E-GENTING
PROGRAMMING COMPETITION 2012

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.	Offer Server	500
2.	Ticket Financial Summary	200
3.	Network-Synchronised Clock	350
4.	Reset Controller	100

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 OFFER SERVER

The Aggressive Banking Company (ABC) has several hundred automatic teller machines (ATMs) scattered around the metropolis. ABC would like to use the ATMs to help its business clients market their products and services.

In exchange for a fee, ABC would like to be able to show a screen similar to that in Figure 1 when a targeted customer logs in to an ATM.

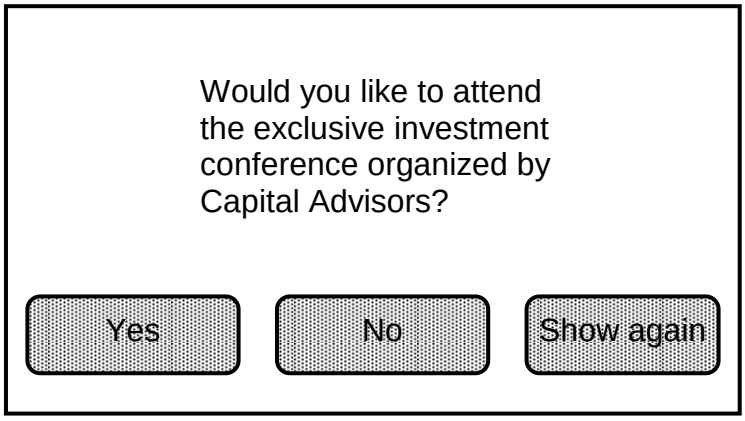


Figure 1 - Offer Screen

If the customer presses the 'Yes' or 'No' button, ABC would like to send a message to the business client's server (BCS) to tell the client that the customer has accepted or declined the offer. The BCS will then determine whether or not it can accept the customer's reply (for example, the BCS may need to reject a customer's acceptance if all the places in the conference are taken) and send an answer back to the ATM to tell the customer whether a place has been reserved or not.

The overall structure of the proposed system is represented in Figure 2.

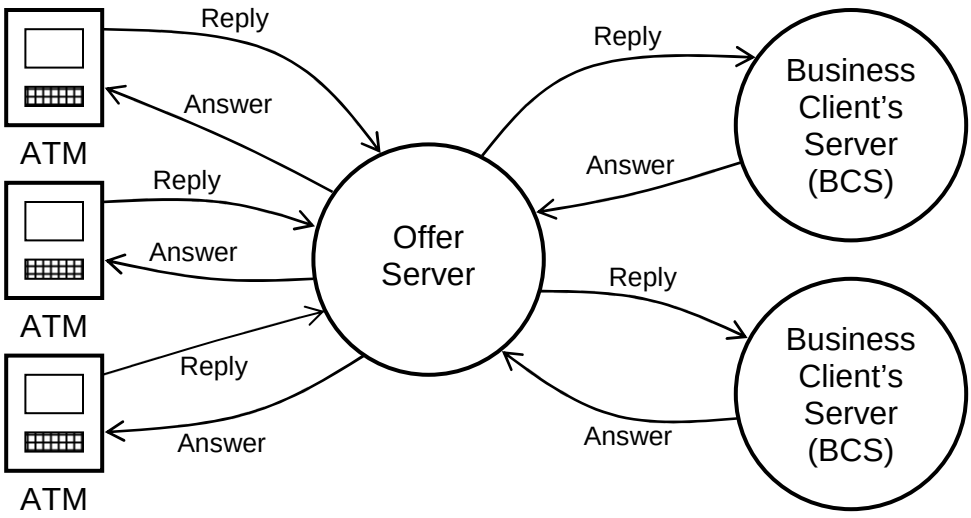


Figure 2 - Offer Processing

Your task is to program an Offer Server class that relays each reply from the ATMs to the appropriate BCS and then receives the answer from the BCS and relays the answer back to the ATM that submitted the original reply.

There may be multiple ATMs, multiple BCSs and each ATM may show multiple offers to a customer.

The configuration of offers shown to customers is passed to the Offer Server class in an array. Each of the array elements has the structure depicted in Figure 3.

C++	Java	C#

`struct Offer_t {`
`  long  offerCode;`
`  long  customerId;`
`  long  bcsId;`
`};`
	`public class Offer {`
	`  int  offerCode;`
	`  int  customerId;`
	`  int  bcsId;`
	`}`
		`public class Offer {`
		`  public int  offerCode;`
		`  public int  customerId;`
		`  public int  bcsId;`
		`}`

`offerCode`
is an offer code that distinguishes different offers made to the same customer. If the same offer is made to multiple customers, each of the offers may have the same offer code. The composite key, offerCode and customerId, uniquely identifies an individual offer made to a specific customer.

`customerId`
is a customer identifier that uniquely identifies a customer.

`bcsId`
is a business client's server identifier that uniquely identifies the BCS.

Figure 3 - Offer Structure

The Offer Server class, to be written by you, must have the external interface depicted in Figure 4.

The Offer Communications class, which has been created by other programmers and can be used by the Offer Server class to send messages to the ATMs and BCSs, has the external interface depicted in Figure 5.

When the Offer Server receives a reply from an ATM, it must look up the BCS identifier from the offer array and relay the reply to the BCS identified by the BCS identifier. When the Offer Server receives an answer from the BCS, it must relay the answer back to the ATM that originally sent the reply. This basic flow is depicted in Figure 6.

To overcome unreliable communications, the ATMs may retransmit replies if the Offer Server does not send an answer in a reasonable period of time. If the Offer Server receives a reply that contains the same offer code and customer identifier as a previous reply, it must process the reply differently depending on whether or not it is still in the process of obtaining an answer from the BCS. If the Offer Server is still in the process of obtaining an answer from the BCS, it should quietly ignore the repeated reply and not relay the repeated reply to the BCS. If the Offer Server has received an answer from the BCS, it must resend the answer to the ATM.

The Offer Server should fully validate replies received from the ATMs and should reject invalid replies by passing a rejection message back to the ATM via the `SendToATM` method.

C++

```
class OfferServer_c {
    // Private declarations to be defined
    // by the programmer
public:
    void Initiate (
        class OfferComms_c *offerComms
        const Offer_t offerArr,
        size_t offerCnt);
                    // Initiate the Offer
                    // Server
    void Tick ();
                    // Process a one-second
                    // clock tick
    void ReplyReceived (long atmId,
        long offerCode, long customerId,
        bool offerAccepted);
                    // Process a reply
    void AnswerReceived (long bcsId,
        long offerCode, long customerId,
        bool replyAccepted,
        const char *reason);
                    // Process an answer
};
```

Java

```
public class OfferServer {
    // Private declarations to be defined
    // by the programmer
    public void initiate (
        OfferComms offerComms
        Offer [] offerArr,
        int offerCnt)
                    // Initiate the Offer
                    // Server
    public void tick ()
                    // Process a one-second
                    // clock tick
    public void replyReceived (int atmId,
        int offerCode, int customerId,
        bool offerAccepted)
                    // Process a reply
    public void answerReceived (int bcsId,
        int offerCode, int CustomerId,
        bool replyAccepted,
        String reason)
                    // Process an answer
}
```

C#

```
public class OfferServer {
    // Private declarations to be defined
    // by the programmer
    public void Initiate (
        OfferComms offerComms
        const Offer [] offerArr,
        int offerCnt);
                    // Initiate the Offer
                    // Server
    public void Tick ();
                    // Process a one-second
                    // clock tick
    public void ReplyReceived (int atmId,
        int offerCode, int customerId,
        bool offerAccepted);
                    // Process a reply
    public void AnswerReceived (int bcsId,
        int offerCode, int customerId,
        bool replyAccepted,
        const string reason);
                    // Process an answer
};
```

`Initiate/initiate`
: is called when the Offer Server is initiated.

`Tick/tick`
: is called once each second as time elapses.

`ReplyReceived/replyReceived`
: is called when the Offer Server receives a reply from an ATM.

`AnswerReceived/answerReceived`
: is called when the Offer Server receives an answer from a BCS.

`offerComms`
: is a pointer or a reference to a class that the Offer Server class can use to send messages to BCSs and ATMs.

`offerArr`
: is a pointer or reference to the offer array.

`offerCnt`

is the number of offers in the offer array.

`atmId`
is a 32-bit integer that uniquely identifies an ATM.

`offerCode`
is an offer code that identifies the offer and is in the same domain as the `offerCode` column in the database.

`customerId`
is a customer identifier that uniquely identifies the customer and is in the same domain as the `customerId` column in the database.

`offerAccepted`
is a Boolean value that is true if the customer accepted the offer and false if the customer declined the offer.

`bcsId`
is a business client's server identifier that uniquely identifies the BCS and is in the same domain as the `bcsId` column in the database.

`replyAccepted`
is a Boolean value that is true if the BCS accepted the customer's reply and false if the BCS rejected the customer's reply.

