the amount paid out by
                    // the bill dispenser
     //...
}
```

`cardNumber`
: adalah satu rentetan yang mengandungi nombor kad.

`nilai pulangan oleh readScreenPixels`
: adalah satu rujukan kepada satu tatasusunan integer yang mengandungi komponen-komponen warna RGB bagi piksel-piksel pada skrin.

`fileName`
: adalah satu rentetan yang mengandungi nama fail longgok skrin dari mana komponen-komponen warna RGB untuk imej skrin terdahulu perlu diekstrakkan.

`nilai pulangan oleh readScreenDump`
: adalah satu rujukan kepada satu tatasusunan integer yang mengandungi komponen-komponen warna RGB bagi piksel-piksel dalam fail longgok skrin.

`nilai pulangan oleh getAmountPaid`
: adalah amaun yang dibayar oleh pengeluar bil sejak panggilan terakhir kepada `GetAmountPaid` dalam dolar . Contohnya, jika pengeluar bil telah membayar $50 sejak masa terakhir `GetAmountPaid` dipanggil, maka `GetAmountPaid` akan memulangkan nilai 50.

Jurutera-jurutera perisian Burn mempunyai lebih kurang 50 sampel nombor kad dan PIN dalam satu fail 'TestCards.txt'. Panjang nombor-nombor kad adalah 16 digit dan panjang PIN adalah 6 digit. Setiap pasangan nombor kad-PIN disimpan dalam baris yang berasingan, dengan nombor kad dan PIN dipisahkan oleh satu aksara ruang. 10 baris pertama TestCards.txt adalah seperti dalam Figure 3.

```
0856669126311954 212384
4996552050476648 806909
9764925646670808 089694
7570032704671097 693010
1516410663483838 565200
6611851186922681 140063
0969351051899105 097351
4928296487649896 272250
6505872582625350 543119
2906025111421829 668369
```

Figure 13 – TestCards.txt

Pengolok kad bank mesti memilih satu pasangan baru nombor kad-PIN secara pseudo-rawak dari data dalam TestCard.txt bagi setiap kitar pengujian.

Setiap akaun yang akan digunakan untuk pengujian telah dimuatkan dengan satu amaun wang yang banyak yang tidak akan dihabiskan dalam pelaksanaan biasa pengolok kad bank tersebut.

Untuk menguji sama ada sesuatu kad diterima atau tidak, atau sama ada sesuatu PIN diterima atau tidak, atau sama ada pilihan akaun diterima atau tidak, pengolok kad bank mesti membanding paparan di atas skrin dengan longgok skrin yang diambil dari pelaksanaan yang berjaya. Sampel longgok-longgok skrin dari pelaksanaan terdahulu yang berjaya disimpan dalam fail-fail seperti disenarai di Table 2.

Table 5 – Nama Fail-fail Longgok Skrin

Nama fail longgok skrin	Kandungan
AcceptedCard.rgb	Satu longgok skrin yang dipaparkan selepas sesuatu kad berjaya dimasukkan.
AcceptedPIN.rgb	Satu longgok skrin yang dipaparkan selepas sesuatu PIN berjaya diinputkan.
AcceptedAccount.rgb	Satu longgok skrin yang dipaparkan selepas sesuatu akaun berjaya dipilih.

Setiap longgok skrin mengandungi tepat 307, 200 kata 32-bit, dikandungi dengan komponen-komponen warna RGB bagi setiap piksel pada paparan. Pengolok kad bank tersebut harus menggunakan fungsi `ReadScreenDump` untuk mengekstrak kata-kata 32-bit dari fail-fail longgok skrin. Begitu juga dengan fungsi `ReadScreenPixels` yang akan memulangkan tepat 307,200 kata 32-bit.

Disebabkan kursor pada skrin akan berkelip, piksel-piksel skrin bagi sesuatu operasi yang berjaya mungkin tidak berpadanan secara tepat dengan kandungan fail longgok skrin yang sepadan dengannya. Pengolok kad bank harus menganggap tatasusunan yang dipulangkan oleh `ReadScreenDump` dan `ReadScreenPixels` sebagai setara jika bilangan perbezaan piksel di antara kedua-duanya adalah tidak lebih dari 200 piksel.

