E-GENTING PROGRAMMING COMPETITION

General instructions:

1. Answer one or more of the questions provided.
2. The competition is an open book test.
3. The duration for the competition is 8 hours.
4. Discussion between participants is not allowed.
5. Credit will be given for work-in-progress, especially notes such as data flow diagrams and state transition diagrams or tables.
6. Additional credit will be given to those who can answer more than one question.
7. 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.
8. The marks allocated for the questions are as follows:

No	Name	Marks
1.	Old Tape Data Recovery	900
2.	Lollypop Cruise Lines	450
3.	Data Projection and Sorting	225
4.	Traffic Light System	100

9. You should make use of all the standard library functions of the chosen language.
10. You are NOT expected to answer all questions.

1 OLD TAPE DATA RECOVERY

Twenty years ago an eminent scientist with considerable foresight undertook extensive climatic data collection across peninsula Malaysia. There is now considerable debate as to whether two decades of progressively increasing vehicle emissions and industrial pollution have altered the climate of the peninsula. A team of scientists have been appointed to study the matter. The team would like to make use of the twenty-year old data but it is unable to read the old backup tape format. Your job is to write a computer programme to extract individual data files from the old tapes.

The old tapes are an image copy of the original disk file system. The disk file system consisted of 1024 byte blocks arranged in the following way:

Block number	Contents
0	Bootstrap loader for the operating system
1 to 31	Directory blocks
32 onwards	Data blocks

Each directory block consisted of an array of directory entries, each with the following structure:

Byte number	Contents
0 to 15	File name
16 to 19	File length in bytes.
20 to 23	Block number of the first data block in the file
24 to 27	Block number of a pointer data block that contains the block numbers of the 2^{nd} to the 257^{th} data blocks in the file
28 to 31	Block number of a pointer data block that contains the block numbers of a second level of pointer data blocks that contain the block numbers of the 258^{th} to the $65{,}793^{rd}$ data blocks in the file.

File names were between 1 and 16 characters long. If the file name was less than 16 characters, it was null terminated. If it was 16 characters long, the null terminator was omitted. Unused directory entries had a null character in the first character of the file name.

Unused data block numbers were zero. For example, if the length of the file was 1024 bytes, only the first block number in the directory entry would be used. If the length of the file was 2048 bytes, the first and second block numbers in the directory entry would be active, but only the first block number in the first pointer block would be used.

The byte order of the block numbers was big-endian (i.e. the most significant byte came first). Your computer program must run on an Intel microprocessor, a small-endian computer.

Your computer program must accept a file name as an input parameter, extract the file from the tape and write it to the local disk drive.

Because the backup media is tape, it can only be sequentially read and rewound. Your computer program should endeavour to extract the requested file from the tape in the minimum number of passes. Your computer program must not use more than 500k of data RAM. There is not enough free disk space to unload the entire tape onto disk and then randomly access the copy of the tape on disk.

C and C++ programmers may assume that the second argument to the `fseek` system call is a byte offset and is not restricted to a value returned by the `ftell` system call.

2 LOLLYPOP CRUISE LINES

On the good ship, Lollypop,
It's a sweet trip,
Into bed you'll hop,
And dream away,
On the good ship, Lollypop!
Sidney Clare

Lollypop Cruise Lines operates a cruise ship called 'Lollypop'. Each day, the crew of Lollypop needs to download approximately 1Mb of information from a shore-based file server via satellite.

The communications connections are as shown in figure 1 below.

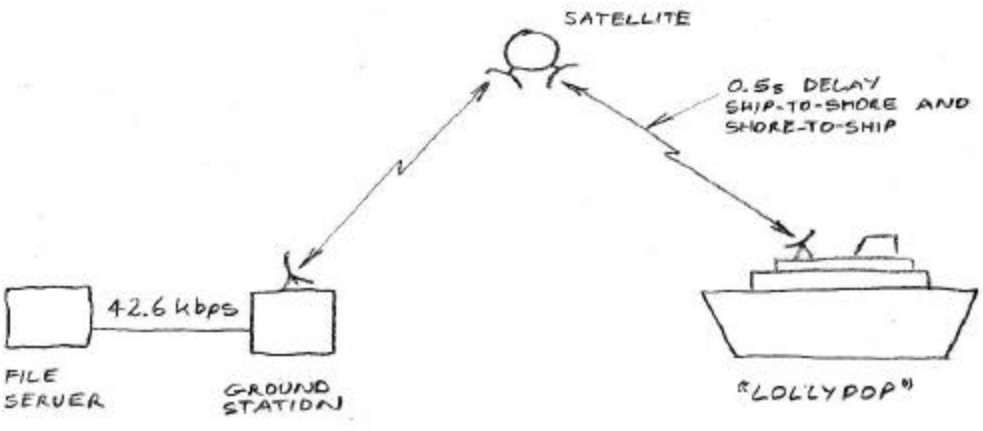

FIGURE 1

The file server is connected to the satellite ground station via a 42.6 kbps modem. The

ground station relays the data to the satellite, which passes the data on to Lollypop. The satellite connection has, for all intents and purposes, infinite bandwidth. It can relay gigabits per second. However, it introduces a delay of 0.5 seconds as a consequence of the distance to the satellite and amount of time it takes for the microwave signal to reach the satellite and return to earth.