`reason`
is a string that explains why the BCS rejected the customer's reply. It is only defined if the BCS rejected the customer's reply. If the BCS accepted the customer's reply it may be null or it may refer to arbitrary data that should not be accessed.

Figure 4 - Offer Server Class Declaration

C++

```
class OfferComms_c {
public:
    void SendToBCS (
            long bcsId, long atmId,
            long offerCode,long customerId,
            bool offerAccepted);
                    // Send a reply to a
                    // business client's
                    // server
    void SendToATM (long atmId,
            long offerCode,long customerId,
            bool replyAccepted,
            const char *reason);
                    // Send an answer to
                    // an ATM
};
```

Java

```
public class OfferComms {
    public void sendToBCS (
            int bcsId, int atmId,
            int offerCode, int customerId,
            bool offerAccepted)
                    // Send a reply to a
                    // business client's
                    // server
    public void sendToATM (int atmId,
            int offerCode, int customerId,
            bool replyAccepted,
            String reason)
                    // Send an answer to
                    // an ATM
}
```

C#

```
class OfferComms {
    public void SendToBCS (
            int bcsId, int atmId,
            int offerCode, int customerId,
            bool offerAccepted);
                    // Send a reply to a
                    // business client's
                    // server
    public void SendToATM (int atmId,
            int offerCode, int customerId,
            bool replyAccepted,
            const string reason);
                    // Send an answer to
                    // an ATM
};
```

`SendToBCS/sendToBCS`
sends a reply to a business client's server.

`SendToATM/sendToATM`
sends an answer to an ATM.

`bcsId, atmId, offerCode, customerId, offerAccepted, replyAccepted and reason`
have the same meanings as the parameters with the same names in the Offer Server class.

Figure 5 - Offer Communications Class

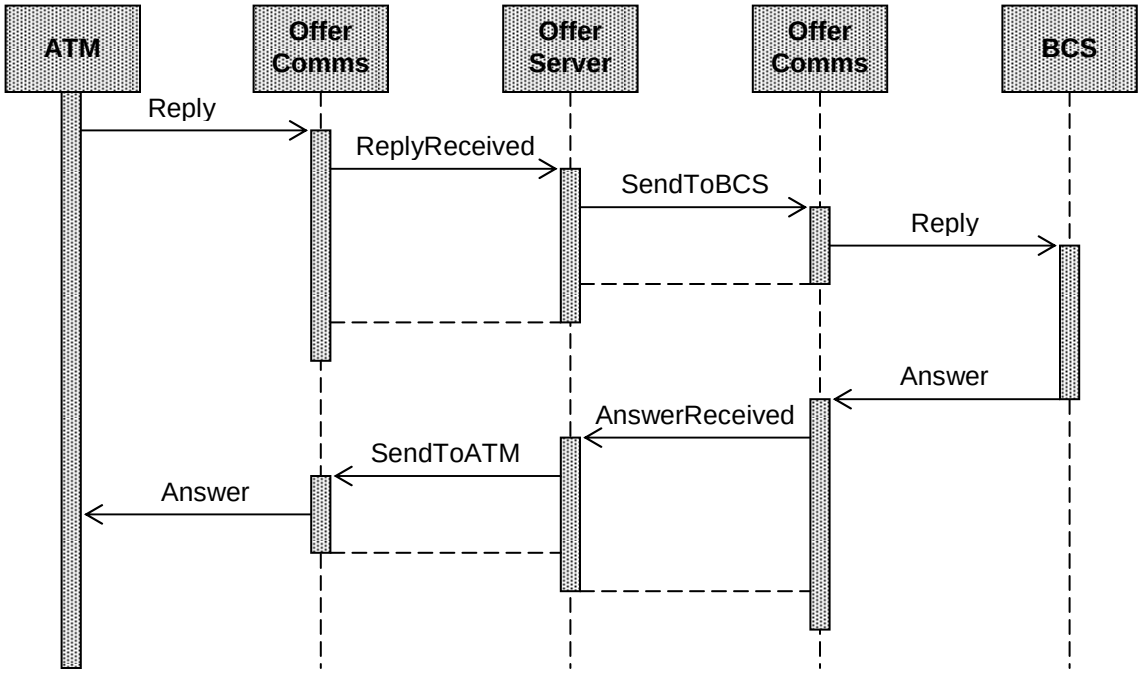


Figure 6 - Sequence Diagram of Basic Flow

If the Offer Server does not receive an answer within 30 seconds from the time when it relayed a reply to a BCS, it must resend the reply to the BCS.

As a consequence of repeatedly sending replies to a BCS, a BCS may send repeated answers. If the Offer Server receives an answer that it has already received, it must quietly ignore the repeated answer and not relay it to the ATM.

All messages may be arbitrarily delayed in the communications infrastructure for periods of up to 5 minutes and may be delivered out-of-order. For example, an answer from a BCS may be delayed. As a consequence of the delay the Offer Server may resend the reply and the BCS may resend the answer. The retransmission of the answer may not be delayed and may be received immediately. The Offer Server may then communicate with the BCS in relation to another offer from another customer and then after all that has been completed, the original answer may be delivered late and out-of-sequence. In these situations, the Offer Server must quietly ignore the late and out-of-sequence answer.

The Offer Server should fully validate all answers received from the BCSs. If an answer is found to be invalid, the Offer Server should log a message to the error output stream and in other respects ignore the answer.

The Offer Server should process replies sent from different ATMs in parallel. For example, if it receives a reply from ATM 1 and relays it to BCS 23 and then before it receives an answer, receives another reply from ATM 2 to be relayed to BCS 24, then the Offer Server should send the second reply to BCS 24 without waiting for the answer from BCS 23.

Nevertheless, if the Offer Server receives two replies that need to be relayed to the same BCS, it must queue the replies to the BCS and not send a queued reply until an answer is received for the preceding reply.

The version of the Offer Server that you are to write is not required to withstand arbitrary shutdown and restart. You may assume that once started the Offer Server continues to run until all the offers are dealt with in some way and that when new offers are to be made the Offer Server is shut down and restarted with the new offers in a new offer array.

2 TICKET FINANCIAL SUMMARY

A theme park uses a ticket-based cash management system to facilitate the acceptance of cash at each ride while avoiding the inconvenience and risk associated with having to dispense change.

To take a ride, a customer can either insert cash or a ticket dispensed by a previous ride into a bill validator at the turnstile that controls entry to the ride. In either case, the turnstile provides any change in the form of a ticket that can be inserted in another ride or converted into cash at one of the automatic teller machines (ATM) located near the theme park exit.

If a ticket is lost or damaged, the theme park may cancel the ticket and give the customer a refund.

If a ticket is not used for 6 months, the ticket automatically expires.

Your task is to write a Ticket Financial Summary that summarises ticket activity at the theme park.

The reporting program must accept the following parameters:

1. from-date;
2. to-date.

The Ticket Financial Summary must contain the following information:

1. the reporting period (from-date and to-date);
2. if any trading day in the reporting period is still open for further transactions, the word ‘PRELIMINARY’, otherwise the word ‘FINAL’;
3. the number of transactions and the total value of the transactions of each of the following for the reporting period:
    a. opening unredeemed tickets;
    b. tickets issued by rides;
    c. tickets accepted by rides;
    d. tickets cashed at ATMs;
    e. cancelled tickets;
    f. expired tickets;
    g. closing unredeemed tickets.

The Ticket Financial Summary should be laid out in accordance with the example in Figure 7.

```
PRELIMINARY/FINAL TICKET FINANCIAL SUMMARY
FOR DD-MM-YY TO DD-MM-YY
                                      Number            Value

Opening unredeemed tickets             1,520        18,240.40
Tickets issued by rides               22,920       340,227.27
Tickets accepted by rides             18,517       292,420.85
Tickets cashed at ATMs                 3,927        40,650.96
Cancelled tickets                          5            72.24
Expired tickets                          120           927.72
                                      -------     -----------
Closing unredeemed tickets             1,871        24,395.90
```

Figure 7 - Ticket Financial Report Example

The reporting program must extract the information it needs to produce the Ticket Financial Summary from the theme park database. The database contains the following tables that are relevant to the report:

```
create table dayStatus (
     tradingDate    date not null,
     dayOpen        smallint not null
);
create table tickets (
     ticketNo       integer not null,
     ticketStatus   smallint not null,
     ticketValue    double precision not null
);
create unique index ticketNoInd on tickets (ticketNo);
create table transactions (
     txNo           integer not null,
     ticketNo       integer not null,
     locationId     integer not null,
     txTime         timestamp not null,
     txDate         date not null,
     txType         smallint not null
);
create index txDateInd on transactions (txDate);
```

`dayStatus`
: is a table that contains a single row for the current trading day.

`dayStatus.tradingDay`
: is the date of the current trading day. Sometimes the theme park stays open until the early hours of the next morning. In these situations the transactions that occur in the early hours of the next morning are posted to the previous trading day.