Pengolok kad bank tersebut mesti meniru aksi memasukkan sesuatu PIN dengan menggunakan fungsi `EmulateTouch` untuk meniru aksi pengguna menyentuh papan kunci pada skrin sentuh. ATM tersebut memproses PIN secara automatik apabila semua enam digit telah dimasukkan. Pengolok kad bank tersebut mesti menunggu secara pseudo-rawak di antara 0.05 hingga 1.0 saat bagi setiap klik pada butang. Susunan skrin untuk penginputan PIN adalah seperti dalam Figure 4. Dimensinya adalah dalam unit piksel yang boleh terus diguna dalam fungsi `EmulateTouch`.

Susunan skrin untuk memilih sesuatu akaun adalah seperti dalam Figure 5. Pengolok kad bank mesti memilih akaun cek, simpanan atau kredit secara pseudo-rawak.

Susunan skrin untuk menginputkan amaun yang ingin dikeluarkan adalah seperti dalam Figure 6. Pengolok kad bank mesti memasukkan amaun yang bernilai di antara $10 hingga $1000 secara pseudo-rawak. Amaun tersebut mestilah dalam gandaan $10. Pengolok kad bank tersebut mesti memasukkan amaun dolar tepat diikuti dengan satu titik perpuluhan dan kemudian 2 digit sifar dan akhirnya butang OK. Pengolok kad bank tersebut mesti menunggu di antara 0.05 hingga 1.0 saat secara pseudo-rawak bagi setiap klik pada butang.

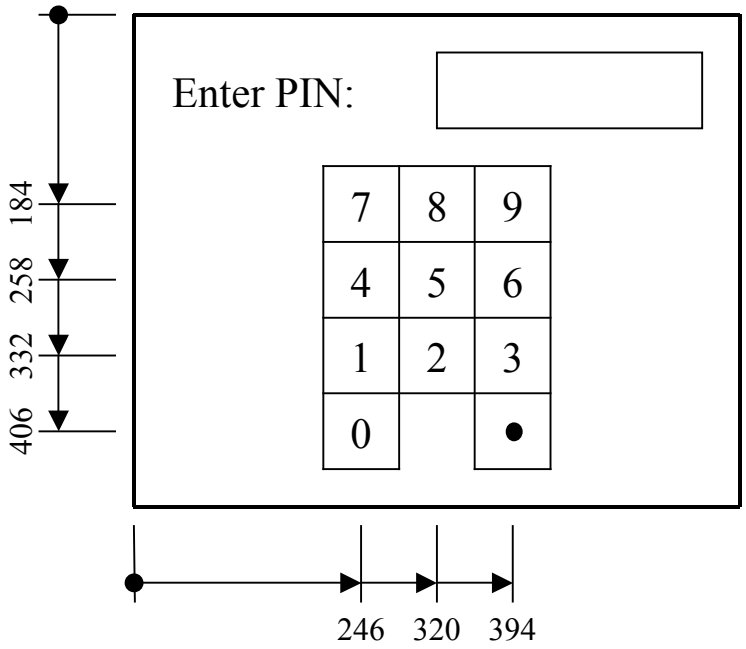

Figure 14 – PIN Screen Layout

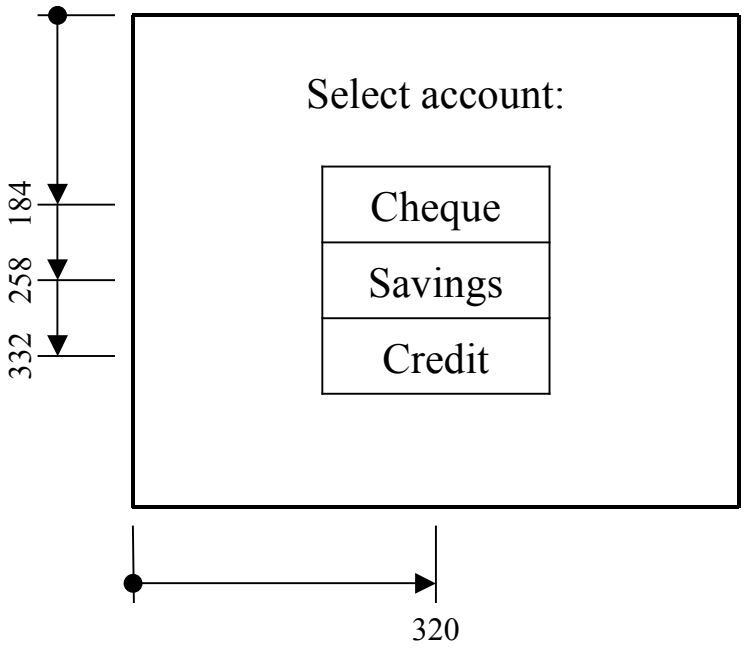

Figure 15 – Account Selection Screen Layout

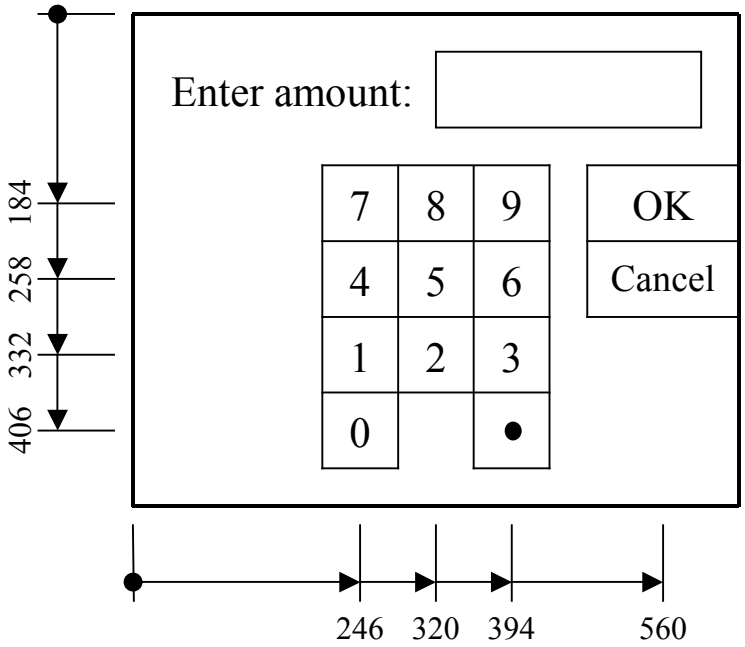

Figure 16 – Amount Screen Layout