The crew of Lollypop have complained about the amount of time it is taking to download the 1Mb daily data file. According to their reports, the download is taking between 20 minutes and half an hour. They need to reduce this time to around 5 minutes in order to complete their daily routine on time.

The ISP that operates the satellite ground station has suggested upgrading the connection between the file server and the ground station to a 256 kbps leased line. This upgrade will cost around RM120,000.

A technician on Lollypop, who understands the nature of the protocol between the file server and Lollypop, is doubtful that increasing the speed of the connection between the file server and the ground station will have any effect.

Apparently the protocol between Lollypop and the file server is very simple. Lollypop sends a 64-byte message to the ground station requesting a 1kb block from the file server. The file server then responds with the 1kb block in a message 1088 bytes long. When Lollypop receives the 1kb block it repeats the sequence by sending another 64-byte message to request the next block.

The Lollypop only waits 5s for a response from the file server. If either the 64-byte message or the 1kb block are lost or corrupted in transit, the 5s period will expire, and the Lollypop will retransmit the original 64-byte message to restart the sequence. The technician on board the Lollypop, in an effort to find out more about the cause of the long download time, kept records of data losses. He found that on the average, the likelihood of either a ship-to-shore or shore-to-ship satellite message being lost is 1/200. No message losses were recorded on the file server to ground station link.

The technician believes that the best way to reduce the download time will be to change the communications protocol to transfer 8 blocks in parallel. He suggests that the existing protocol be split into 8 parallel processes. The first process should take responsibility for transferring the first block in the file. The second process, the second block and so forth. When a process finishes transferring a block, it should move on to transfer the next block waiting to be transferred until all the block transfers are completed.

Your task is to resolve the dispute between the ISP and the technician by writing a computer simulation program that simulates the transfer of, say, 1000 1Mb download files and estimates the transfer time under the following conditions:

1. the existing single-threaded protocol with the 42.6 kbps modem speed;

2. the existing single-threaded protocol with the 256 kbps leased line speed proposed by the ISP;

3. the 8-thread protocol proposed by the technician with the existing 42.6 kbps modem speed.

For the purposes of your model, you can assume that any delays introduced by the file server, ground station and satellite are negligible. The ground station and the file server both queue messages to be transmitted across the line between them in the order the messages are put in the respective transmit queues. The connection between the file server and the ground station is serial, full duplex, asynchronous, 8 data bits, one stop bit and no parity.

3 DATA PROJECTION AND SORTING

A small business has a large text-format customer data file. The business requires a computer program that can select specific columns from the file and output the selected columns, sorted, in constant-width fields.

The data for each customer is stored on a single line in the file. Each line contains the following fields:

Field name	Description
custId	customer identifier
name	customer's name
typeCode	a customer type code that groups customers with similar needs
region	a code that identifies the region that the customer comes from, for example, 'KL' or 'Seremban'
address	the customer's address

Each field is stored as a string enclosed in inverted commas. For example, "Barney Rubble". If the string, itself, contains an inverted comma, the inverted comma is escaped with a backslash character. For example, the sequence \" indicates an inverted comma embedded in the body of the string. A backslash character may be included in a string by escaping it with another backslash character. For example the sequence \\ indicates a single backslash character embedded in the body of the string.

One or more space characters separate adjacent fields.

The small business requires a computer program that can accept a string of field names and then read the customer file, select the specified field names from the customer file, sort the selected data and then output the fields in constant-width columns.

All output columns should be sorted in ascending sequence.

The string of field names must be entered in a line similar to the example below:

region, name, typeCode

A comma and one or more spaces will separate the field names.

The above field name string should generate an output similar to the following:

AUS	Humphrey B. Bear	Kids' Show Host
US	Donald Duck	Comic Character
US	Goofy	Comic Character
US	Mickey Mouse	Comic Character

In this case the primary sorting order is region code, followed by the customer's name and then the customer type code.

You may assume that sufficient RAM is available to load the entire customer database into a RAM array.

4 TRAFFIC LIGHT SYSTEM

A city council has recently purchased a new traffic light controller for a four-way intersection. Vehicles enter the intersection from the north, east, south and west. The traffic light controller can be programmed in C, C++ or Java. Your task is to program the traffic light controller to execute the cycle requested by the city's traffic manager.

