E-GENTING
PROGRAMMING COMPETITION 2011

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.	Rest Break Profile	200
2.	XML Conversion	350
3.	Report Status	500
4.	Square Root Function	50

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 REST BREAK PROFILE

A chain of supermarkets operates a shift system at the checkout counters where each cashier works for a period and then has a 20-minute break and then works for another period and has another 20-minute break and so on. However, some cashiers have complained that they have not been able to take their full 20-minute break because the replacement cashier has not turned up on time.

To deal with this conflict the management of the supermarket chain would like to include an assessment of whether or not each cashier was returning from rest breaks on time in the cashier's annual appraisal. The management would like the IT department to produce a histogram for each cashier, called the Rest Break Profile, similar to the example in Figure 1.

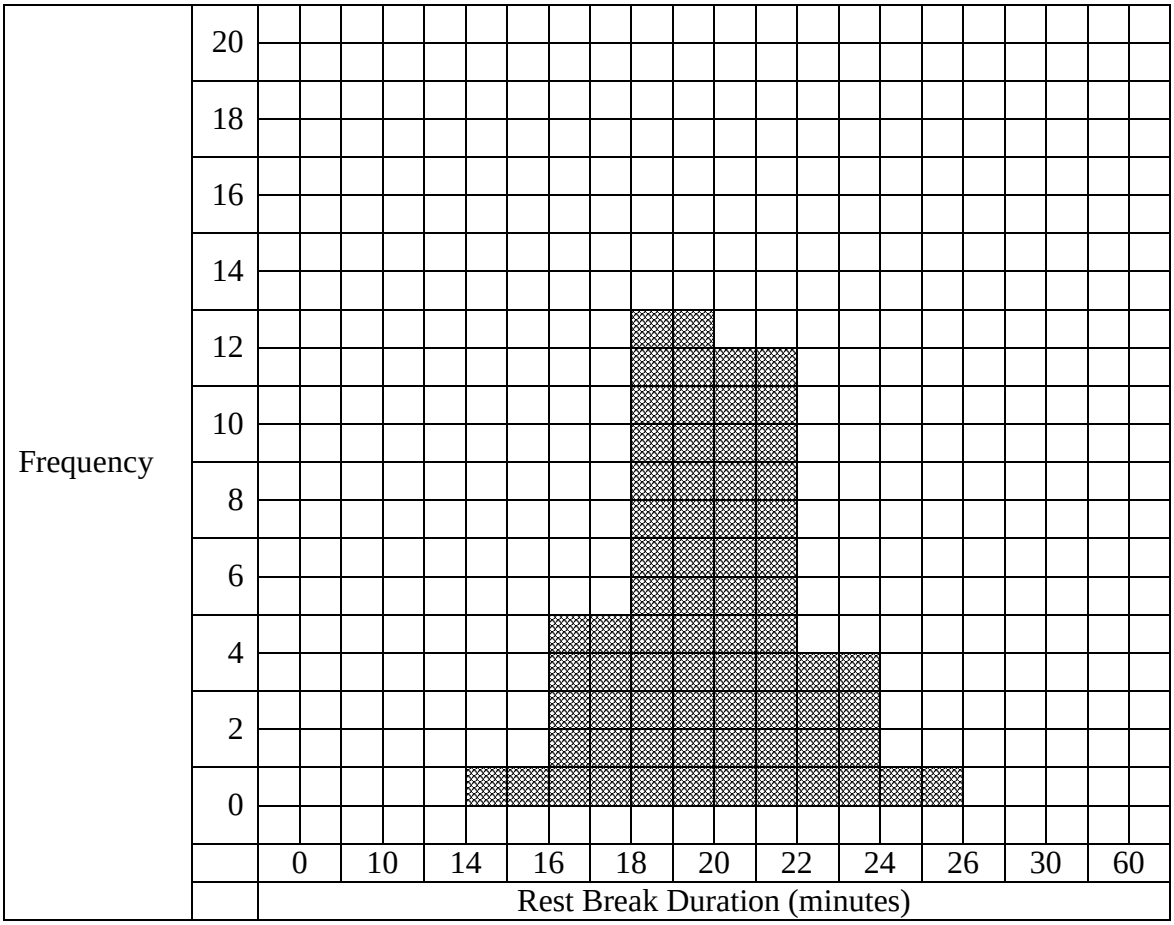


Figure 1 – Rest Break Profile

As part of the financial control system, cashiers log on to the cash register when they start their shift and log off when they finish. The financial control system records each of these sessions in the shifts table presented in Figure 2.

```
create table shifts (
     shiftDate      date not null,
     shiftOpenTime  timestamp not null,
     shiftCloseTime timestamp not null,
     cashierCode    char(8) not null
);
```

`date`
: is the trading day date of the shift.

`shiftOpenTime`
: is the real time at which the log-on session started.

`shiftCloseTime`
: is the real time at which the log-on session finished.

`cashierCode`
: is an 8-character code that identifies the cashier who logged on to the terminal.

Figure 2 – shifts Table

The management believes that it should be possible to determine the duration of each rest break by looking at the sequence of shift rows for an individual cashier and taking the difference between the shiftCloseTime of each closing shift and the shiftOpenTime of the next shift in sequence.

Your task is to write a computer program that analyses the data in the shifts table and produces a Rest Break Profile for an individual cashier.

Your program must accept the following parameters:

1. cashier code;
2. from-date;
3. to-date.

Your program must determine the durations of the rest breaks taken by the cashier identified by the cashier code, between the from-date and to-date, inclusive. It must then group the rest breaks into the intervals shown in Figure 1 and generate an output that shows the number of rest breaks in each interval. Rest breaks longer than 60 minutes may be assumed to be the time between the last shift of one day and the first shift of the next day and may be disregarded. Note that the durations of the rest break intervals are not equal.

Your program should produce a properly formed HTML document that contains HTML table-drawing elements that construct a histogram similar to that shown in Figure 1 that can be viewed on a web browser.

Alternatively, but less desirably, your program may produce a text file similar to the example in Figure 4.

If the frequency of rest breaks in any interval is greater than 20, your program must automatically scale the Y axis using scales from the series 1/2, 1/5, 1/10, 1/20, 1/50, 1/100, … so that the histogram just fits into 20 vertical graduations.