ATM tersebut berupaya mengeluarkan lebih kurang lima wang kertas per saat, jadi beberapa saat mungkin diperlukan untuk mengeluarkan bilangan wang yang besar. Selepas input amaun yang ingin dikeluarkan, pengolok kad bank mesti menunggu selama lima saat dan kemudian memanggil fungsi `GetAmountPaid` setiap satu saat sambil mengumpul amaun yang dikeluarkan sehingga `GetAmountPaid` memulangkan sifar.

Apabila `GetAmountPaid` memulangkan sifar, pengeluaran telah tamat dan amaun yang dikumpulkan ialah jumlah amaun yang dikeluarkan.

Pengolok kad bank tersebut mesti semak bahawa amaun yang dikeluarkan adalah tepat.

Apabila pengolok kad bank dimulakan untuk kali pertama, ia mesti memanggil fungsi `GetAmountPaid` untuk reset amaun yang tertinggal kesan daripada pengujian yang sebelumnya.

Jika pengolok kad bank mengesan sesuatu kesilapan dalam operasi ATM, ia mesti menulis satu mesej yang menghuraikan kesilapan itu kepada output piawaian dan kemudian menamatkan urus niaga.

7 GABUNG DAN SUSUN

Sistem kawalan kod sumber E-Genting menjana senarai perubahan kod sumber seperti dalam Figure 7. Setiap senarai mengandungi perubahan yang dilakukan ke atas beberapa fail sumber. Perubahan kepada setiap fail sumber bermula dengan baris kepala. Baris kepala bermula dengan satu siri yang mengandungi 24 tanda sama ('=') diikuti dengan satu ruang, satu nama fail, kemudiannya satu ruang, satu versi perubahan ('37.2' dalam Figure 7) dan akhirnya satu lagi siri yang mengandungi 24 tanda sama.

Semasa fasa pengujian dalam kitar pembangunan perisian, beberapa fail senarai perubahan yang berjenis seperti dalam Figure 7 mungkin diberikan kepada penguji untuk pengujian. Fail-fail tersebut mungkin mengandungi beberapa perubahan kepada fail sumber yang sama, setiap dengan versi yang berbeza.