`dayStatus.dayOpen`
: is zero if the current trading day is closed or one if the current trading day is open.

`tickets`
: is a table that stores the current status of each ticket.

`tickets.ticketNo`
: is a ticket number that uniquely identifies the ticket.

`tickets.ticketStatus`
: is a ticket status code. The ticket status codes are listed in Table 1.

`tickets.ticketValue`
: is the value of the ticket in cents. For example, $12.34 is stored as 1234.0.

`ticketNoInd`
: is a database index on `tickets.ticketNo` that lets a database user rapidly find a `tickets` row from a known ticket number.

`transactions`
: is a table that stores information about transactions that affect tickets.

`transactions.txNo`
: is a transaction number that uniquely identifies the transaction. Transaction numbers are always allocated in the sequence in which the transactions occur.

`transactions.ticketNo`
: is the ticket number of the ticket to which the transaction relates. It is in the same domain as `tickets.ticketNo`.

`transactions.locationId`
: is a location identifier that uniquely identifies the ride, ATM or cashier that processed the transaction.

`transactions.txTime`
: is the time at which the transaction occurred.

`transactions.txDate`
: is the trading day date on which the transaction occurred.

`transactions.txType`
: is a transaction type code. The transaction type codes are listed in Table 2.

`txDateInd`
: is a database index on transaction date that lets a database user rapidly select rows from the `transactions` table from a transaction date key.

Table 1 – Ticket Status Codes

Code	Meaning
1	The ticket has been issued by a ride but has not yet been redeemed.
2	The ticket has been accepted by a ride.
3	The ticket has been cashed at an ATM.
4	The ticket has been cancelled.
5	The ticket has expired.

Table 2 - Transaction Type Codes

Code	Meaning
1	Issue by a ride
2	Acceptance by a ride
3	Encashment at an ATM
4	Cancellation
5	Expiry

The reporting program should make use of the database indexes, where appropriate, to minimise the execution time of the reporting program.

Programmers writing in C or C++ may make use of the date conversion functions described in Figure 8. Java and C# programmers should use the date conversion functions provided by the Java and C# 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 8 – Double Time Functions

3 NETWORK-SYNCHRONISED CLOCK

A publically accessible time server accepts HTTP requests of the form shown in Figure 9 and returns an HTTP response of the form shown in Figure 10.

The examples in Figure 9 and Figure 10 include the HTTP header and in the examples, the codes '\r' and '\n' represent the carriage return and new line characters respectively.

A typical browser would show the HTTP response in the form shown in Figure 11.

Users of the time server have observed that it can be difficult to identify the error in the returned time because the user can't tell whether the delay is from the request being transmitted to the server or the response being returned to the client.

What the users would like to see is a dynamic clock of the kind shown in Figure 12.

```
GET /TIME HTTP/1.1\r\n
User-Agent: TimeClient\r\n
Host: time.rwgenting.com:2212\r\n
Connection: close\r\n
\r\n
```

Figure 9 - HTTP Time Request

```
HTTP/1.1 200 OK\r\n
Server: TimeServer\r\n
Date: Wed, 26 Sep 2012 06:00:30 GMT\r\n
Accept-ranges: none\r\n
Content-type: text/plain\r\n
Connection: close\r\n
Content-length: 65\r\n
\r\n
<html>\r\n
<body>\r\n
Wed, 26 Sep 2012 14:00:30 MYT\r\n
</body>\r\n
</html>\r\n
```

Figure 10 - HTTP Time Response

Wed, 26 Sep 2012 14:00:30 MYT

Figure 11 – Typical Browser Display

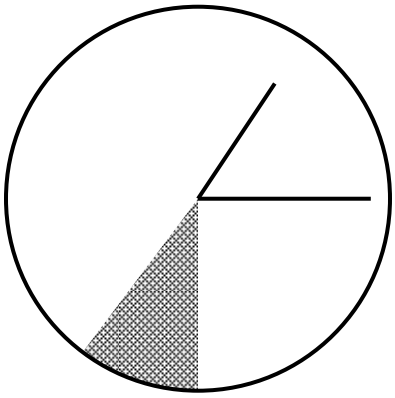


Figure 12 - Clock Face

The clock face in Figure 12 shows that the time is somewhere between 1:15 and 30 seconds and 1:15 and 37 seconds.

Your task is to create a client program that queries the time server and renders the clock face.

The client program must record the time from the local client clock immediately before it sends a HTTP time request to the time server (the 'send time') and then record the time again immediately after it receives the HTTP time response (the 'receive time'). The time elapsed between the send time and the receive time (the 'time lag') is the potential error in the value of the time returned from the time server (the 'server time'). I.e. the real time is somewhere between the server time and the time returned by the time server plus the time lag. The client program must show the seconds component of this time range as the gray sector on the clock face.

Having obtained the server time, calculated the time lag and rendered the clock face, the client program must advance the time shown on the clock face every second by moving the gray sector in one-second shifts around the clock face and the hour and minute hands in the same way as a conventional clock.

If the time lag is 60 seconds or more, the client program must paint the entire clock face gray.

The client program must position the hour and minute hands to the closest minute to the middle position between the server time, as advanced by the elapse of time, and the server time plus the time lag.

Having obtained an initial time value, the client program must symmetrically increase the displayed time range to allow for a potential error of ±10 seconds per day in the precision of the clock on the client computer.

After receiving the server time from the time server, rendering the clock face and initiating the one-second refresh, the client program must rest for 60 seconds and then send another HTTP time request to the time server. On receipt of the HTTP time response, the client program must calculate the intersection between the time range derived from the new request and the displayed time range as adjusted for the error in the

precision of the clock on the client computer and set the displayed time range to the intersection.

If the time range derived from the new request and the displayed time range do not intersect, the client program must assume that the displayed time range is defective and must set the displayed time range to the time range derived from the new request.

The client program must repeat the above process 60 seconds after receiving each HTTP time response.

If the client program is unable to obtain an HTTP time response from the time server (for example, because the time server cannot be contacted or is off line), the client program must continue to symmetrically increase the displayed time range to allow for the imprecision of the clock on the client computer and wait for a further 60 seconds and try to contact the time server again. The client program must repeat this process, waiting 60 seconds between retries, until the client program successfully contacts the time server. When the client program successfully contacts the time server, it must process the response in the manner described in the preceding paragraphs.

4 RESET CONTROLLER

Maintenance engineers working on an established fleet of medium size passenger ferries have found that on some occasions they are unable to download data from the data loggers on the ferries. With a rise in grounding and collision incidents, they are concerned that critical data might be lost. They have traced the problem to power fades while starting the engines. The electric motors that start the engines draw a large amount of power from the ferry batteries and this causes the input power to the data loggers to hover at around the mimimum level for operations. Although the data loggers have been designed to withstand a simple power shutdown, when the power hovers around the minimum level, something goes wrong with the data loggers and sometimes the data stored inside them gets lost.

Rather than replacing all the data loggers, which would be prohibitively expensive, the engineers propose to retrofit a reset controller as shown in Figure 13.

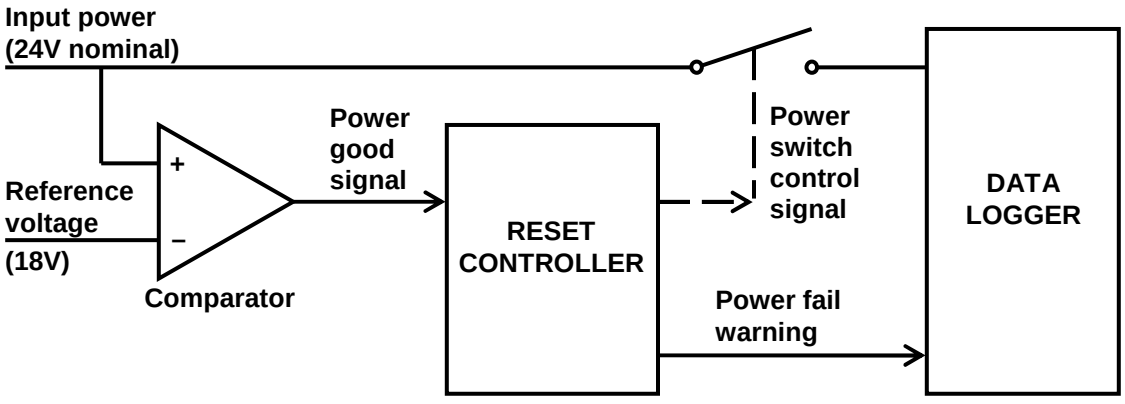


Figure 13 – Reset Controller Schematic

The idea is that the reset controller will compare the nominally 24 Volt input power with an 18 Volt reference. When the input voltage falls below the reference voltage, the reset controller will initiate an orderly shutdown of the data logger to prevent loss of information. Before shutting down the data logger, the reset controller must activate a power fail warning signal which causes the data logger to save critical information in non-volatile memory before shutting down. Your task is to program the reset controller.