The requested cycle is as follows:

Phase period (seconds)	North facing lights	East facing lights	South facing lights	West facing lights
30	Green	Red	Red	Red
5	Amber	Red	Red	Red
60	Red	Green	Red	Red
5	Red	Amber	Red	Red
40	Red	Red	Green	Red
5	Red	Red	Amber	Red
50	Red	Red	Red	Green
5	Red	Red	Red	Amber

The full traffic light cycle should only be executed when vehicles are waiting on each branch entering the intersection. If there are no vehicles waiting at a branch when the time arrives for the lights to turn green, the green and amber phases for the branch should be skipped and the sequence should move on to the next branch.

If there are no vehicles waiting on any branch, then the cycle should continue as if there were vehicles waiting at every branch.

The C and C++ programming interface consists of the following functions:

```
typedef enum {RED, AMBER, GREEN} Colour;
                    // Traffic light colours
void Display (Colour northColour, Colour eastColour,
     Colour southColour, Colour westColour);
                    // Set the traffic light colours
void InitApp ();
                    // Initiate application
void Tick ();
                    // Process clock tick
void VehicleWaiting (int northWait, int eastWait,
     int southWait, int westWait);
                    // Process change in vehicle
                    // waiting sensor inputs
```

The `Display` function is called by the traffic light application to set the colours displayed by the physical traffic lights. The parameters `northColour, eastColour, southColour` and `westColour` are the traffic light colours to be displayed towards the north, east, south and west branches of the intersection respectively.

The `InitApp` function is an entry point in the application. It is called by the traffic light operating system to initialise any application data. The `InitApp` function must initialise any application data and then return immediately. The application must not hold control in the `InitApp` function. `InitApp` should call `Display` to establish the initial traffic light settings.

The `Tick` function is an another entry point in the application. It is called by the traffic light operating system once every second. The application (which you are to write) must use the calls it receives via the `Tick` function to measure the elapse of time.

The `VehicleWaiting` function is another entry point in the application. It is called by the traffic light operating system whenever the operating system senses a change in the state of the vehicle waiting sensors. The parameters `northWait, eastWait, southWait` and `westWait` are zero if no vehicles are waiting at the corresponding branches of the intersection and non-zero if vehicles are waiting.

The above declarations are contained in the header file '`lightsys.h`'.

The Java programming interface consists of the following Java interfaces:

```
interface TrafficSystem {
     static final int   RED = 0;
     static final int   AMBER = 1;
     static final int   GREEN = 2;
                    // Traffic light colour codes
     void display (int northColour, int eastColour,
          int southColour, int westColour);
                    // Set the traffic light colours
```

```
}

class TrafficApp {
     abstract void initApp (TrafficSystem ts);
                    // Initiate application
     abstract void tick ();
                    // Process clock tick
     abstract void vehicleWaiting (int northWait,
          int eastWait, int southWait, int westWait);
                    // Process change in vehicle
                    // waiting sensor inputs
}
```

The `TrafficSystem` interface represents the traffic light operating system.

The `display` method in the `TrafficSystem` interface is called by the traffic light application to set the colours displayed by the physical traffic lights. The parameters `northColour, eastColour, southColour` and `westColour` are the traffic light colours to be displayed towards the north, east, south and west branches of the intersection respectively. Their values may be either `RED, AMBER` or `GREEN`.

The `TrafficApp` class is the super-class of the traffic light application. The traffic light application must override the `initApp, tick` and `vehicleWaiting` methods.

The `initApp` method is called by the traffic light operating system to initialise any application data. The `initApp` method must initialise any application data and then return immediately. The application must not hold control in the `initApp` method. The `initApp` method must save the reference to the `TrafficSystem` instance so that the application can call the `display` method later on. The `initApp` method should call `display` to establish the initial traffic light settings.

The `tick` method is called by the traffic light operating system once every second. The application (which you are to write) must use the calls it receives via the `tick` method to measure the elapse of time.

The `vehicleWaiting` method is called by the traffic light operating system whenever it senses a change in the state of the vehicle waiting sensors. The parameters `northWait, eastWait, southWait` and `westWait` are zero if no vehicles are waiting at the corresponding branches of the intersection and non-zero if vehicles are waiting.