Programmers writing in C or C++ may make use of the date conversion functions described in Figure 3. 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 3 – Double Time Functions

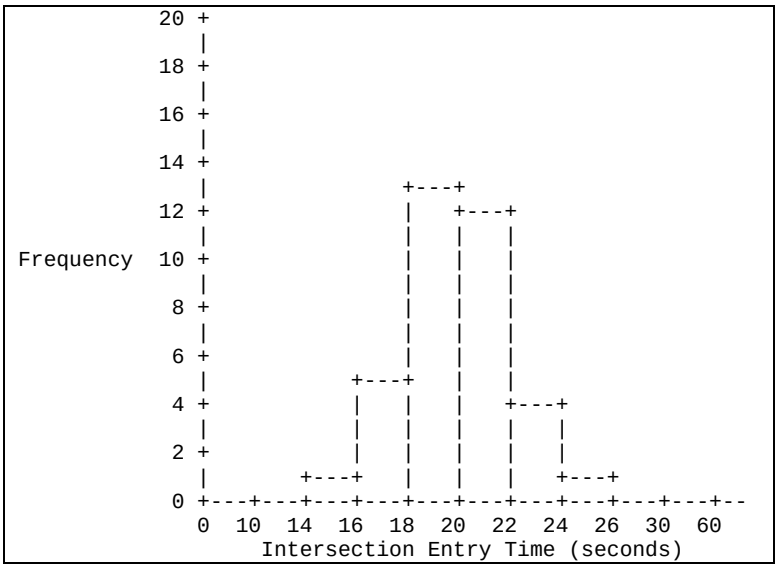


Figure 4 – Alternative Text Format Output

2 XML CONVERSION

A communications project needs a converter that can translate an instance of a C++ structure (or a Java data class) into an XML message and vice versa. The project manager has an idea that it might be possible to construct a map that defines the relationship between the fields in a structure and the XML elements to be written or read. The map could then be passed to the writing and reading functions to convert an instance of the structure to XML and vice versa. This arrangement would let an application convert a large variety of different structures with a minimum of structure-specific programming.

For example, consider the structure in Figure 5. To convert the structure Item_t, an application would create a conversion map for the structure by instantiating the XmlMap_c template and then calling the AddInt and AddString methods to define the fields in Item_t. Once the conversion map is defined the application could call WriteXml or ReadXml to convert an instance of Item_t into XML or back again. An example of the XML equivalent of the Item_t structure is presented in Figure 6.

Similarly, the same concept could be developed in Java as shown in Figure 7. In this case, the two string arguments to addInt and addString refer to the XML element name and the name of the field in the Java class. The name of the field in the Java class can be passed to Class.getDeclaredField to obtain an instance of Field that refers to the field in the class to be converted.

Your task is to program an XML Converter generally based on the project manager's concept.

The XML Converter must be able to convert C++ structures or Java classes that contain integer and string fields into their XML equivalents and vice versa.

The XML Converter must have a means for defining a conversion map that defines the relationship between the structure to be converted and its XML equivalent.

The XML Converter must have methods that an application can call that accept a reference to the conversion map and convert an instance of the structure into its XML equivalent and the XML equivalent back into an instance of the structure.

At this time, the XML Converter is only required to convert integer and string fields, but the XML Converter must be designed in a way that allows it to be enhanced to convert other types of fields.

The XML Converter is not required to convert special characters in strings into their XML escape sequences. For example, it is not required to convert '&' into '&'. The XML Converter may assume that strings do not contain any special characters.

The XML Converter is not required to emit or decode the XML header '<?xml version="1.0" ?>'.

The XML Converter should disregard white space (spaces, tabs and new line characters) between field elements.

The method that converts XML back into the internal format should have a means for telling the application that the XML data was invalid and should accept the XML field elements in any order.

The XML Converter is not required to exactly follow the application programming interface presented in Figure 5 and Figure 7. You can define your own methods for defining the conversion map and converting the structure provided the 'must-level' requirements presented in the preceding paragraphs are satisfied.

```
// The structure to be converted

struct Item_t {
      int         itemNo;
      string      itemName;
      int         quantity;
};

// An application might construct an XML conversion map as follows:

XmlMap_c<Item_t> itemMap;
itemMap.AddInt ("itemNo", &Item_t::itemNo);
itemMap.AddString ("itemName", &Item_t::itemName);
itemMap.AddInt ("quantity", &Item_t::quantity);

// It could then use the following function to write the XML equivalent
// of an instance of Item_t to the standard output stream

WriteXml (&cout, "item", &item, &itemMap);

// And the application could use the following function to
// read the XML equivalent from the standard input stream

ReadXml (&cin, "item", &item, &itemMap);
```

Figure 5 – Example XML Conversion Functions in C++

```
<item>
    <itemNo>1234</itemNo>
    <itemName>Widget</itemName>
    <quantity>7</quantity>
</item>
```

Figure 6 – Example of Item_t Structure Converted into XML

```
// The structure to be converted

public class Item {
      int         itemNo;
      String      itemName;
      int         quantity;
}

// An application might construct an XML conversion map as follows:

XmlMap itemMap;
itemMap = new ItemMap(Item.class);
itemMap.addInt ("itemNo", "itemNo");
itemMap.addString ("itemName", "itemName");
itemMap.addInt ("quantity", "quantity");

// It could then use the following function to write the XML equivalent
// of an instance of Item_t to the standard output stream

XmlWriter.writeXml (System.out, "item", item, itemMap);

// And the application could use the following function to
// read the XML equivalent from the standard input stream

XmlReader.readXml (System.in, "item", item, itemMap);
```

Figure 7 – Example XML Conversion Functions in Java

3 REPORT STATUS

A reporting program receives a selection expression similar to the example in Figure 8 and converts it into an expression tree as shown in Figure 9. The program uses the selection expression to select the data to be included in the report. The selection keys include ‘date’, ‘colour’ and ‘type’.

On the database, recent data is preliminary and still subject to change, whereas older data has been finalised and is rarely changed. Specifically, data with a date up to and including a ‘commit date’ is final, whereas data after the commit date is preliminary.

The reporting requirements specify that if the report includes any data after the commit date it must be labelled ‘preliminary’, otherwise it must be labelled ‘final’.

Your task is to write a function that accepts an expression pointer and the commit date and returns a Boolean flag that is true if the report is prelimnary or false if the report is final.

Your function should assume that the preliminary or final status is independent of the non-date selection keys. For example, if the commit date is 18 October 2011 and the selection expression is the one in Figure 8, then the report is preliminary irrespective of the data included or excluded by the ‘colour = ‘green’’ condition.

Nevertheless, your function must correctly analyse compound expressions such as the one presented in Figure 10, which would result in a preliminary report for a commit date of 17 October 2011.

The expression nodes are defined by the C++ declarations in Figure 11 or the Java declarations in Figure 12. The operations perform the functions defined in Table 1.

All dates are stored in strings in YYYY-MM-DD format. Your function may assume that the date it receives is correctly formatted and a valid date.