When the reset controller is first powered up, it must activate the power fail warning and switch off the data logger by deactivating the power switch control signal. It must then monitor the power good signal and wait for it to go high, which means that the input power is above 18 Volts, which is well above the 12 Volts needed to operate the data logger. When the reset controller detects that the power good signal is in the high state, it must wait a further 3 seconds and then switch on the data logger and deactivate the power fail warning. If the power good signal reverts to the low state while the controller is waiting the further 3 seconds, the reset controller must restart the power up process and wait once again for the power good signal to switch to the high state before waiting another 3 seconds and so on.

Once the data logger has been switched on, the reset controller must poll the power good signal at least once every millisecond. On a high-to-low transition of the power good signal, the reset controller must activate the power fail warning to the data logger and start waiting. If, during the wait, the power good signal returns to the high state, the reset controller must deactivate the power fail warning and return to polling the power good signal. If the power good signal remains low for 20 milliseconds or more, the reset controller must turn off the data logger and initiate the power up sequence described in the previous paragraph.

As a general rule, the reset controller should detect changes in state of the input signals by sampling their values at intervals of one millisecond.

The data controller program must use the functions presented in Figure 14 to access and set the various signals.

C++

```
class ResetIO_c {
    // ...
public:
    static bool IsPowerGood();
                    // Test the power good
                    // signal
    static void SetPowerSwitch (
            bool powerOn);
                    // Turn the power on or
                    // off
    static void SetPowerWarn (
            bool warnOn);
                    // Activate or
                    // deactivate the power
                    // fail warning
};
```

Java

```
public class ResetIO {
    public static boolean isPowerGood()
                    // Test the power good
                    // signal
    public static void setPowerSwitch (
            boolean powerOn)
                    // Turn the power on or
                    // off
    public static void setPowerWarn (
            boolean warnOn)
                    // Activate or
                    // deactivate the power
                    // fail warning
};
```

C#

```
class ResetIO {
    public static bool IsPowerGood();
                    // Test the power good
                    // signal
    public static void SetPowerSwitch (
            bool powerOn);
                    // Turn the power on or
                    // off
    public static void SetPowerWarn (
            bool warnOn);
                    // Activate or
                    // deactivate the power
                    // fail warning
};
```

`IsPowerGood/isPowerGood`
: tests the state of the power good signal. It returns 1 (or true) if the power good signal is high (i.e. the input power is equal or above 18V) or 0 (or false) if the power good signal is low (i.e. the input power is below 18V).

`SetPowerSwitch/setPowerSwitch`
: turns the power switch on and off.

`powerOn`
: is the state the power switch is to be turned to. If it is 1 (or true), the power switch is turned on. If it is 0 (or false), the power switch is turned off.

`SetPowerWarn/setPowerWarn`
: sets the state of the power fail warning signal to the data logger.

`warnOn`
: is the state of the power fail warning. If it is 1 (or true), the power fail warning is activated. If it is 0 (or false) the power fail warning is deactivated.

Figure 14 – Reset Controller I/O Functions

PERTANDINGAN PENGATURCARAAN E-GENTING 2012

Arahan-arahan am:

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

No	Tajuk	Markah
1.	Pelayan Tawaran	500
2.	Rumusan Kewangan Tiket	200
3.	Jam Selaras Rangkaian	350
4.	Pengawal Reset	100

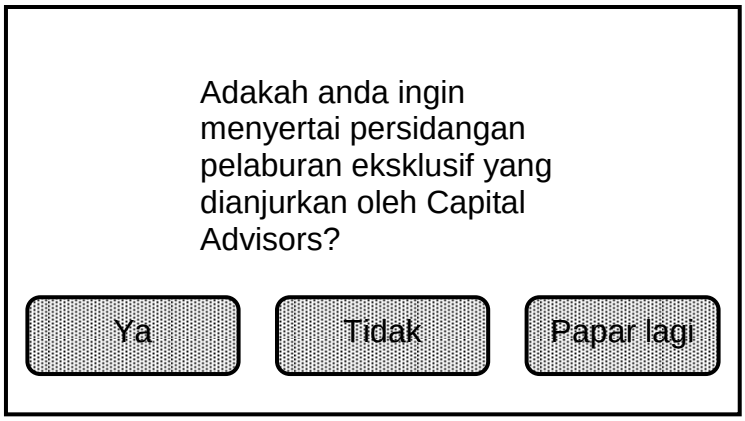
10. Kecuali dinyatakan, program anda boleh ditulis dengan menggunakan mana-mana bahasa pengaturcaraan utama di bawah mana-mana sistem operasi utama.
11. Kecuali dinyatakan, anda boleh menggunakan semua fungsi piawaian perpustakaan dalam bahasa pengaturcaraan dan sistem operasi yang anda pilih.
12. Perkataan-perkataan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

1 PELAYAN TAWARAN

Syarikat Aggressive Banking (ABC) memiliki ratusan mesin wang automatik (ATM) di sekitar bandaraya. ABC ingin menggunakan mesin-mesin tersebut untuk membantu pelanggan-pelanggan perniagaannya memasarkan produk dan perkhidmatan mereka.

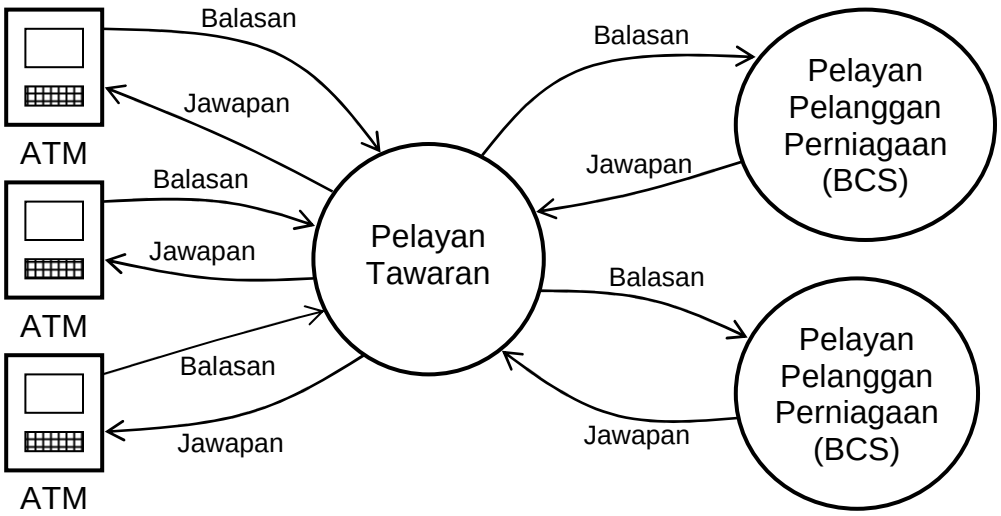
Dengan mengenakan yuran perkhidmatan, ABC ingin memaparkan skrin seperti dalam Gambarajah 1 apabila seseorang pelanggan sasaran mendaftar masuk ke sesebuah ATM.



Gambarajah 1 – Skrin Tawaran

Jika pengguna tersebut menekan butang ‘Ya’ atau ‘Tidak’, ABC ingin menghantar satu mesej kepada komputer pelayan pelanggan perniagaan (BCS) untuk memberitahu bahawa tawaran tersebut telah diterima atau ditolak. BCS kemudian akan menentukan sama ada ia boleh menerima balasan pengguna tersebut (contohnya, BCS boleh menolak seseorang pengguna yang menerima sesuatu tawaran jika semua tempat dalam persidangan tersebut telah dipenuhi) dan membalas ATM tersebut untuk memaklumkan kepada pengguna sama ada tempat telah ditempah untuknya.

Struktur keseluruhan sistem yang dicadangkan adalah seperti dalam Gambarajah 2.



Gambarajah 2 – Pemprosesan tawaran

Tugas anda ialah untuk mengaturcara satu kelas Pelayan Tawaran yang menghantar setiap balasan daripada ATM kepada BCS yang sesuai dan kemudian menerima jawapan daripada BCS dan mengembalikan jawapan tersebut kepada ATM yang menghantar balasan asal tersebut.

Adalah berkemungkinan bahawa terdapat pelbagai ATM dan pelbagai BCS dan setiap ATM mungkin memaparkan pelbagai tawaran kepada seseorang pengguna.

Konfigurasi untuk tawaran yang dipaparkan kepada pengguna dihantar kepada Pelayan Tawaran dalam satu tatasusunan. Setiap elemen dalam tatasusunan mempunyai struktur seperti dalam Gambarajah 3.

C++	Java	C#

`struct Offer_t {`
`  long  offerCode;`
`  long  customerId;`
`  long  bcsId;`
`};`
	`public class Offer {`
	`  int  offerCode;`
	`  int  customerId;`
	`  int  bcsId;`
	`}`
		`public class Offer {`
		`  public int  offerCode;`
		`  public int  customerId;`
		`  public int  bcsId;`
		`}`

`offerCode`
: merupakan satu kod tawaran yang membezakan pelbagai tawaran yang ditunjuk kepada pengguna yang sama. Jika tawaran yang sama ditunjukkan kepada pelbagai pengguna, setiap tawaran tersebut boleh mempunyai kod tawaran yang sama. Kunci komposit, offerCode dan customerId, mengenalpasti sesuatu tawaran yang ditawarkan kepada seseorang pengguna secara unik.