PERTANDINGAN PENGATUCARAAN E-GENTING

Arahan-arahan am:

1. Jawab satu atau lebih daripada soalan-soalan yang diberikan.
2. Pertandingan ini adalah satu ujian dimana calon-calon boleh merujuk kepada buku-buku atau bahan-bahan rujukan.
3. Masa yang diperuntukkan kepada pertandingan ini ialah lapan jam.
4. Perbincangan di antara peserta-peserta adalah tidak dibenarkan.
5. Kredit akan diberikan bagi hasil kerja dalam proses, khasnya, nota-nota seperti rajah pengaliran data dan rajah atau jadual *state transitions*.
6. Kredit tambahan akan diberikan kepada mereka yang boleh menjawab lebih daripada satu soalan.
7. Program-program anda akan dinilai berdasarkan kepada betapa mudahnya mereka boleh dibaca dan difahami.
   - Takukan mestilah bersih and selari.
   - Nama-nama pembolehubah harus memberi gambaran mengenai isi kandungan pembolehubah-pembolehubah berkenaan.
   - Pasangan (*coupling*) di antara modul-modul mestilah nyata.
   - Setiap modul harus melakukan satu perkara dengan baik
8. Markah yang diperuntukkan kepada setiap soalan adalah seperti berikut:

No	Nama	Markah
1.	Pengembalian Data Dari Pita Lama	900
2.	Syarikat Pelayaran Lollypop	450
3.	Tayangan dan Penyusunan Data	225
4.	Sistem Lampu Isyarat LaluLintas	100

9. Anda harus menggunakan fungsi-fungsi perpustakaan piawai bagi bahasa pengaturcaraan yang dipilih.
10. Anda TIDAK dijangka akan menjawab semua soalan.

1 PENGEMBALIAN DATA DARI PITA LAMA

Pada dua puluh tahun yang lalu, seorang saintis yang unggul dan mempunyai kebolehan yang agak tinggi untuk mengetahui sesuatu yang belum berlaku telah mengutip data mengenai keadaan cuaca di seluruh Semenanjung Malaysia secara besar-besaran. Kini, terdapat agak banyak perdebatan mengenai sama ada pengeluaran asap kenderaan dan pencemaran dari sektor industri yang semakin bertambah sepanjang dua dekad yang lampau telah mengubah suasana iklim Semenanjung Malaysia. Sekumpulan saintis telah dilantik untuk mengkaji perkara ini. Kumpulan tersebut ingin menggunakan data yang berusia dua puluh tahun itu tetapi mereka tidak berupaya memperolehi kembali data itu dari pita-pita *backup* yang berformat lama. Tugas anda adalah menulis satu program komputer untuk memetik fail-fail data dari pita-pita lama itu.

Pita-pita lama itu adalah satu *image copy* bagi sistem fail cakera yang asal. Sistem fail itu terdiri daripada blok-blok 1024 *bytes* yang disusun secara berikut:

Nombor blok	Isi kandungan
0	*Bootstrap loader* bagi sistem pengendalian
1 hingga 31	Blok-blok *directory*
32 ke atas	Blok-blok data

Setiap blok *directory* terdiri daripada satu *array* yang mengandungi butir-butir bagi sekumpulan *directory*. Butir-butir setiap *directory* diatur dalam struktur yang berikut:

Nombor byte	Isi kandungan
0 hingga 15	Nama fail
16 hingga 19	Saiz fail dalam unit *byte*
20 hingga 23	Nombor blok bagi blok data yang pertama dalam fail itu
24 hingga 27	Nombor blok bagi blok data penunding (*pointer*) yang mengandungi nombor-nombor blok bagi blok data kedua hingga blok data ke-257 dalam fail itu
28 hingga 31	Nombor blok bagi satu blok data penunding yang mengandungi nombor-nombor blok untuk blok-blok data penunding tahap kedua. Blok-blok data penunding tahap kedua mengandungi nombor-nombor blok bagi blok-blok data yang ke-258 sampai ke-65,793 dalam fail itu.

Nama-nama fail adalah panjangnya di antara 1 dan 16 huruf. Sekiranya nama fail itu adalah kurang daripada 16 huruf, ia diakhiri dengan huruf *null*. Jika nama fail itu adalah sepanjang 16 huruf, huruf *null* ditinggalkan. Bahagian-bahagian blok *directory* yang dimaksudkan untuk kandungan butir-butir *directory* tetapi tidak terpakai mengandungi huruf *null* di *byte* yang pertama bagi nama fail.

Nombor-nombor blok-blok data yang tidak digunakan adalah sifar. Sebagai contoh, jika saiz fail itu adalah 1024 bytes, hanya nombor blok pertama bagi bahagian directory yang berkenaan akan digunakan. Sekiranya saiz fail itu adalah 2048 bytes, nombor blok pertama dan kedua akan menjadi aktif tetapi hanya nombor blok pertama di blok penunding pertama akan digunakan.

Susunan byte bagi nombor-nombor blok adalah *big-endian* (iaitu *byte* yang paling bernilai datang dahulu). Program konputer anda mestilah dilaksanakan dengan *microprcessor* Intel, sebuah komputer *little-endian*.