C and C++ programmers may make use of the double time functions in Figure 3.

```
(date >= '2011-10-18' and date <= '2011-10-20') and colour = 'green'
```

Figure 8 – Example Selection Expression

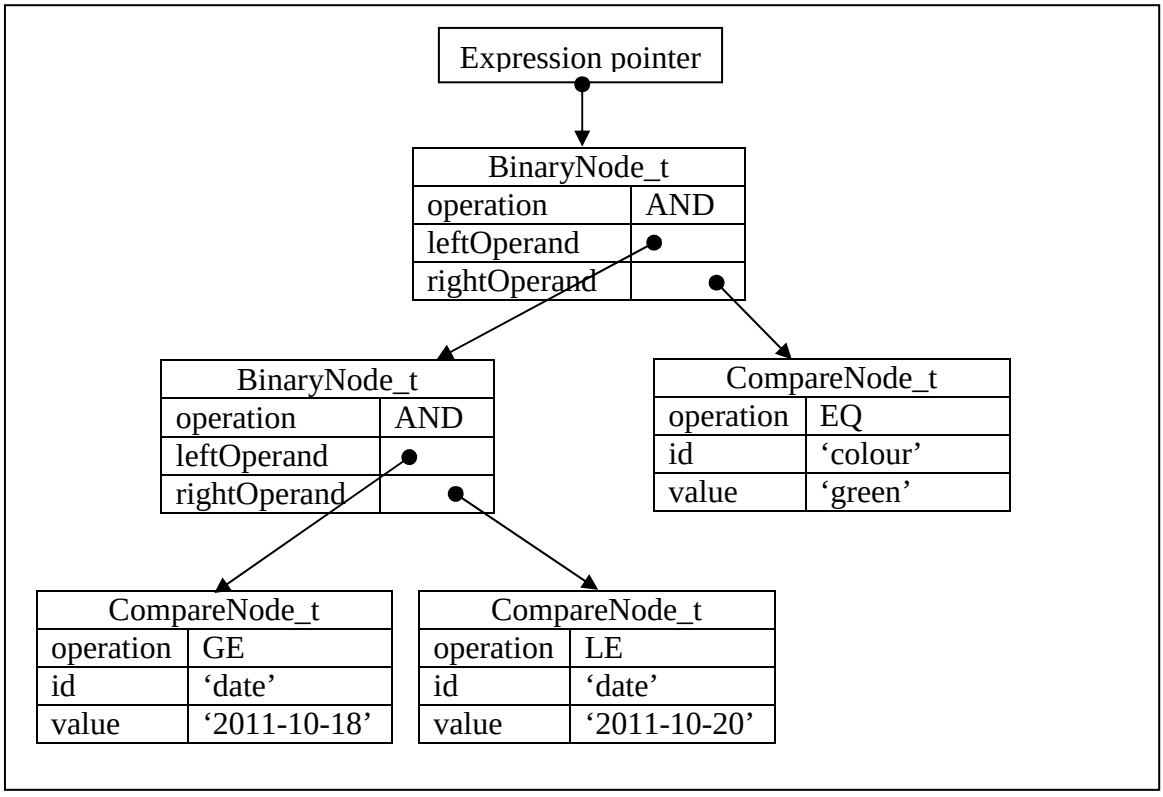


Figure 9 – Expression Tree Structure

```
(colour = 'yellow' and (date >= '2011-10-15' and date <= '2011-10-17'))
                                    or
 (type = 'premium' and (date >= '2011-10-16' and date <= '2011-10-19'))
```

Figure 10 – Example Compound Expression

```
enum Operation_t {EQ, NE, GT, GE, LT, LE, AND, OR, NOT};

struct Node_t {
      Operation_t       operation;
};

struct CompareNode_t : public Node_t {
      string            id;
      string            value;
};

struct BinaryNode_t : public Node_t {
      Node_t            *leftOperand;
      Node_t            *rightOperand;
};

struct UnaryNode_t : public Node_t {
      Node_t            *operand;
};
```

Figure 11 – Expression Node Declarations in C++

```
public class Expr {
      public static enum Operation {
            EQ, NE, GT, GE, LT, LE, AND, OR, NOT}

      public static class Node {
            Operation         operation;
      }

      public static class CompareNode extends Node {
            String            id;
            String            value;
      }

      public static class BinaryNode extends Node {
            Node              leftOperand;
            Node              rightOperand;
      }

      public static class UnaryNode extends Node {
            Node              operand;
      }
}
```

Figure 12 – Expression Node Declarations in Java

Table 1 – Operation Node Types and Values

Operation	Node type	Node value
EQ	CompareNode_t	True if and only if the value of the selection key ‘id’ is equal to the literal string ‘value’.
NE	CompareNode_t	True if and only if the value of the selection key ‘id’ is not equal to the literal string ‘value’.
GT	CompareNode_t	True if and only if the value of the selection key ‘id’ is greater than the literal string ‘value’.
GE	CompareNode_t	True if and only if the value of the selection key ‘id’ is greater than or equal to the literal string ‘value’.
LT	CompareNode_t	True if and only if the value of the selection key ‘id’ is less than the literal string ‘value’.
LE	CompareNode_t	True if and only if the value of the selection key ‘id’ is less than or equal to the literal string ‘value’.
AND	BinaryNode_t	True if and only if the values of the expression trees addressed by ‘leftOperand’ and ‘rightOperand’ are both true.
OR	BinaryNode_t	True if and only if either or both of the expression trees addressed by ‘leftOperand’ and ‘rightOperand’ are true.
NOT	UnaryNode_t	True if and only if the value of the expression tree addressed by ‘operand’ is false.

4 SQUARE ROOT FUNCTION

You have been assigned to a project to develop an application for a small microprocessor. The microprocessor has floating point arithmetic, but there is no mathematical library. In order to evaluate some statistical parameters, the project needs a simple and robust square root function.

The project manager (for reasons that remain a mystery) has decided to do this by implementing a trial and error algorithm. The algorithm works like this:

For numbers greater than one, the square root of the number always lies between one and the number. Consequently, to get started, the square root function must set a lower limit to one and an upper limit to the number. It must then determine the midpoint between the lower and upper limits and evaluate the square of the midpoint. If the square of the midpoint is greater than the number, the square root function must move the upper limit to the midpoint and conversely if the square of the midpoint is less than the number, it

must move the lower limit to the midpoint. After moving the appropriate limit, the square root function must evaluate a new midpoint and repeat the process until the desired precision is obtained.

For numbers less than one, the algorithm is essentially the same, except that the initial lower limit is the number and the initial upper limit is one.