`customerId`
: merupakan id yang mengenalpasti seseorang pengguna secara unik.

`bcsId`
: merupakan id yang mengenalpasti sesebuah pelayan pelanggan perniagaan (BCS) secara unik.

Gambarajah 3 – Struktur Tawaran

Kelas Pelayan Tawaran yang anda perlu hasilkan mesti mempunyai antara-muka luaran seperti dalam Gambarajah 4.

Kelas Komunikasi Tawaran, yang telah dihasilkan oleh pengaturcara lain dan boleh digunakan oleh kelas Pelayan Tawaran untuk menghantar mesej-mesej kepada ATM dan BCS, mempunyai antara-muka luaran seperti dalam Gambarajah 5.

Apabila Pelayan Tawaran tersebut menerima sesuatu balasan daripada sesebuah ATM, ia mesti mencari id BCS dari tatasusunan tawaran dan menghantar balasan tersebut kepada BCS yang dikenalpasti melalui id BCS tersebut. Apabila Pelayan Tawaran tersebut menerima sesuatu jawapan daripada BCS, ia mesti menghantar jawapan tersebut kembali kepada ATM di mana balasan tersebut berasal. Aliran asas ini ditunjukkan dalam Gambarajah 6.

Untuk mengatasi keadaan komunikasi yang tidak stabil, ATM boleh menghantar semula balasan jika Pelayan Tawaran tidak mengembalikan jawapan dalam tempoh masa yang munasabah. Jika Pelayan Tawaran menerima sesuatu balasan yang mengandungi kod tawaran dan id pengguna yang sama seperti mana-mana balasan yang sebelumnya, ia mesti memproses balasan tersebut secara berbeza bergantung kepada sama ada ia masih dalam proses menunggu jawapan daripada BCS atau tidak. Jika Pelayan Tawaran masih dalam proses menunggu jawapan daripada BCS, ia harus mengabaikan balasan berulang tersebut dan tidak menghantar balasan berulang tersebut kepada BCS. Jika Pelayan

Tawaran telah menerima jawapan daripada BCS, ia mesti menghantar semula jawapan tersebut kepada ATM tersebut.

Pelayan Tawaran harus membuat pengesahan sepenuhnya bagi balasan yang diterima daripada ATM dan harus menolak balasan yang tidak sah dengan mengembalikan satu mesej kepada ATM melalui fungsi `SendToATM`.

C++

```
class OfferServer_c {
    // Private declarations to be defined
    // by the programmer
public:
    void Initiate (
        class OfferComms_c *offerComms
        const Offer_t offerArr,
        size_t offerCnt);
                    // Initiate the Offer
                    // Server
    void Tick ();
                    // Process a one-second
                    // clock tick
    void ReplyReceived (long atmId,
        long offerCode, long customerId,
        bool offerAccepted);
                    // Process a reply
    void AnswerReceived (long bcsId,
        long offerCode, long customerId,
        bool replyAccepted,
        const char *reason);
                    // Process an answer
};
```

Java

```
public class OfferServer {
    // Private declarations to be defined
    // by the programmer
    public void initiate (
        OfferComms offerComms
        Offer [] offerArr,
        int offerCnt)
                    // Initiate the Offer
                    // Server
    public void tick ()
                    // Process a one-second
                    // clock tick
    public void replyReceived (int atmId,
        int offerCode, int customerId,
        bool offerAccepted)
                    // Process a reply
    public void answerReceived (int bcsId,
        int offerCode, int CustomerId,
        bool replyAccepted,
        String reason)
                    // Process an answer
}
```

C#

```
public class OfferServer {
    // Private declarations to be defined
    // by the programmer
    public void Initiate (
        OfferComms offerComms
        const Offer [] offerArr,
        int offerCnt);
                    // Initiate the Offer
                    // Server
    public void Tick ();
                    // Process a one-second
                    // clock tick
    public void ReplyReceived (int atmId,
        int offerCode, int customerId,
        bool offerAccepted);
                    // Process a reply
    public void AnswerReceived (int bcsId,
        int offerCode, int customerId,
        bool replyAccepted,
        const string reason);
                    // Process an answer
};
```

`Initiate/initiate`
: dipanggil apabila Pelayan Tawaran dimulakan.

`Tick/tick`
: dipanggil sekali bagi setiap saat yang berlalu.

`ReplyReceived/replyReceived`
: dipanggil apabila Pelayan Tawaran menerima balasan daripada sesebuah ATM.

`AnswerReceived/answerReceived`
: dipanggil apabila Pelayan Tawaran menerima jawapan daripada sesebuah BCS.

`offerComms`
: merupakan satu penunjuk atau rujukan kepada satu kelas yang boleh digunakan oleh Pelayan Tawaran untuk menghantar mesej-mesej kepada BCS dan ATM.

`offerArr`
: merupakan satu penunjuk atau rujukan kepada tatasusunan tawaran.

`offerCnt`
: merupakan bilangan tawaran dalam tatasusunan tawaran.

`atmId`
: merupakan integer 32-bit yang mengenalpasti sesebuah ATM secara unik.

`offerCode`
: merupakan satu kod tawaran yang mengenalpasti tawaran tersebut dan berada dalam domain yang sama dengan lajur `offerCode` dalam pangkalan data.

`customerId`
: merupakan id pengguna yang mengenalpasti seseorang pengguna secara unik dan berada dalam domain yang sama dengan lajur `customerId` dalam pangkalan data.

`offerAccepted`
: merupakan nilai Boolean yang benar jika pengguna menerima tawaran tersebut dan palsu jika pengguna menolak tawaran tersebut.

`bcsId`
: merupakan id Pelayan Pelanggan Perniagaan yang mengenalpasti sesebuah BCS secara unik dan berada dalam domain yang sama dengan lajur `bcsId` dalam pangkalan data.

`replyAccepted`
: merupakan nilai Boolean yang benar jika BCS tersebut menerima balasan pengguna dan palsu jika BCS menolak balasan pengguna.

`reason`
: merupakan satu rentetan yang menjelaskan mengapa BCS menolak balasan pengguna. Ia hanya ditakrif jika BCS menolak balasan pengguna. Jika BCS menerima balasan pengguna, ia boleh bernilai null atau ia boleh merujuk kepada sebarang data yang tidak harus dibaca.

Gambarajah 4 – Pengisytiharan Kelas Pelayan Tawaran

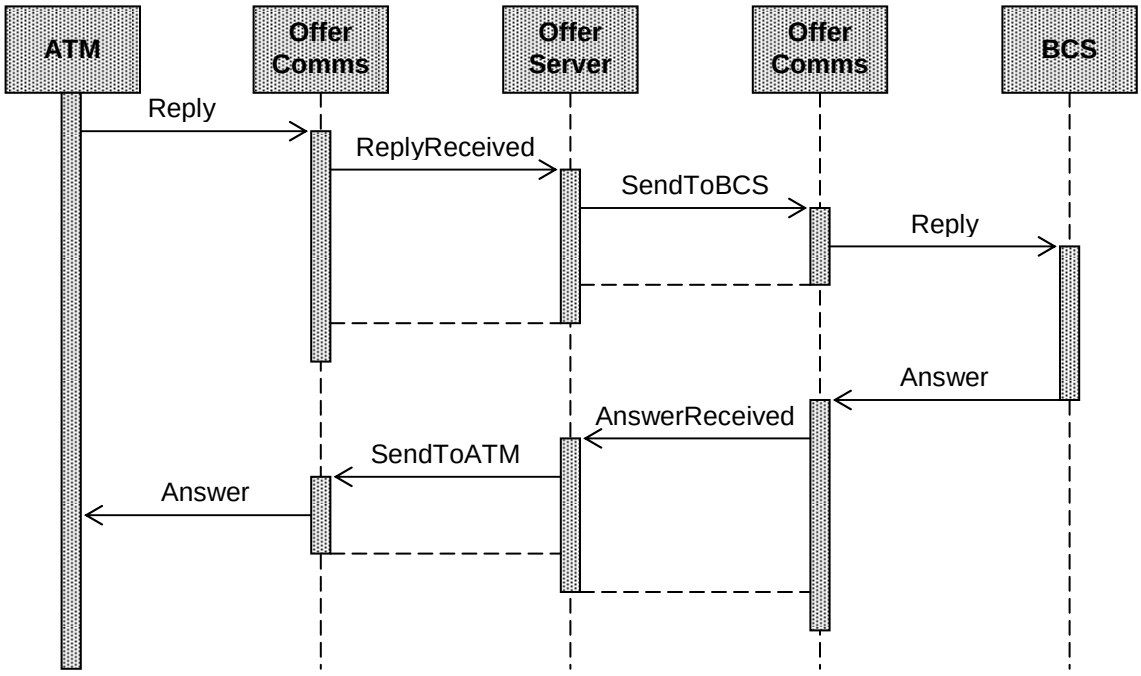
C++	Java

`class OfferComms_c {`
`public:`
`  void SendToBCS (`
`  long bcsId, long atmId,`
`  long offerCode,long customerId,`
`  bool offerAccepted);`
`  // Send a reply to a`
`  // business client's`
`  // server`
`  void SendToATM (long atmId,`
`  long offerCode,long customerId,`
`  bool replyAccepted,`
`  const char *reason);`
`  // Send an answer to`
`  // an ATM`
`};`
	`public class OfferComms {`
	`  public void sendToBCS (`
	`  int bcsId, int atmId,`
	`  int offerCode, int customerId,`
	`  bool offerAccepted)`
	`  // Send a reply to a`
	`  // business client's`
	`  // server`
	`  public void sendToATM (int atmId,`
	`  int offerCode, int customerId,`
	`  bool replyAccepted,`
	`  String reason)`
	`  // Send an answer to`
	`  // an ATM`
	`}`