Program komputer anda mestilah menerima satu nama fail sebagai satu parameter input, membaca kandungan fail tersebut daripada pita saintis unggul itu dan menulisnya ke atas cakera komputer.

Disebabkan media *backup* itu adalah pita, kandungannya hanya boleh dibaca satu demi satu secara berikutan dan pita itu boleh diputar balik ke mana-mana posisi yang dikehendaki dari posisi semasa. Program komputer anda harus berikhtiar untuk memperolehi kandungan fail yang dikehendaki daripada pita itu dengan bilangan *passes* yang minimum. Program anda tidak boleh menggunakan lebih daripada 500k data RAM. Tidak terdapat ruang cakera bebas untuk memunggah (*unload*) keseluruhan pita itu ke cakera tetap komputer dan kemudiannya akses salinan pita di cakera itu.

Programmer-programmer C dan C++ boleh membuat andaian bahawa argumen kedua kepada panggilan sistem `fseek` adalah satu *byte offset* dan ia tidak semestinya dihadkan kepada nilai yang dipulangkan oleh panggilan sistem `ftell`.

2 SYARIKAT PELAYARAN LOLLYPOP

Di atas kapal yang baik, Lollypop,
Ia satu pelayaran yang manis,
Ke atas katil kamu akan meloncat,
Dan berangan-angan,
Di atas kapal yang baik, Lollypop
Sidney Clare

Syarikat Pelayaran Lollypop mengendalikan sebuah kapal layar bernama ‘Lollypop’. Setiap hari, anak-anak kapal Lollypop perlu memuat-turun (*download*) kira-kira 1Mb maklumat daripada satu pelayan fail (*file server*) yang berada di darat menerusi satelit.

Sambungan komunikasi di antara kapal layar Lollypop dan pelayan fail itu adalah seperti ditunjukkan di gambarajah 1 di bawah.

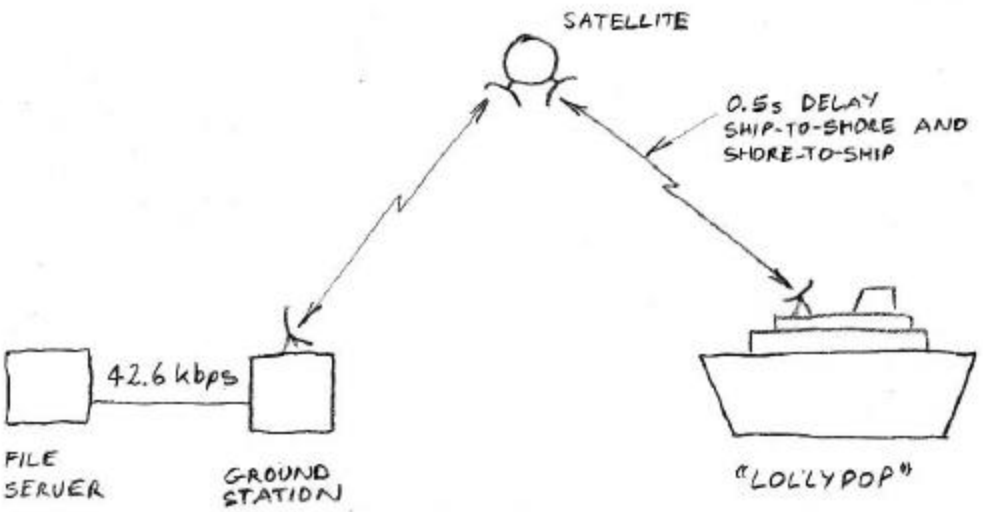

FIGURE 1

Pelayan fail disambungkan ke stesen darat (*ground station*) satelit menerusi satu *modem* dengan kadar laju 42.6kbps. Stesen satelit itu menghantar data ke satelit yang kemudiannya menyampaikan data itu ke Lollypop. Bagi segala maksud dan tujuan, sambungan satelit itu boleh dianggap mempunyai *bandwidth* yang tak terhingga. Ia boleh menghantar data pada kelajuan gigabits per saat. Namun demikian, ia menyebabkan penangguhan selama 0.5 saat, hasil daripada jarak ke satelit itu dan jumlah masa yang diperlukan oleh isyarat gelombang mikro untuk sampai ke satelit dan pulang ke bumi.

Anak-anak kapal Lollypop pernah mengadu tentang masa yang mereka perlu untuk memuat-turun fail data harian 1Mb. Mengikut laporan mereka, muat turun demikian mengambil masa di antara 20 minit dan setengah jam. Mereka perlu mengurangkan masa ini ke lebih kurang 5 minit supaya dapat menghabiskan tugas harian mereka dalam masa yang diperuntukkan.