The required precision for double precision floating point numbers is 8 significant digits. As the actual square root of the number is at any time between the upper and lower limits, the precision at any iteration can be determined by dividing the difference between the limits by the lower limit. When this is less than $1/10^8$ any number between the limits will be an estimate of the square root of the number to the required precision. To minimise the error, the square root function should return the midpoint between the final limits that satisfy the precision requirement.

The square root function must return exact values for the special cases of zero and one.

If an application attempts to calculate the square root of a negative number, the square root function should display an appropriate message on the standard error stream and terminate the program.

Your task is to write the square root function.

The square root function must have the declaration in Figure 13.

For C and C++:

```
double SquareRoot (double x)
{
     /*...*/
}
```

For Java:

```
public class MicroMath {
     public static double SquareRoot (double x)
     {
          /*...*/
     }
}
```

Figure 13 – Square Root Function Declaration

PERTANDINGAN PENGATURCARAAN E-GENTING 2011

Arahan-arahan am:

1. Jawab satu atau lebih soalan yang diberikan.
2. Pertandingan ini adalah satu ujian di mana peserta-peserta dibenarkan untuk 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 menyelesaikan masalah tersebut dengan sedikit usaha tambahan.
6. Sekiranya jawapan anda merupakan penyelesaian yang munasabah, anda akan menerima kredit untuk hasilan methodikal seperti gambar rajah aliran data, gambar rajah peralihan keadaan, jadual dan sebagainya.
7. Jumlah markah anda adalah 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.
    - Pergandingan di antara modul-modul mestilah nyata.
    - Setiap modul harus melakukan satu perkara dengan baik.
9. Markah yang diperuntukan kepada setiap soalan adalah seperti berikut:

No	Tajuk	Markah
1.	Profil Waktu Rehat	200
2.	Penukar XML	350
3.	Status Laporan	500
4.	Fungsi Punca Kuasa Dua	50

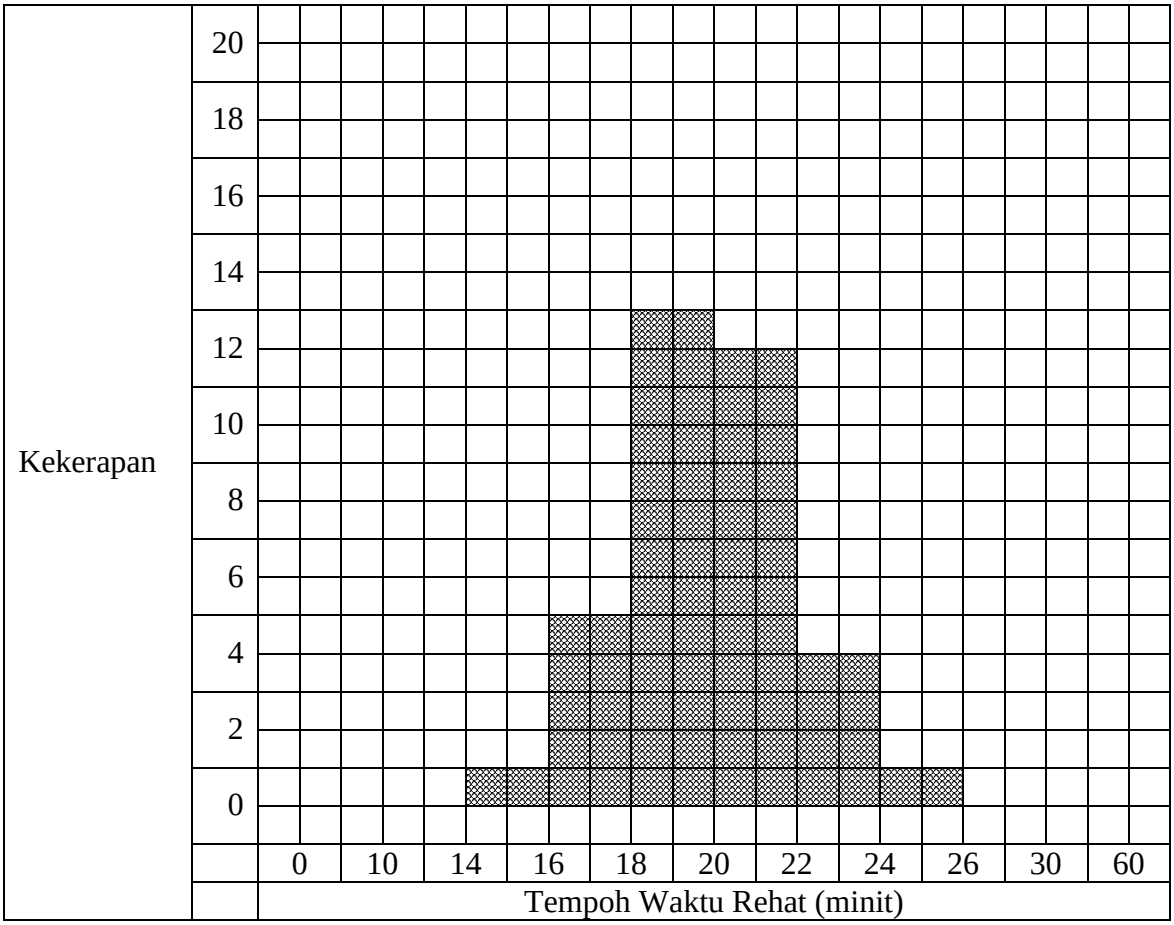
10. Kecuali dinyatakan, program anda boleh ditulis dengan menggunakan mana-mana bahasa pengaturcaraan utama di bawah mana-mana sistem operasi utama.
11. Kecuali dinyatakan, anda boleh menggunakan semua fungsi piawaian perpustakaan dalam bahasa pengaturcaraan dan sistem operasi yang anda pilih.
12. Perkataan-perkataan ‘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.

5 PROFIL WAKTU REHAT

Sebuah rangkaian pasar raya menggunakan sistem giliran di kaunter bayaran yang memperuntukan setiap juruwang suatu tempoh kerja diikuti dengan waktu rehat selama 20 minit. Selepas waktu rehat, juruwang akan kembali bekerja untuk tempoh seterusnya yang diikuti 20 minit waktu rehat lagi dan ini diulangi seterusnya. Akan tetapi, sesetengah juruwang mengadu bahawa mereka tidak berpeluang untuk berehat selama 20 minit kerana juruwang yang harus menggantikan mereka lewat hadir pada waktu yang tepat.