C#	

`class OfferComms {`
`  public void SendToBCS (`
`  int bcsId, int atmId,`
`  int offerCode, int customerId,`
`  bool offerAccepted);`
`  // Send a reply to a`
`  // business client's`
`  // server`
`  public void SendToATM (int atmId,`
`  int offerCode, int customerId,`
`  bool replyAccepted,`
`  const string reason);`
`  // Send an answer to`
`  // an ATM`
`};`
	`SendToBCS/sendToBCS`
	menghantar sesuatu balasan kepada BCS.
	`SendToATM/sendToATM`
	menghantar sesuatu jawapan kepada ATM.
	`bcsId, atmId, offerCode, customerId, offerAccepted, replyAccepted and reason`
	mempunyai maksud yang sama seperti parameter yang mempunyai nama yang sama dalam kelas Pelayan Tawaran.

Gambarajah 5 – Kelas Komunikasi Tawaran



Gambarajah 6 – Gambarajah Urutan untuk Aliran Asas

Jika Pelayan Tawaran tidak menerima jawapan dalam 30 saat dari masa ia menghantar sesuatu balasan kepada sesebuah BCS, ia mesti menghantar semula balasan tersebut kepada BCS.

Akibat daripada penghantaran balasan berulang kali kepada sesebuah BCS, BCS juga mungkin menghantar jawapan secara berulang. Jika Pelayan Tawaran tersebut menerima sesuatu jawapan yang ia pernah terima, ia mesti mengabaikan jawapan berulang itu dan tidak menghantar jawapan berulang tersebut kepada ATM.

Semua mesej-mesej mungkin ditangguh oleh infrastruktur komunikasi selama 5 minit dan mungkin dihantar dalam urutan yang berbeza. Sebagai contoh, sesuatu jawapan daripada BCS mungkin ditangguhkan. Akibat daripada itu, Pelayan Tawaran akan menghantar semula balasan tersebut dan BCS akan juga menghantar semula jawapan tersebut. Penghantaran semula jawapan tersebut mungkin tidak ditangguh dan diterima segera. Pelayan Tawaran kemudiannya berkomunikasi dengan BCS mengenai tawaran lain daripada pengguna lain dan setelah ianya lengkap, jawapan asal sebelumnya mungkin dihantar lewat dan tidak dalam urutan yang sepatutnya. Dalam situasi sedemikian, Pelayan Tawaran mesti mengabaikan jawapan yang lewat dan tidak dalam urutan yang sepatutnya.

Pelayan Tawaran harus membuat pengesahan sepenuhnya bagi semua jawapan yang diterima daripada BCS. Jika sesuatu jawapan didapati tidak sah, Pelayan Tawaran harus merakam satu mesej ke aliran output kesilapan dan kemudian mengabaikan jawapan tersebut.

Pelayan Tawaran harus memproses balasan-balasan yang dihantar daripada pelbagai ATM secara serentak. Contohnya, jika ia menerima satu balasan daripada ATM1 dan menghantarnya kepada BCS23 dan sebelum ia menerima jawapan, menerima satu lagi balasan dari ATM2 yang perlu dihantar ke BCS24, maka Pelayan Tawaran tersebut harus menghantar balasan kedua kepada BCS24 tanpa menunggu jawapan daripada BCS23.

Namun begitu, jika Pelayan Tawaran menerima dua balasan yang perlu dihantar kepada BCS yang sama, ia mesti mengatur balasan-balasan tersebut dan tidak menghantar mana-mana balasan dalam aturan sehingga jawapan diterima untuk balasan yang telah dihantar sebelumnya.

Versi Pelayan Tawaran yang anda akan tulis tidak perlu mengendalikan penamatan dan permulaan semula. Anda boleh menganggap bahawa setelah dimulakan, Pelayan Tawaran tersebut terus dilaksanakan sehingga semua tawaran telah dikendalikan dan jika tawaran baru perlu ditambah, Pelayan Tawaran akan ditamatkan dan dimulakan semula dengan tawaran baru dalam tatasusunan tawaran yang baru.

2 RUMUSAN KEWANGAN TIKET

Sebuah taman tema menggunakan sistem pengurusan tunai berasaskan tiket untuk memudahkan penerimaan tunai di setiap permainan sambil mengelakkan kesulitan dan risiko berkaitan dengan pengendalian wang baki tunai.

Untuk menyertai sesuatu permainan, pelanggan boleh memasukkan tunai atau tiket yang diperolehi dari permainan sebelumnya ke dalam satu pengesah bil yang dipasang pada pintu pagar kawalan masuk ke permainan tersebut. Dalam mana-mana kes juga, pintu pagar akan mengembalikan baki tunai dalam bentuk tiket yang boleh digunakan di permainan lain atau ditukar kepada tunai di mana-mana mesin tunai automatik (ATM) berhampiran pintu keluar taman tema tersebut.

Jika sesekeping tiket telah hilang atau rosak, taman tema tersebut boleh membatalkan tiket tersebut dan memulangkan baki tunai kepada pelanggan.

Jika sesekeping tiket tidak diguna dalam masa 6 bulan, tiket tersebut akan luput secara automatik.

Tugas anda ialah menulis satu Rumusan Kewangan Tiket yang merumuskan aktiviti tiket dalam taman tema tersebut.

Program laporan tersebut mesti menerima parameter berikut:

3. tarikh-dari;
4. tarikh-hingga.

Rumusan Kewangan Tiket tersebut mesti mengandungi maklumat berikut:

4. tempoh laporan (tarikh-dari dan tarikh-hingga);
5. jika mana-mana hari urus-niaga dalam tempoh laporan masih lagi terbuka untuk menerima transaksi, perkataan ‘PRELIMINARY’, jika tidak perkataan ‘FINAL’;
6. bilangan transaksi dan jumlah nilai transaksi dalam tempoh laporan bagi setiap maklumat berikut :
    a. tiket yang belum ditebus pada permulaan tempoh laporan (opening unredeemed tickets);
    b. tiket yang dikeluarkan oleh permainan (tickets issued by rides);
    c. tiket yang diterima oleh permainan (tickets accepted by rides);
    d. tiket yang ditunaikan di ATM (tickets cashed at ATMs);
    e. tiket yang dibatalkan (cancelled tickets);
    f. tiket yang telah luput (expired tickets);
    g. tiket yang belum ditebus pada akhir tempoh laporan (closing unredeemed tickets).

Rumusan Kewangan Tiket tersebut harus diatur seperti contoh dalam Gambarajah 7.

```
PRELIMINARY/FINAL TICKET FINANCIAL SUMMARY
FOR DD-MM-YY TO DD-MM-YY
                                      Number           Value

Opening unredeemed tickets             1,520       18,240.40
Tickets issued by rides               22,920      340,227.27
Tickets accepted by rides             18,517      292,420.85
Tickets cashed at ATMs                 3,927       40,650.96
Cancelled tickets                          5           72.24
Expired tickets                          120          927.72
                                      -------     -----------
Closing unredeemed tickets             1,871       24,395.90
```

Gambarajah 7 – Contoh Laporan Kewangan Tiket

Program laporan tersebut mesti mengekstrak maklumat yang dikehendaki untuk menghasilkan Ringkasan Kewangan Tiket daripada pangkalan data taman tema. Pangkalan data tersebut mengandungi jadual-jadual berikut yang berkaitan dengan laporan tersebut:

```
create table dayStatus (
     tradingDate    date not null,
     dayOpen        smallint not null
);
create table tickets (
     ticketNo       integer not null,
     ticketStatus   smallint not null,
     ticketValue    double precision not null
);
create unique index ticketNoInd on tickets (ticketNo);
create table transactions (
     txNo           integer not null,
     ticketNo       integer not null,
     locationId     integer not null,
     txTime         timestamp not null,
     txDate         date not null,
     txType         smallint not null
);
create index txDateInd on transactions (txDate);
```

`dayStatus`

merupakan satu jadual yang mengandungi satu baris untuk hari urusniaga semasa.