```
======================== drs/GuiComp.cpp 37.2 ========================
14a15
> // 24-08-07   JS      Changed 'comp issue loc date' to 'shift date'
145c146
<       frm.FedText (COL_WIDTH*2, l, "Comp issue loc date");
---
>       frm.FedText (COL_WIDTH*2, l, "Shift date");
======================== drs/GuiIncd.cpp 37.2 ========================
14a15
> // 24-08-07   JS      Changed 'comp issue loc date' to 'shift date'
146c147
<       frm.FedText (COL_WIDTH*2, l, "Comp issue loc date");
---
>       frm.FedText (COL_WIDTH*2, l, "Shift date");
```

Figure 17 – Senarai Perubahan Fail Sumber

Pasukan pengujian telah meminta satu aturcara yang akan menggabungkan perubahan dari beberapa senarai perubahan kepada satu output, dengan menyusun perubahan-perubahan mengikut nama fail dan kemudian versi.

Tugas anda ialah mengaturcarakan satu aturcara yang menerima satu senarai nama-nama fail dari baris perintah, dengan setiapnya mengenal pasti satu fail yang mengandungi senarai perubahan berjenis seperti dalam Figure 7, kemudiannya menyusun perubahan-perubahan dalam senarai-senarai perubahan mengikut nama fail serta versi, dan akhirnya

output perubahan itu ke output piawaian dalam format yang sama seperti fail-fail input kepada aturcara itu.

Dalam senarai-senarai perubahan tersebut, baris yang bermula dengan satu tanda sama merupakan satu baris kepala. Perubahan-perubahan dalam fail sumber tidak akan mengandungi baris yang bermula dengan satu tanda sama.

Nama-nama fail tidak akan mengandungi sebarang ruang.

Dalam perbandingan nombor-nombor versi, fungsi penyusunan harus menganggap versi tersebut sebagai dua nombor berasingan yang dipisahkan oleh satu titik perpuluhan, yang merupakan satu nombor terbitan dan satu nombor aras. Aturcara tersebut harus menyusun perubahan-perubahan kepada fail yang sama mengikut nombor terbitan dan kemudian nombor aras. Contohnya, versi 37.5 harus dipapar sebelum versi 37.12.

8 PEMANTAU PENCEMARAN UDARA

Dalam satu usaha untuk meningkatkan kesedaran masyarakat umum dan industri tentang punca-punca pencemaran udara, Jabatan Alam Sekitar telah memasang sebilangan besar pengesan pencemaran udara di sekitar Lembah Kelang . Pengesan-pengesan ini menghantar maklumat lokasinya yang dicapai dari penerima GPS yang terletak dalam pengesan, bersama dengan nilai indeks pencemaran udara (API) di lokasi itu, kepada satu pelayan pusat setiap 10 minit.

Pengalaman telah menunjukkan bahawa maklumat kasar ini sukar difaham. Apa yang Jabatan Alam Sekitar inginkan ialah satu peta dinamik dalam talian yang boleh dicapai melalui Internet dan menunjukkan kawasan-kawasan dengan pencemaran udara yang tinggi dan rendah dengan warna-warna yang berbeza, sama seperti seri warna hypsometri pada peta topografikal.

Data kasar dari pengesan-pengesan pencemaran udara boleh didapati dengan memuat turun sebuah fail teks melalui Internet. Untuk tujuan pengujian, satu laman telah dibangunkan di http://genting.com.my/rnd/airdata.txt. Contoh data dalam airdata.txt adalah seperti ditunjuk dalam Figure 8. Format bagi setiap baris dalam airdata.txt adalah seperti ditunjuk dalam Figure 9.