Untuk menyelesaikan masalah ini, pihak pengurusan pasar raya ingin mengambil kira sama ada setiap juruwang kembali dari waktu rehat pada masa yang tepat dalam penilaian prestasi tahunan juruwang. Pihak pengurusan meminta jabatan IT menghasilkan histogram yang dinamakan sebagai Profil Waktu Rehat untuk setiap juruwang, seperti contoh berikut dalam Figure 1:



Gambarajah 14 – Profil Waktu Rehat

Sebagai sebahagian daripada sistem kawalan kewangan, juruwang daftar masuk ke mesin tunai apabila memulakan tempoh kerja dan daftar keluar apabila tempoh kerja mereka tamat. Sistem kawalan kewangan tersebut mencatat setiap sesi ke dalam jadual pangkalan data seperti berikut:

```
create table shifts (
     shiftDate      date not null,
     shiftOpenTime  timestamp not null,
     shiftCloseTime timestamp not null,
     cashierCode    char(8) not null
);
```

`date`
: merupakan tarikh perniagaan bagi tempoh tersebut.

`shiftOpenTime`
: merupakan masa sesi daftar masuk bermula.

`shiftCloseTime`
: merupakan masa sesi daftar masuk ditamatkan.

`cashierCode`
: merupakan 8 aksara kod yang mengenalpasti juruwang yang daftar masuk ke terminal.

Gambarajah 15 – Jadual shif

Pihak pengurusan percaya bahawa tempoh setiap waktu rehat dapat dikenalpasti daripada urutan rekod bagi seseorang juruwang dengan mengambil kira perbezaan masa di antara waktu sesi daftar masuk ditamatkan dan waktu daftar masuk bagi sesi yang seterusnya.

Tugas anda ialah untuk mengaturcara satu program komputer yang menganalisa data dalam jadual pangkalan data dan menghasilkan satu Profil Waktu Rehat bagi seseorang juruwang.

Program anda mesti menerima parameter seperti berikut:

1. kod juruwang;
2. tarikh-dari;
3. tarikh-sehingga.

Program anda mesti menentukan jangka masa waktu rehat yang diambil oleh juruwang yang dikenalpasti melalui kod juruwang tersebut, di antara tarikh-dari ke tarikh-sehingga, termasuk kedua-dua tarikh tersebut. Program tersebut mesti mengumpul tempoh waktu rehat ke dalam selang masa seperti dalam Figure 1 dan menjana satu output yang memaparkan bilangan waktu rehat dalam setiap selang masa. Waktu rehat yang lebih lama daripada 60 minit dianggap sebagai masa di antara tempoh kerja terakhir bagi sesuatu hari dengan tempoh kerja pertama untuk hari berikutnya yang boleh diabaikan. Perhatikan bahawa tempoh-tempoh selang masa adalah tidak sama.

Program anda harus menghasilkan dokumen HTML yang menggunakan elemen-elemen jadual HTML untuk membina histogram seperti yang ditunjukkan dalam Figure 1 dan boleh dipaparkan dalam pelayar web.

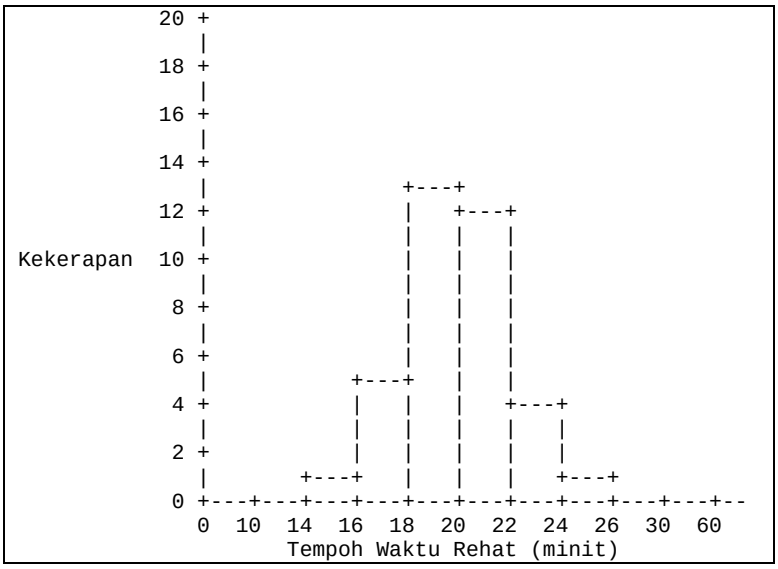
Alternatif lain yang kurang digemari ialah program anda boleh memilih untuk menghasilkan fail teks seperti contoh dalam Figure 4.

Jika kekerapan waktu rehat di mana-mana selang masa adalah melebihi 20, program anda mesti mengubah skala paksi Y kepada skala siri 1/2, 1/5, 1/10, 1/20, 1/50, 1/100, … supaya histogram tersebut dapat dimuatkan ke dalam 20 ansuran-ansuran tegak.

Pengaturcara-pengaturcara C atau C++ boleh menggunakan fungsi-fungsi penukaran tarikh seperti dalam Figure 3. Pengaturcara-pengaturcara Java dan C-sharp harus menggunakan fungsi-fungsi penukaran tarikh yang dibekalkan oleh perpustakaan-perpustakaan bahasa Java dan C-sharp.

```
typedef double Dtime_t;
                            // Jenis data masa double.  Tarikh dan masa disimpan
                            // dalam unit saat sejak 00:00 pada 1 January 1970.
                            // Tarikh disimpan dalam unit saat sejak 00:00 pada
                            // 1 January 1970 hingga 00:00 pada tarikh yang
                            // disimpan.
static const Dtime_t NULL_DTIME = -(65536.0*65536.0*65536.0*65536.0);
                            // Nilai masa double terpelihara (sebelum dentuman
                            // besar) digunakan sebagai nilai masa yang tak sah.
Dtime_t PackDtime (int yr, int mo, int dy, int hr, int mi, int se);
                            // Menghasilkan satu masa double time value daripada
                            // bahagian-bahagiannya.  yr adalah tahun 4-digit
                            // (e.g.2008).  mo adalah nombor bulan kalendar
                            // (e.g. 1 untuk January).  dy adalah haribulan
                            // (e.g. 1 untuk hari pertama sesuatu bulan).
                            // hr, mi and se adalah bahagian jam, minit dan saat
                            // daripada masa 24 jam.
                            // PackDtime memulangkan NULL_DTIME jika bahagian-
                            // bahagian masa tidak dapat menghasilkan tarikh
                            // dan masa yang sah.
void UnpackDtime (Dtime_t dtime, int *yr, int *mo, int *dy,
      int *hr, int *mi, int *se);
                            // Membahagikan satu nilai masa double time
                            // kepada bahagian-bahagiannya.
Dtime_t DtimeFromSql (const char *sqlTime);
                            // Menukar satu SQL date atau timestamp kepada format
                            // masa double.  DtimeFromSql memulangkan NULL_DTIME
                            // jika SQL date or timestamp tidak sah.
char *SqlDateFromDtime (char *sqlDate, Dtime_t dtime);
                            // Memuatkan satu SQL date string dengan nilai masa
                            // double.  SqlDateFromDtime memulangkan sqlDate.
char *SqlTimestampFromDtime (char *sqlTimestamp, Dtime_t dtime);
                            // Memuatkan satu SQL timestamp string dengan nilai
                            // masa double.  SqlTimestampFromDtime memulangkan
                            // sqlTimestamp.
Dtime_t DtimeFromUserDate (const char *userDate);
                            // Menukar satu string tarikh dalam format DD-MM-YY
                            // kepada format masa double.
char *UserDateFromDtime (char *userDate, Dtime_t dtime);
                            // Menukar bahagian tarikh daripada nilai masa double
                            // kepada satu string tarikh dalam format DD-MM-YY.
                            // UserDateFromDtime memulangkan userDate.
char *UserTimeFromDtime (char *userTime, Dtime_t dtime);
                            // Menukar bahagian masa daripada nilai masa double
                            // kepada satu string masa dalam format HH:MM:SS.
                            // UserTimeFromDtime memulangkan userTime.
```