ISP yang mengendalikan stesen darat satelit itu telah mencadangkan supaya sambungan di antara pelayan fail dan stesen darat dipertingkatkan kepada satu talian sewa (*leased line*) 256kbps. Peningkatan perkhidmatan ini berharga lebih kurang RM120,000.

Seorang juruteknik di Lollypop yang memahami sifat protokol komunikasi di antara pelayan fail dan Lollypop merasa sangsi bahawa peningkatan kelajuan sambungan di antara mereka akan memberi apa-apa kesan.

Nampaknya, protokol yang sedang digunakan itu adalah sangat ringkas. Lollypop menghantar satu pesanan 64-byte kepada stesen darat satelit yang meminta blok 1kb daripada pelayan fail. Pelayan fail kemudian membalas dengan blok 1kb dalam satu *message* sepanjang 1088 bytes. Apabila Lollypop menerima blok 1kb yang dikehendaki, ia mengulangi tertib itu dengan menhantarkan satu lagi pesanan 64-byteuntuk meminta blok data yang berikutnya.

Lollypop hanya menunggu jawapan daripada pelayan fail selama 5 saat. Sekiranya salah satu daripada pesanan 64-byte dan blok data 1kb hilang atau menjadi rosak dalam perjalanannya, tempoh 5 saat itu akan berakhir dan Lollypop akan menghantar sekali lagi pesanan 64-byte yang asal untuk memulakan tertib pemindahan data sekali lagi. Juruteknik di Lollypop, dalam usahanya untuk mengetahui lebih mengenai punca masa muat turun yang panjang, menyimpan rekod-rekod kehilangan data. Dia mendapati bahawa, pada puratanya, kemungkinan kehilangan *message* satelit kapal-ke-darat (*ship-to-shore*) atau darat-ke-kapal (*shore-to-ship*) adalah 1/200 setiap satu. Tiada kes kehilangan *message* bagi sambungan di antara pelayan fail dan stesen darat pernah dicatat.

Juruteknik mempercayai bahawa cara yang paling baik untuk mengurangkan masa muat turun ialah menukarkan protokol komunikasi supaya ia boleh memindahkan 8 blok data secara selari. Dia bercadang bahawa protokol yang wujud sekarang dibahagikan kepada 8 proses selari. Proses pertama harus bertanggungjawab untuk memindahkan blok data pertama dalam fail yang diperlukan. Proses kedua pula bertanggungjawab atas blok kedua dan seterusnya. Apabila sesuatu proses habis memindahkan satu blok, ia harus menghantar blok seterusnya yang sedang menunggu gilirannya sehingga semua pemindahan blok selesai.

Tugas anda adalah menyelesaikan perselisihan di antara ISP dan juruteknik Lollypop itu dengan menulis satu program simulasi komputer yang mengsimulasi pemindahan, katalah, 1000 fail muat turun yang saiz setiapnya 1Mb dan menganggarkan jumlah masa pemindahan di bawah keadaan-keadaan yang berikut:

1. protokol *single-threaded* yang wujud kini dengan kelajuan *modem* 42.6kbps;

2. protokol *single-threaded* yang wujud kini dengan talian sewa berkelajuan 256kbps seperti yang diusulkan oleh ISP;

3. protokol *8-thread* yang dicadangkan oleh juruteknik Lollypop dengan kelajuan *modem* 42.6kbps yang wujud sekarang.

Bagi tujuan model anda, anda boleh membuat andaian bahawa sebarang penangguhan yang disebabkan oleh pelayan fail, stesen darat dan satelit boleh diabaikan. Kedua-dua stesen darat dan pelayan fail mengaturkan (*queue*) *messages* masing-masing yang perlu dipindahkan melalui talian sambungan mereka mengikut masa ketibaan *messages*. Sambungan di antara pelayan fail dan stesen darat adalah *serial*, *full duplex*, *asynchronous, 8 data bits*, 1 *stop bit* dan tiada *parity*.

3 TAYANGAN DAN PENYUSUNAN DATA

Sebuah syarikat perniagaan yang kecil mempunyai satu fail data pelanggan yang besar dimana data pelanggan adalah dalam format teks. Sysrikat itu memerlukan satu program

komputer yang boleh memilih lajur-lajur (*columns*) tertentu daripada fail pelanggan dan mempamerkan mereka di ruangan-ruangan yang sama luas.