```
E10126033N00259537194
E10149349N00255301342
E10147289N00252330231
E10128254N00258029010
E10134282N00313547133
E10146512N00306312421
E10133274N00303149145
E10129093N00300315194
E10123597N00311052107
E10135371N00311513370
          :
          :
```

Figure 18 – Sampel Data Pencemaran Udara dalam airdata.txt

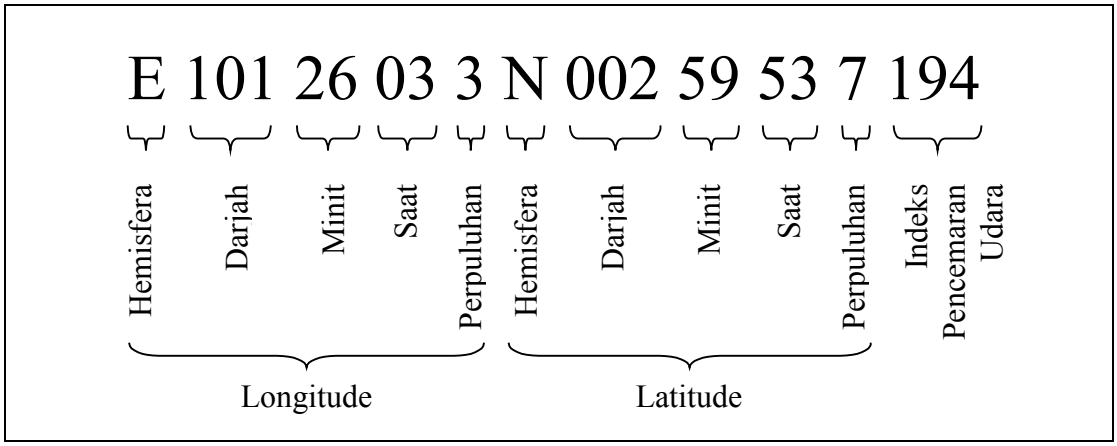

Figure 19 – Format barisan bagi airdata.txt

Hemisfera longitude adalah sama ada ‘E’ bagi timur atau ‘W’ bagi barat. Hemisfera latitude adalah sama ada ‘N’ bagi utara atau ‘S’ bagi selatan.

Semua koordinasi dalam airdata.txt terletak di antara longitude 101° 20’ hingga 101° 50’ timur and latitude 002° 50’ hingga 003° 20’ utara.

Lembah Kelang adalah cukup dekat dengan Khatulistiwa untuk menganggap longitude dan latitude sebagai koordinat x-y yang ringkas tanpa sebarang pengherotan dari segi penglihatan.

Jabatan Alam Sekitar mempunyai lebih kurang 500 pengesan-pengesan pencemaran udara bertabur di sekitar Lembah Kelang . Namun demikian, bilangan ini berubah-ubah dari masa ke semasa kerana pengesan-pengesan akan diambil keluar untuk penyelenggaraan dan kemudian dipasang semula ke lokasi yang sama atau yang berlainan.

Tugas anda ialah menghasilkan satu aturcara klien yang boleh dimuat turun melalui Internet, contohnya sebagai Java applet, yang mempersembahkan satu peta pencemaran udara yang memaparkan kawasan-kawasan yang tercemar dengan tahap berbeza dalam warna yang berbeza. Adalah bukan satu keperluan untuk mengaturcara klien tersebut dengan Java. Anda bebas untuk mengaturcara klien tersebut dalam sebarang bahasa pengaturcaraan yang membolehkan aturcara tersebut dimuat turun melalui Internet dan dilaksanakan dalam pelayar web yang biasa. Satu contoh output yang diperlukan adalah seperti yang ditunjukkan dalam Figure 10.

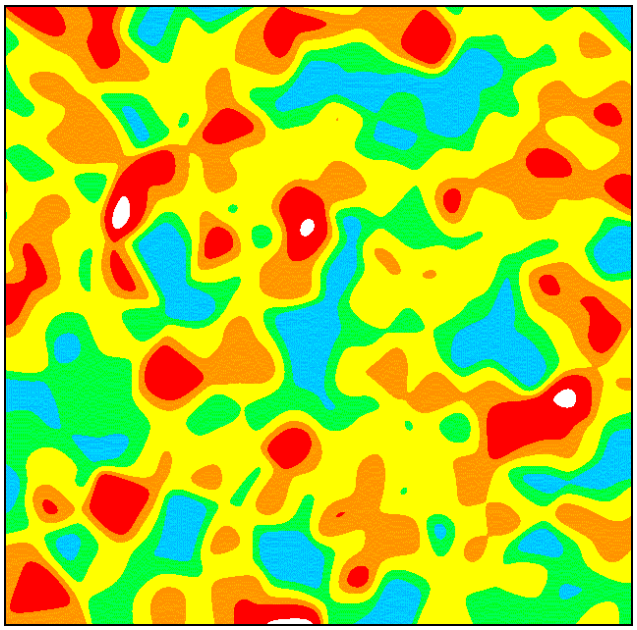

Figure 20 – Contoh Peta Pencemaran Udara

Tahap pencemaran udara di kawasan-kawasan yang berbeza mesti diwarna mengikut warna-warna dalam Table 3.