Gambarajah 16 – Fungsi-fungsi masa dan tarikh



Gambarajah 17 – Output alternatif dalam format teks

6 PENUKARAN XML

Sesuatu projek komunikasi memerlukan satu penukar yang berupaya untuk menterjemahkan sesuatu struktur C++ (atau data kelas Java) kepada mesej XML dan sebaliknya. Pengurus projek tersebut berpendapat bahawa ia adalah munasabah untuk membina satu peta yang menentukan hubungan di antara pembolehubah-pembolehubah dalam sesuatu struktur dengan elemen-elemen XML yang akan ditulis atau dibaca. Peta tersebut kemudiannya boleh dihantar kepada fungsi-fungsi menulis dan membaca untuk menukar sesuatu struktur kepada XML dan sebaliknya. Cara ini akan membolehkan sesuatu aplikasi menukar struktur-struktur yang berlainan dengan pengaturcaraan struktur khusus yang minimum.

Sebagai contoh, pertimbangkan struktur dalam Figure 5. Untuk menukar struktur Item_t, sesuatu aplikasi akan menghasilkan peta penukaran bagi struktur tersebut dengan templat XmlMap_c dan kemudian memanggil fungsi AddInt dan AddString untuk mentakrif pembolehubah-pembolehubah dalam Item_t. Setelah peta penukaran ditakrif, aplikasi tersebut akan memanggil WriteXml atau ReadXml untuk menukar sesuatu Item_t kepada XML atau sebaliknya. Contoh XML yang setara dengan struktur Item_t ditunjukkan di Figure 6.

Konsep yang sama juga boleh dibina dalam Java seperti yang ditunjukkan di Figure 7. Dalam kes ini, dua argumen kepada addInt dan addString merujuk kepada nama elemen XML dan nama pembolehubah dalam kelas Java. Nama pembolehubah dalam kelas Java boleh dihantar ke Class.getDeclaredField untuk memperolehi sesuatu nilai jenis Field yang merujuk kepada pembolehubah dalam kelas yang perlu ditukar.

Tugas anda ialah mengaturcara satu penukar XML berdasarkan konsep pengurus projek tersebut.

Penukar XML tersebut mesti berupaya untuk menukar struktur C++ atau kelas-kelas Java yang mengandungi pembolehubah-pembolehubah integer dan string kepada XML yang setara dan juga sebaliknya.

Penukar XML tersebut mesti berupaya untuk mentakrif peta penukaran yang menentukan hubungan di antara struktur yang perlu ditukar dengan XML yang setara dengannya.

Penukar XML tersebut mesti mempunyai fungsi-fungsi yang boleh dipanggil oleh aplikasi dan menerima satu rujukan kepada peta penukaran. Penukar XML tersebut kemudiannya menukar sesuatu struktur kepada XML yang setara dengannya atau menukar sesuatu XML kembali kepada struktur yang setara dengannya.

Untuk sekarang, Penukar XML tersebut hanya perlu menukar pembolehubah-pembolehubah integer dan string sahaja, tetapi Penukar XML tersebut mesti direka bentuk supaya ia boleh dinaiktaraf untuk membolehkan penukaran pembolehubah yang berlainan jenis.

Penukar XML tersebut tidak perlu menukar aksara-aksara khas dalam string kepada urutan dalam XML. Contohnya, penukar tersebut tidak perlu menukar ‘&’ kepada ‘&’. Penukar XML tersebut boleh menganggap bahawa string tidak akan mengandungi mana-mana aksara-aksara khas.

Penukar XML tersebut tidak perlu menjana atau dekod permulaan XML ‘<?xml version=“1.0” ?>’.

Penukar XML tersebut harus mengabaikan ruang kosong (aksara-aksara ruang, tab dan baris baru) di antara elemen-elemen.

Fungsi yang menukar XML kembali kepada format struktur harus berupaya untuk memberitahu aplikasi jika XML adalah tidak sah dan harus menerima elemen-elemen XML dalam mana-mana susunan.

Penukar XML tersebut tidak perlu mengikut secara tepat kepada antaramuka pengaturcaraan seperti ditunjukkan dalam Figure 5 dan Figure 7. Anda bebas untuk menentukan fungsi-fungsi yang digunakan dalam pentakrifan peta penukaran dan penukaran struktur asalkan keperluan wajib yang dibentangkan dalam perenggan-perenggan sebelum ini dipenuhi.

```
// Struktur yang boleh ditakrif

struct Item_t {
      int         itemNo;
      string      itemName;
      int         quantity;
};

// Program boleh memetakan penukaran XML seperti berikut:

XmlMap_c<Item_t> itemMap;
itemMap.AddInt ("itemNo", &Item_t::itemNo);
itemMap.AddString ("itemName", &Item_t::itemName);
itemMap.AddInt ("quantity", &Item_t::quantity);

// Peta tersebut boleh digunakan oleh fungsi berikut untuk menulis
// XML yang ditukar daripada nilai Item_t ke output am

WriteXml (&cout, "item", &item, &itemMap);

// Dan program boleh mengunakan fungsi beriku untuk membaca XML
// daripada input piawai untuk ditakrif kepada nilai Item_t

ReadXml (&cin, "item", &item, &itemMap);
```