Data bagi setiap pelanggan adalah diwakilli oleh satu baris tunggal di dalam fail pelanggan. Setiap baris mengandungi *fields* yang berikut:

Nama *field*	Huraian
custId	pengecam pelanggan
name	nama pelanggan
typeCode	satu kod jenis pelanggan yang mengumpulkan pelanggan-pelanggan dengan keperluan yang semacam
region	satu kod yang mewakilli kawasan dari mana pelanggan itu datang
address	alamat pelanggan

Setiap *field* disimpan sebagai satu rentetan (*string*) yang didahului dan diekori dengan satu tanda pengikat kata. Misalnya, "Barney Rubble". Jika rentetan itu sendiri mengandungi satu tanda pengikat kata, tanda itu adalah didahului dengan satu huruf *backslash*. Sebagai contoh, tertib huruf \" yang terdapat di dalam sesuatu rentetan mewakilli satu tanda pengikat kata. Huruf *backslash* boleh dimasukkan ke dalam sesuatu rentetan dengan mendahuluinya dengan satu huruf yang sama. Contohnya, tertib huruf \\ menunjukkan bahawa satu huruf *backslash* terdapat di dalam rentetan keseluruhannya.

Satu atau lebih huruf *space* mengasingkan *fields* yang bersebelahan.

Syarikat perniagaan kecil itu memerlukan satu program komputer yang boleh menerima satu rentetan nama-nama *fields* yang dikehendaki, membaca fail pelanggannya, memilih *fields* yang dikehendaki, menyusun data yang terpilih dan mempamerkan data di ruangan-ruangan yang sama lebar.

Semua lajur yang dipilih mestilah disusun secara menaik.

Rentetan nama-nama *fields* mestilah ditaip masuk dalam satu baris seperti contoh di bawah:

```
region, name, typeCode
```

Satu tanda comma dan satu atau lebih ruang kosong akan mengasingkan nama-nama *fields* itu.

Rentetan di atas harus menghasilkan laporan seperti berikut:

```
AUS    Humphrey B. Bear    Kids' Show Host
US     Donald Duck         Comic Character
US     Goofy               Comic Character
```

US Mickey Mouse Comic Character

Dalam kes ini, laporan itu disusun mengikut kod kawasan dahulu, diikuti oleh nama pelanggan dan kemudian jenis pelanggan.

Anda boleh mengandaikan bahawa terdapat RAM yang mencukupi bagi memuatkan kesemua data pelanggan ke dalam satu RAM *array*.

4 SISTEM LAMPU ISYARAT LALULINTAS

Baru-baru ini, sebuah majlis bandaraya telah membeli satu alat pengawal lampu isyarat lalulintas yang baru bagi satu simpang empat. Kenderaan-kenderaan masuk ke simpang itu dari arah utara, timur, selatan dan barat. Alat pengawal lampu lalulintas itu boleh diprogram sama ada dalam bahasa pengaturcaraan C, C++ atau Java. Tugas anda ialah memprogramkan alat pengawal itu untuk melaksanakan kitaran lampu lalulintas yang dikehendaki oleh pengurus lalulintas bandar itu.

Kitaran yang dikehendaki adalah seperti berikut:

Tempoh fasa (saat)	Lampu menghadap utara	Lampu menghadap timur	Lampu menghadap selatan	Lampu menghadap barat
30	Hijau	Merah	Merah	Merah
5	Jingga	Merah	Merah	Merah
60	Merah	Hijau	Merah	Merah
5	Merah	Jingga	Merah	Merah
40	Merah	Merah	Hijau	Merah
5	Merah	Merah	Jingga	Merah
50	Merah	Merah	Merah	Hijau
5	Merah	Merah	Merah	Jingga

Kitaran lampu lalulintas yang genap hanya patut dilaksanakan apabila terdapat kenderaan yang sedang menanti di setiap cabang yang sampai gilirannya untuk kenderaan-kenderaannya memasuki simpang empat itu. Jika tiada kenderaan yang sedang menunggu di sesuatu cabang apabila sampai masanya bagi lampu cabang itu bertukar ke warna hijau, fasa-fasa lampu hijau dan jingga cabang itu harus dilangkaui dan turutan acara ini harus beralih ke cabang seterusnya.

Jika tidak terdapat kenderaan yang sedang menunggu di mana-mana cabang, kitaran lampu lalulintas harus dilanjutkan seolah-olah terdapat kenderaan yang sedang menanti di setiap cabang.

Program antaramuka C dan C++ yang berkenaan terdiri daripada fungsi-fungsi yang berikut:

```
typedef enum {RED, AMBER, GREEN} Colour;
                    // Warna-warna lampu lalulintas
```

```
void Display (Colour northColour, Colour eastColour,
     Colour southColour, Colour westColour);
                    // Tetapkan warna-warna lampu
                    // lalulintas
void InitApp ();
                    // Mulakan aplikasi
void Tick ();
                    // Proses detik jam
void VehicleWaiting (int northWait, int eastWait,
     int southWait, int westWait);
                    // Proses perubahan pada inputs
                    // kepada alat pengesan kenderaan
                    // menunggu
```

Fungsi `Display` dipanggil oleh aplikasi lampu lalulintas untuk menentukan warna-warna yang dipamerkan oleh tiang lampu isyarat lalulintas. Parameter-parameter `northColour`, `eastColour`, `southColour` dan `westColour` adalah warna-warna yang dipamerkan kepada cabang-cabang utara, timur, selatan dan barat simpang empat itu masing-masing.