Table 6 – Pengekodan Warna Indeks Pencemaran Udara

Indeks Pencemaran Udara (API)	Status	Komponen-komponen warna RGB		
		Merah	Hijau	Biru
0-50	Baik	0	204	255
51-100	Sederhana	0	255	64
101-200	Tidak sihat	255	255	0
201-300	Sangat tidak sihat	255	153	0
301-500	Berbahaya	255	0	0
Melebihi 500	Kecemasan	255	255	255

Peta pencemaran udara tersebut harus dipersembahkan pada skala lebih kurang 1:330,000. Sebagai rujukan anda, satu minit longitude pada Khatulistiwa (satu batu nautikal) adalah 1852 meter atau 72913 inci.

Equation 2 memberi satu cara untuk menganggar API pada sesuatu titik (x, y) dalam had koordinat-koordinat dalam airdata.txt. Aturcara klien tersebut harus menggunakan Equation 2 atau algoritma yang hampir dengannya untuk menentukan warna yang perlu ditunjuk oleh setiap piksel dalam peta pencemaran udara.

Equation 5 – Interpolasi Indeks Pencemaran Udara

$$api(x,y) = \frac{\sum_{i=1}^{n} ad_i \, w(x,y,x_i,y_i)}{\sum_{i=1}^{n} w(x,y,x_i,y_i)}$$

Where:

api(x,y)	adalah API anggaran di titik (x,y)
x	ialah longitude dalam minit
y	ialah latitude dalam minit
n	ialah bilangan pengesan.
ad_i	ialah API yang dilaporkan oleh pengesan i dalam airdata.txt.
x_i	ialah longitude dalam minit bagi pengesan i.
y_i	ialah latitude dalam minit bagi pengesan i.
$w(x,y.x_i,y_i)$	ialah fungsi penimbang yang ditakrif berikut.

Equation 6 – Fungsi Penimbang

$$w(x,y.x_i,y_i) = e^{-\left(\frac{((x_i - x)^2 + (y_i - y)^2)}{A^2}\right)}$$

Where:

$w(x,y.x_i,y_i)$	ialah nilai untuk fungsi penimbang.
x	ialah longitude dalam minit.
y	ialah latitude dalam minit.
x_i	ialah longitude dalam minit untuk pengesan i.
y_i	ialah latitude dalam minit untuk pengesan i.
e	ialah nombor Euler. (lebih kurang 2.7128)
A	ialah pemalar 0.9 batu nautical.

Apabila menghitung API bagi mana-mana titik (x, y), pengesan-pengesan dengan nilai fungsi penimbang kurang daripada 10^{-6} boleh diabaikan.

Disebabkan pengesan-pengesan akan menghantar maklumat terkini kepada pelayan setiap 10 minit, secara purata, sekurang-kurangnya terdapat satu baris dalam airdata.txt akan berubah setiap 1.2 saat. Oleh yang demikian, klien tersebut harus mencapai salinan airdata.txt yang lain dari pelayan dan menjana semula paparan tersebut setiap 30 saat. Untuk memenuhi keperluan ini, aturcara klien tersebut harus berupaya untuk menukar keputusan-keputusan pengesan kepada satu peta pencemaran udara dalam sekurang-kurangnya 30 saat. Masa penukaran yang kurang daripada 5 saat lebih digemari. Aturcara klien tersebut harus memaparkan peta pencemaran udara lama apabila sedang menjana peta baru.

Percubaan yang terdahulu untuk mengaturcara pemantau pencemaran udara telah menunjukkan bahawa implementasi yang tidak matang yang cuba menghitung fungsi penimbang bagi setiap pengesan bagi setiap piksel mengambil masa lebih kurang 51 saat untuk menjana imej baru.

Kredit untuk soalan ini akan dibahagikan kepada dua bahagian, bahagian pertama ialah menghasilkam pemantau pencemaran udara yang berfungsi secara sempurna dan bahagian kedua ialah memastikan pemantau tersebut mencapai matlamat prestasinya.