Gambarajah 18 – Contoh fungsi-fungsi penukaran XML dalam C++

```
<item>
    <itemNo>1234</itemNo>
    <itemName>Widget</itemName>
    <quantity>7</quantity>
</item>
```

Gambarajah 19 – Contoh struktur Item_t yang ditukar kepada XML

```
// Struktur yang boleh ditakrif

public class Item {
      int         itemNo;
      String      itemName;
      int         quantity;
}

// Program boleh memetakan penukaran XML seperti berikut:

XmlMap itemMap;
itemMap = new ItemMap(Item.class);
itemMap.addInt ("itemNo", "itemNo");
itemMap.addString ("itemName", "itemName");
itemMap.addInt ("quantity", "quantity");

// Peta tersebut boleh digunakan oleh fungsi berikut untuk menulis
// XML yang ditukar daripada nilai Item ke output am

XmlWriter.writeXml (System.out, "item", item, itemMap);

// Dan program boleh mengunakan fungsi beriku untuk membaca XML
// daripada input piawai untuk ditakrif kepada nilai Item

XmlReader.readXml (System.in, "item", item, itemMap);
```

Gambarajah 20 – Contoh fungsi-fungsi penukar XML dalam Java

7 STATUS LAPORAN

Satu program laporan menerima ungkapan pemilihan seperti contoh dalam Figure 8 dan menukarnya kepada struktur ungkapan seperti dalam Figure 9. Program tersebut menggunakan ungkapan pemilihan tersebut untuk memilih data yang harus dimasukkan ke dalam laporan. Kekunci pemilihan termasuklah ‘date’, ‘colour’ dan ‘type’.

Dalam pangkalan data, data baru adalah berada dalam status awal dan tertakluk kepada perubahan. Manakala data lama adalah muktamad dan jarang berubah. Khasnya, data dengan tarikh sehingga dan termasuk ‘commit date’ adalah muktamad, manakala data dengan tarikh selepas ‘commit date’ adalah dalam status awal.

Keperluan laporan menyatakan bahawa jika laporan tersebut mengandungi mana-mana data selepas ‘commit date’, ia mesti dilabel dengan ‘preliminary’, jika tidak ia mesti dilabel dengan ‘final’.

Tugas anda ialah menulis satu fungsi yang menerima satu penunjuk ungkapan dan ‘commit date’ dan mengembalikan satu nilai Boolean yang benar jika laporan berada dalam status awal atau palsu jika laporan telah muktamad.

Fungsi anda harus menganggap bahawa status awal atau muktamad adalah tidak bergantung kepada kunci-kunci bukan tarikh. Contohnya, jika ‘commit date’ adalah 18 Oktober 2011 dan ungkapan pemilihan adalah seperti dalam Figure 8, maka laporan tersebut adalah dalam status awal tanpa mengira data yang dimasukkan atau diabaikan oleh syarat ‘color = ‘green’’.

Walau bagaimanapun, fungsi anda mesti menganalisa ungkapan kompaun seperti dalam Figure 10 secara betul, yang di mana akan menghasilkan laporan dengan status awal bagi ‘commit date’ 17 Oktober 2011.

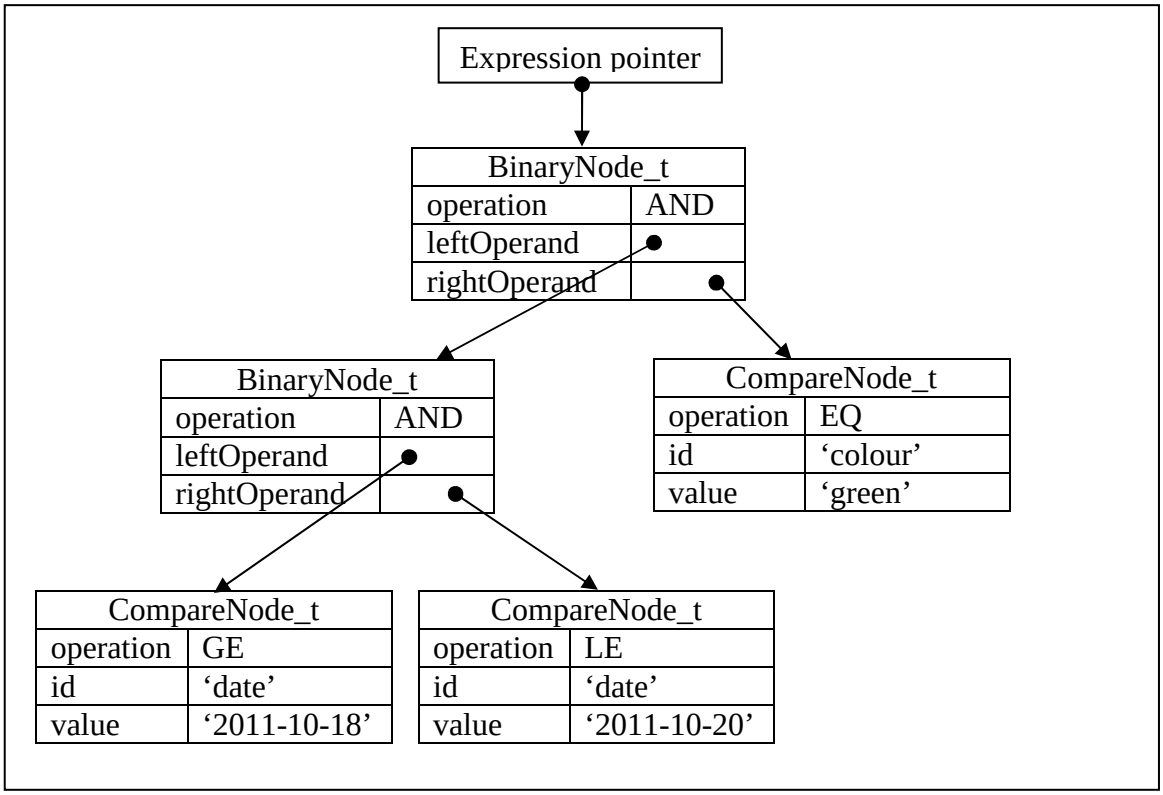
Nod-nod ungkapan adalah ditakrif oleh pengisytiharan C++ dalam Figure 11 atau pengisytiharan Java dalam Figure 12. Operasi-operasi tersebut berfungsi seperti dijelaskan dalam Table 1.

Semua tarikh disimpan dalam string berformat YYYY-MM-DD. Fungsi anda boleh menganggap bahawa tarikh yang diterima telah diformat secara betul dan merupakan tarikh yang sah.