`dayStatus.tradingDay`

merupakan tarikh untuk hari urusniaga semasa. Kadang kala taman tema tersebut buka sehingga awal pagi hari yang berikutnya. Dalam situasi sebegini, transaksi-transaksi yang berlaku dalam awal pagi hari yang berikutnya diambil kira ke dalam hari urusniaga sebelumnya.

`dayStatus.dayOpen`
: ialah sifar jika hari urusniaga semasa telah tutup dan satu jika hari urusniaga semasa masih terbuka.

`tickets`
: merupakan satu jadual yang meyimpan status semasa bagi setiap tiket.

`tickets.ticketNo`
: merupakan nombor tiket yang mengenalpasti sesuatu tiket secara unik.

`tickets.ticketStatus`
: merupakan kod status tiket. Kod-kod status tiket disenaraikan dalam Jadual 1.

`tickets.ticketValue`
: merupakan nilai tiket dalam sen. Contohnya, $12.34 disimpan sebagai 1234.0

`ticketNoInd`
: merupakan indeks pangkalan data pada `tickets.ticketNo` yang membolehkan pengguna pangkalan data mencapai satu baris tiket dengan cepat melalui nombor tiket yang diketahui.

`transactions`
: merupakan satu jadual yang menyimpan maklumat mengenai transaksi-transaksi yang berkaitan dengan tiket.

`transactions.txNo`
: merupakan satu nombor transaksi yang mengenalpasti sesuatu transaksi secara unik. Nombor transaksi sentiasa diperuntukkan dalam urutan di mana transaksi berlaku.

`transactions.ticketNo`
: merupakan nombor tiket bagi tiket di mana transaksi dikaitkan. Ia berada dalam domain yang sama dengan `tickets.ticketNo.`

`transactions.locationId`
: merupakan id lokasi yang mengenalpasti sesuatu permainan, ATM atau juruwang yang memproses transaki tersebut secara unik.

`transactions.txTime`
: merupakan masa di mana transaksi tersebut berlaku.

`transactions.txDate`
: merupakan tarikh hari urusniaga di mana transaksi tersebut berlaku.

`transactions.txType`
: merupakan satu kod jenis transaksi. Kod-kod jenis transaksi disenaraikan dalam Jadual 2.

`txDateInd`
: merupakan indeks pangkalan data pada tarikh transaksi yang membolehkan pengguna pangkalan data mencapai sesuatu baris daripada jadual `transactions` dengan cepat melalui tarikh transaksi.

Jadual 1 – Kod-kod Status Tiket

Kod	Makna
1	Tiket telah dikeluarkan oleh satu permainan tetapi belum lagi ditebus.
2	Tiket telah diterima oleh satu permainan.
3	Tiket telah ditunaikan di ATM.
4	Tiket telah dibatalkan.
5	Tiket telah luput.

Jadual 2 – Kod-kod Jenis Transaksi

Kod	Makna
1	Dikeluarkan oleh satu permainan
2	Penerimaan oleh satu permainan
3	Penunaian di satu ATM
4	Pembatalan
5	Proses meluputkan tiket

Program laporan tersebut harus menggunakan indeks pangkalan data secara sesuai supaya mengurangkan masa perlaksanaan program laporan tersebut.

Pengaturcara-pengaturcara yang menulis dalam C atau C++ boleh menggunakan fungsi-fungsi penukaran tarikh seperti diterangkan dalam Gambarajah 8. Pengaturcara-pengaturcara Java dan C# harus menggunakan fungsi-fungsi penukaran tarikh yang dibekal oleh perpustakaan piawai bahasa Java dan C#.

```
#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.
```

Gambarajah 8 – Fungsi-fungsi Tarikh dan Masa

3 JAM SELARAS RANGKAIAN

Sebuah pelayan masa yang boleh dicapai secara awam menerima permintaan HTTP seperti dalam Gambarajah 9 dan mengembalikan balasan HTTP seperti dalam Gambarajah 10.

Contoh dalam Gambarajah 9 dan Gambarajah 10 termasuk pemula HTTP dan dalam contoh tersebut, kod '\r' dan '\n' mewakili aksara- aksara "carriage return" dan baris baru masing-masing.

Sesuatu pelayar laman web akan memaparkan balasan HTTP masa seperti dalam Gambarajah 11.

Pengguna pelayan masa tersebut mendapati bahawa adalah sukar untuk mengenalpasti kesilapan dalam masa yang dikembalikan sebab pengguna tidak pasti sama ada tangguhan berasal dari permintaan yang dihantar ke pelayan atau dari balasan yang dikembalikan kepada pengguna.

Pengguna ingin melihat satu jam dinamik seperti yang ditunjukkan dalam Gambarajah 12.

```
GET /TIME HTTP/1.1\r\n
User-Agent: TimeClient\r\n
Host: time.rwgenting.com:2212\r\n
Connection: close\r\n
\r\n
```

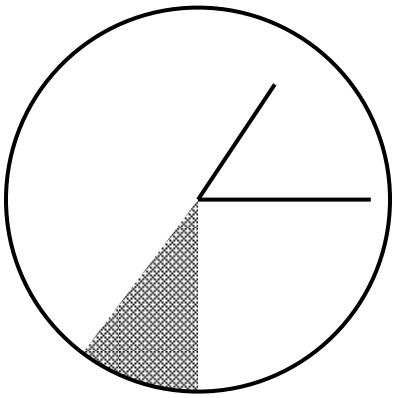
Gambarajah 9 – Permintaan Masa HTTP

```
HTTP/1.1 200 OK\r\n
Server: TimeServer\r\n
Date: Wed, 26 Sep 2012 06:00:30 GMT\r\n
Accept-ranges: none\r\n
Content-type: text/plain\r\n
Connection: close\r\n
Content-length: 65\r\n
\r\n
<html>\r\n
<body>\r\n
Wed, 26 Sep 2012 14:00:30 MYT\r\n
</body>\r\n
</html>\r\n
```

Gambarajah 10 – Balasan Masa HTTP

Wed, 26 Sep 2012 14:00:30 MYT

Gambarajah 11 – Paparan dalam Pelayar Laman Web



Gambarajah 12 - Jam

Jam dalam Gambarajah 12 menunjukkan bahawa masa sekarang ialah di antara 1:15 saat 30 sehingga 1:15 saat 37.

Tugas anda ialah menghasilkan satu program pelanggan yang menyoalminta pelayan masa dan memaparkan muka jam.

Program pelanggan tersebut mesti mencatat masa dari jam tempatan komputer pelanggan dengan segera sebelum ia menghantar permintaan masa HTTP kepada pelayan masa (masa penghantaran) dan kemudiannya mencatat masa dengan segera sekali lagi setelah menerima balasan masa HTTP (masa penerimaan). Perbezaan di antara masa penghantaran dan masa penerimaan (lat masa) adalah nilai ralat yang berkemungkinan bagi nilai masa yang dikembalikan oleh pelayan masa (masa pelayan), iaitu masa sebenar bernilai di antara masa pelayan dan masa yang dikembalikan oleh pelayan masa tambah lat masa. Program pelanggan tersebut mesti memaparkan komponen saat bagi tempoh masa ini sebagai sektor kelabu pada muka jam.

Setelah mendapat masa pelayan, mengira lat masa dan memaparkan muka jam, program pelanggan tersebut mesti memajukan masa yang dipaparkan pada muka jam setiap saat dengan menggerakkan sektor kelabu sekitar muka jam, dan petunjuk minit dan jam digerakkan dalam cara yang sama seperti jam biasa.

Jika lat masa ialah 60 saat atau lebih, program pelanggan tersebut mesti memaparkan seluruh muka jam dalam warna kelabu.

Program pelanggan tersebut mesti meletakkan petunjuk minit dan jam pada minit yang terdekat dengan nilai tengah di antara masa pelayan dan masa pelayan tambah lat masa, apabila dimajukan oleh masa yang berlalu.

Setelah mendapat nilai masa permulaan, program pelanggan tersebut mesti menambah tempoh masa yang dipaparkan secara simetri untuk membenarkan kemungkinan ralat ±10 saat setiap hari dalam ketepatan jam pada komputer pelaggan.

Setelah menerima masa pelayan daripada pelayan masa, memaparkan muka jam dan memulakan kemas paparan bagi setiap saat, program pelanggan tersebut mesti berehat selama 60 saat dan kemudian menghantar satu lagi permintaan masa HTTP kepada pelayan masa. Apabila balasan masa HTTP diterima, program pelanggan tersebut mesti mengira penindihan di antara lingkungan masa yang dihasilkan daripada permintaan baru dengan lingkungan masa paparan yang dilaras berhubung dengan ralat ketepatan pada komputer pelanggan dan melaraskan lingkungan masa paparan kepada penindihan tersebut.