Fungsi `InitApp` ialah satu titik masuk (*entry point*) bagi aplikasi lampu lalulintas itu. Ia dipanggil oleh sistem pengendalian lampu lalulintas untuk memberi nilai permulaan kepada data aplikasi. Fungsi ini mesti menetapkan nilai awal bagi data aplikasi dan kemudian kembali ke pemanggilnya serta-merta. Aplikasi itu tidak harus memegang kawalan dalam fungsi `InitApp`. `InitApp` harus memanggil `Display` untuk menetapkan keadaan permulaan lampu lalulintas.

Fungsi `Tick` adalah titik masuk yang lain dalam aplikasi itu. Ia dipanggil oleh sistem pengendalian lampu isyarat lalulintas sekali setiap saat. Aplikasi itu (yang anda akan menulis) mestilah menggunakan panggilan-panggilan yang diterimanya menerusi fungsi `Tick` untuk mengukur masa yang berlalu.

Fungsi `VehicleWaiting` ialah satu lagi titik masuk dalam aplikasi itu. Ia dipanggil oleh sistem pengendalian lampu isyarat lalulintas bila-bila masa sistem pengendalian itu mengesan sesuatu perubahan pada keadaan alat pengesan kenderaan. Parameter-parameter northWait, eastWait, southWait dan westWait adalah sifar sekiranya tiada kenderaan yang sedang menunggu di cabang-cabang yang berkaitan dan bukan sifar di sebaliknya.

Program antaramuka Java terdiri daripada antaramuka-antaramuka Java yang berikut:

```
interface TrafficSystem {
     static final int   RED = 0;
     static final int   AMBER = 1;
     static final int   GREEN = 2;
                    // Kod-kod warna lampu lalulintas
     void display (int northColour, int eastColour,
          int southColour, int westColour);
```

```
                    // Tetapkan warna-warna lampu
                    // isyarat lalulintas
}

class TrafficApp {
     abstract void initApp (TrafficSystem ts);
                    // Mulakan aplikasi
     abstract void tick ();
                    // Proses detik jam
     abstract void vehicleWaiting (int northWait,
          int eastWait, int southWait, int westWait);
                    // Proses perubahan pada keadaan
                    // alat pengesan kenderaan
}
```

Antaramuka `TrafficSystem` mewakili sistem pengendalian lampu isyarat lalulintas.

Kaedah (*method*) `display` dalam antaramuka `TrafficSystem` dipanggil oleh aplikasi lampu isyarat lalulintas untuk menetapkan warna-warna yang dipamerkan oleh taing lampu isyarat lalulintas. Parameter-parameter `northColour`, `eastColour`, `southColour` dan `westColour` adalah warna-warna yang akan ditunjukkan ke arah cabang utara, timur, selatan and barat masing-masing. Nilai mereka bolehlah salah satu daripada warna `RED`, `AMBER` atau `GREEN`.

Kelas (*class*) `TrafficApp` ialah *super-class* bagi aplikasi lampu lalulintas. Aplikasi lampu lalulintas itu mestilah *override* kaedah-kaedah `initApp`, `tick` dan `vehicleWaiting`.

Kaedah `initApp` dipanggil oleh sistem pengendalian lampu lalulintas untuk memperuntukkan nilai awal kepada sebarang data aplikasi. Kaedah ini mestilah memberi nilai permulaan kepada sebarang data aplikasi dan kemudiannya kembali serta-merta. Aplikasi itu tidak patut memegang kawalan dalam kaedah `initApp`. Kaedah `initApp` mesti menyimpan rujukan (*reference*) kepada *instance* `TrafficSystem` supaya aplikasi itu boleh memanggil kaedah `display` kemudian. Kaedah `initApp` harus memanggil `display` untuk menetapkan keadaan permulaan lampu lalulintas.

Kaedah `tick` dipanggil oleh sistem pengendalian lampu lalulintas sekali setiap saat. Aplikasi itu (yang anda akan menulis) mesti menggunakan panggilan-panggilan ia menerima menerusi kaedah `tick` untuk mengukur masa yang berlalu.

Kaedah vehicleWaiting dipanggil oleh sistem pengendalian lampu lalulintas bila-bila masa sistem pengendalian itu mengesan sesuatu perubahan pada keadaan alat pengesan kenderaan menunggu. Parameter-parameter northWait, eastWait, southWait dan westWait adalah sifar sekiranya tiada kenderaan yang sedang menunggu di cabang-cabang yang berkaitan dan bukan sifar di sebaliknya.