Pengaturcara-pengaturcara C dan C++ boleh menggunakan fungsi-fungsi tarikh dan masa dalam Figure 3.

```
(date >= '2011-10-18' and date <= '2011-10-20') and colour = 'green'
```

Gambarajah 21 – Contoh Ungkapan Pemilihan



Gambarajah 22 – Struktur Ungkapan

```
(colour = 'yellow' and (date >= '2011-10-15' and date <= '2011-10-17'))
                                    or
 (type = 'premium' and (date >= '2011-10-16' and date <= '2011-10-19'))
```

Gambarajah 23 – Contoh Ungkapan Kompaun

```
enum Operation_t {EQ, NE, GT, GE, LT, LE, AND, OR, NOT};

struct Node_t {
      Operation_t       operation;
};

struct CompareNode_t : public Node_t {
      string            id;
      string            value;
};

struct BinaryNode_t : public Node_t {
      Node_t            *leftOperand;
      Node_t            *rightOperand;
};

struct UnaryNode_t : public Node_t {
      Node_t            *operand;
};
```

Gambarajah 24 – Pengisytiharan Nod Ungkapan dalam C++

```
public class Expr {
      public static enum Operation {
            EQ, NE, GT, GE, LT, LE, AND, OR, NOT}

      public static class Node {
            Operation         operation;
      }

      public static class CompareNode extends Node {
            String            id;
            String            value;
      }

      public static class BinaryNode extends Node {
            Node              leftOperand;
            Node              rightOperand;
      }

      public static class UnaryNode extends Node {
            Node              operand;
      }
}
```

Gambarajah 25 – Pengisytiharan Nod Ungkapan dalam Java

Table 2 – Jenis Nod Operasi dan nilai-nilai

Operasi	Jenis Nod	Nilai Nod
EQ	CompareNode_t	Benar jika dan hanya jika nilai kunci pemilihan ‘id’ bersamaan dengan string literal ‘value’.
NE	CompareNode_t	Benar jika dan hanya jika nilai kunci pemilihan ‘id’ tidak bersamaan dengan string literal ‘value’.
GT	CompareNode_t	Benar jika dan hanya jika nilai kunci pemilihan ‘id’ lebih besar daripada string literal ‘value’.
GE	CompareNode_t	Benar jika dan hanya jika nilai kunci pemilihan ‘id’ lebih besar daripada atau bersamaan dengan string literal ‘value’.
LT	CompareNode_t	Benar jika dan hanya jika nilai kunci pemilihan ‘id’ lebih kecil daripada string literal ‘value’.
LE	CompareNode_t	Benar jika dan hanya jika nilai kunci pemilihan ‘id’ lebih kecil daripada atau bersamaan dengan string literal ‘value’.
AND	BinaryNode_t	Benar jika dan hanya jika nilai struktur-struktur ungkapan dalam ‘leftOperand’ dan ‘rightOperand’ adalah turut benar.
OR	BinaryNode_t	Benar jika dan hanya jika salah satu nilai struktur-struktur ungkapan dalam ‘leftOperand’ dan ‘rightOperand’ adalah benar.
NOT	UnaryNode_t	Benar jika dan hanya jika nilai struktur ungkapan dalam ‘operand’ adalah palsu.

8 FUNGSI PUNCA KUASA DUA

Anda ditugaskan kepada satu projek untuk membangunkan satu aplikasi untuk satu pemproses mikro. Pemproses mikro itu mempunyai arimetik titik terapung (floating point), tetapi tidak mempunyai perpustakaan matematik. Untuk menilai sesetengah parameter statistik, project tersebut memerlukan satu fungsi punca kuasa dua yang mudah dan boleh diharap.

Atas sebab-sebab yang tidak diketahui, pengurus projek mengambil keputusan untuk melaksanakannya dengan algoritma cuba-cuba. Algoritma tersebut adalah seperti berikut:

Untuk nombor-nombor yang lebih besar daripada satu, punca kuasa dua nombor tersebut sentiasa berada dalam lingkungan satu hingga nombor tersebut. Seterusnya, sebagai permulaan, fungsi tersebut mesti menetapkan had bawah kepada satu dan had atas kepada nombor tersebut. Ia kemudian mesti menentukan nilai tengah di antara had bawah dan

had atas dan menghitung kuasa dua untuk nilai tengah tersebut. Jika kuasa dua bagi nilai tengah tersebut adalah lebih besar daripada nombor tersebut, maka fungsi tersebut mesti mengubah had atas kepada nilai tengah tersebut. Jika kuasa dua bagi nilai tengah tersebut adalah lebih kecil daripada nombor tersebut, fungsi tersebut mesti mengubah had bawah kepada nilai tengah tersebut. Selepas mengubah kepada had yang sesuai, fungsi tersebut mesti menentukan nilai tengah baru dan mengulangi proses yang sama sehingga ketepatan yang diingini tercapai.

Untuk nombor yang kurang daripada satu, algoritma tersebut adalah sama, kecuali had bawah permulaan adalah nombor tersebut dan had atas adalah satu.

Ketepatan yang diperlukan untuk nombor-nombor titik terapung adalah 8 digit ketara. Disebabkan punca kuasa dua untuk nombor tersebut adalah di antara had atas dan had bawah, ketepatan pada mana-mana ulangan boleh ditentukan melalui pembahagian perbezaan di antara kedua-dua had dengan had bawah. Apabila nilai ini kurang daripada $1/10^8$ sebarang nombor di antara had-had tersebut merupakan anggaran bagi punca kuasa dua untuk nombor tersebut yang mempunyai ketepatan yang diperlu. Untuk mengurangkan ralat, fungsi tersebut harus mengembalikan nilai tengah di antara had-had terakhir yang memenuhi keperluan ketepatan tersebut.

Fungsi tersebut mesti mengembalikan nilai-nilai tepat bagi kes-kes khas yang bernilai sifar dan satu.

Jika sesuatu aplikasi cuba menghitung punca kuasa dua untuk nombor negatif, fungsi punca kuasa dua tersebut harus memaparkan mesej yang sesuai pada saluran kesilapan piawai dan menamatkan program tersebut.

Tugas anda ialah mengaturcara fungsi punca kuasa dua tersebut.

Fungsi punca kuasa dua tersebut mesti mempunyai pengisytiharan seperti dalam Figure 13.

For C and C++:

```
double SquareRoot (double x)
{
     /*...*/
}
```

For Java:

```
public class MicroMath {
     public static double SquareRoot (double x)
     {
          /*...*/
     }
}
```

Gambarajah 26 – Pengisytiharan Fungsi Punca Kuasa Dua