Jika lingkungan masa yang dihasilkan daripada permintaan baru dengan lingkungan masa paparan tidak bertindih, program pelanggan tersebut mesti menganggap bahawa lingkungan masa paparan telah cacat dan menetapkan lingkungan masa paparan kepada lingkungan masa yang dihasilkan daripada permintaan baru.

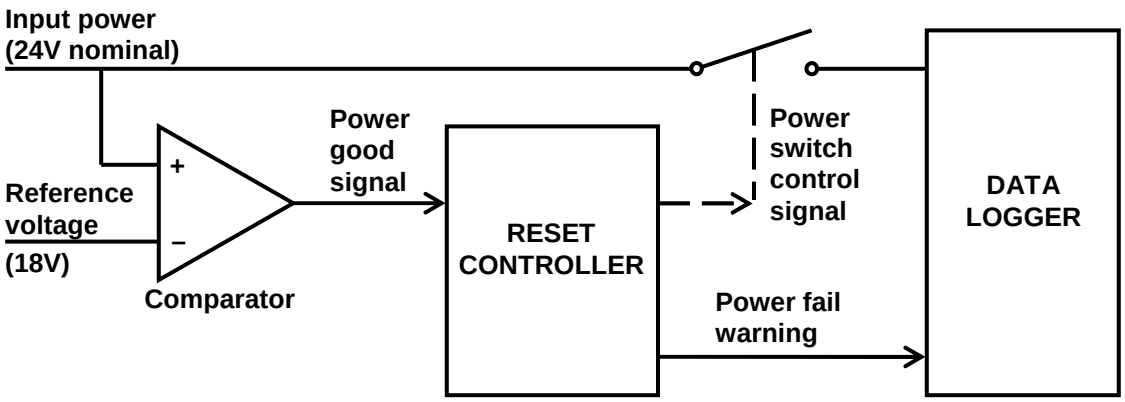
Program pelanggan tersebut mesti mengulangi proses di atas 60 saat selepas menerima setiap balasan masa HTTP.

Jika program pelanggan gagal untuk mendapatkan balasan masa HTTP daripada pelayan masa (sebagai contoh, disebabkan oleh pelayan masa gagal dihubungi atau di luar rangkaian), program pelanggan masih perlu menambah tempoh masa yang dipaparkan secara simetri untuk membenarkan kemungkinan ralat pada ketepatan jam komputer pelanggan dan menunggu selama 60 saat sebelum mencuba menghubungi pelayan masa sekali lagi. Program pelanggan mesti mengulangi proses ini, iaitu menunggu selama 60 saat di antara percubaan, sehingga program pelanggan berjaya menghubungi pelayan masa. Apabila program pelanggan berjaya menghubungi pelayan masa, ia mesti memproses balasan masa seperti yang diterangkan oleh perenggan-perenggan di atas.

4 PENGAWAL RESET

Jurutera-jurutera yang menyelenggarakan sepasukan feri penumpang saiz sederhana mendapati bahawa dalam beberapa peristiwa mereka tidak dapat memuat turun data dari pencatat data yang dipasang dalam feri. Dengan bertambahnya peristiwa-peristiwa hentakan dan pelanggaran, mereka bimbang akan kehilangan data-data yang penting. Mereka mengesan bahawa masalah tersebut berpunca daripada pemudaran kuasa ketika memulakan enjin. Motor elektrik yang memulakan enjin mengambil jumlah besar kuasa dari bateri-bateri feri lalu menyebabkan kuasa input ke pencatat data berlegar di sekitar tahap operasi minima. Walaupun pencatat data tersebut direka untuk menahan putusan kuasa, apabila kuasa berlegar di sekitar tahap minima, masalah berlaku pada pencatat data tersebut dan kadang-kala data yang disimpannya akan hilang.

Memandangkan penggantian pencatat data adalah agak mahal, jurutera-jurutera tersebut mencadangkan supaya satu pengawal reset dipasang seperti dalam Gambarajah 13.



Gambarajah 13 – Skema Pengawal Reset

Cadangan itu ialah pengawal reset akan membandingkan kuasa input 24 Volt nominal dengan rujukan 18 Volt. Apabila voltan input tersebut jatuh di bawah voltan rujukan, pengawal reset tersebut akan mencetuskan proses yang mematikan pencatat data untuk mengelakkan kehilangan data. Sebelum mematikan pencatat data, pengawal reset tersebut mesti mengaktifkan satu isyarat amaran kegagalan kuasa yang mengakibatkan pencatat data tersebut menyimpankan maklumat penting sebelum dimatikan. Tugas anda ialah untuk mengaturcara pengawal reset tersebut.

Apabila pengawal reset tersebut dimulakan, ia mesti mengaktifkan amaran kegagalan kuasa dan mematikan pencatat data dengan menyah-aktif isyarat pengawal suis kuasa. Ia kemudiannya mesti memantau isyarat kuasa baik dan menunggunya mencapai nilai tinggi, di mana kuasa input mencapai18 Volt atau lebih. 12 Volt diperlukan untuk mengendalikan pencatat data. Apabila pengawal reset tersebut mengesan bahawa isyarat kuasa baik adalah dalam keadaan tinggi, ia mesti menunggu selama 3 saat dan kemudian menghidupkan pencatat data dan menyah-aktifkan amaran kegagalan kuasa. Jika isyarat kuasa baik bertukar ke keadaan rendah ketika pengawal tersebut sedang menunggu 3 saat, pengawal reset tersebut mesti mencetuskan proses permulaan semula dan menunggu sekali lagi supaya isyarat kuasa baik menukar ke keadaan tinggi sebelum menunggu 3 saat sekali lagi dan seterusnya.

Setelah pencatat data tersebut telah dimulakan, pengawal reset tersebut mesti memantau isyarat kuasa baik sekurang-kurangnya sekali setiap mili-saat. Dalam peralihan tinggi ke rendah bagi isyarat kuasa baik, pengawal reset tersebut mesti mengaktifkan amaran kegagalan kuasa kepada pencatat data dan mula menunggu. Jika dalam penungguan, isyarat kuasa baik kembali ke keadaan tinggi, pengawal reset tersebut mesti menyah-aktifkan amaran kegagalan kuasa dan kembali memantau isyarat kuasa baik. Jika isyarat kuasa baik kekal dalam keadaan rendah selama 20 mili saat atau lebih, pengawal reset tersebut mesti memadamkan pencatat data dan mencetuskan proses permulaan semula seperti dijelaskan dalam perenggan sebelum ini.

Secara umumnya, pengawal reset tersebut harus mengesan perubahan keadaan isyarat input dengan mengambil sampel nilai dalam selang masa 1 mili saat.

Program pengawal data tersebut mesti menggunakan fungsi-fungsi yang ditunjukkan dalam Gambarajah 14 untuk mencapai dan menetapkan isyarat-isyarat.

C++

```
class ResetIO_c {
public:
    static bool IsPowerGood();
                    // Test the power good
                    // signal
    static void SetPowerSwitch (
            bool powerOn);
                    // Turn the power on or
                    // off
    static void SetPowerWarn (
            bool warnOn);
                    // Activate or
                    // deactivate the power
                    // fail warning
};
```

Java

```
public class ResetIO {
    public static boolean isPowerGood()
                    // Test the power good
                    // signal
    public static void setPowerSwitch (
            boolean powerOn)
                    // Turn the power on or
                    // off
    public static void setPowerWarn (
            boolean warnOn)
                    // Activate or
                    // deactivate the power
                    // fail warning
};
```

C#

```
class ResetIO {
    public static bool IsPowerGood();
                    // Test the power good
                    // signal
    public static void SetPowerSwitch (
            bool powerOn);
                    // Turn the power on or
                    // off
    public static void SetPowerWarn (
            bool warnOn);
                    // Activate or
                    // deactivate the power
                    // fail warning
};
```

`IsPowerGood/isPowerGood`
: mencuba keadaan isyarat kuasa baik. Ia mengembalikan 1 (atau benar) jika isyarat kuasa baik adalah tinggi (iaitu, kuasa input adalah 18V atau lebih) atau 0 (atau palsu) jika isyarat kuasa baik adalah rendah (iaitu kuasa input di bawah 18V).

`SetPowerSwitch/setPowerSwitch`
: menghidup dan mematikan suis kuasa.

`powerOn`
: adalah keadaan di mana suis kuasa harus ditetapkan. Jika ia bernilai 1 (atau benar), suis kuasa dihidupkan. Jika ia bernilai 0 (atau palsu), suis kuasa dimatikan.

`SetPowerWarn/setPowerWarn`
: menetapkan keadaan isyarat amaran kegagalan kuasa ke pencatat data.

`warnOn`
: adalah keadaan bagi amaran kegagalan kuasa. Jika ia bernilai 1 (atau benar), amaran kegagalan kuasa diaktifkan. Jika ia bernilai 0 (atau palsu), amaran kegagalan kuasa dinyah-aktifkan.

Gambarajah 14 – Fungsi-fungsi I/O untuk Pengawal Reset