```
// RouteTime.cpp - FIND THE THREE EARLIST ARRIVAL TIMES
//
// USAGE
// RouteTime startTime fromStationId toStationId
// startTime        is the journey start time in seconds from midnight
// fromStationId    is the station identifier of the starting station
// toStationId      is the station identifier of the destination station
//
// MODULE INDEX
// NAME         CONTENTS
// DcdNum       Decode a number
// main         Main line
//
// MAINTENANCE HISTORY
// DATE     PROGRAMMER AND DETAILS
// 29-09-16 MPF Original
//
//-------------------------------------------------------------------------------

#include <vector>        // C++ vector declaration
#include <set>           // C++ set declaration
#include <map>           // C++ map declaration
#include <iostream>      // C++ I/O stream declaration
using namespace std;
exec sql include sqlca; // Include SQL communications area

//-------------------------------------------------------------------------------

// STOP STRUCTURE

struct Stop_t {
    long            stopArrivalTime; // Arrival time to station
    long            stopDepartTime; // Departure time at station
    long            stopStatId;     // Destination station identifier
    long            stopTrainId;    // Train identifier

    bool
    operator < (const Stop_t &stop)
    const
    {
        if (stopArrivalTime < stop.stopArrivalTime)
            return 1;
        else if (stopArrivalTime > stop.stopArrivalTime)
            return 0;
        if (stopDepartTime < stop.stopDepartTime)
            return 1;
        else if (stopDepartTime > stop.stopDepartTime)
            return 0;
        else
            return stopTrainId < stop.stopTrainId;
    }
};

//-------------------------------------------------------------------------------

// TRAIN STRUCTURE
```

```
struct Train_t {
    long                trainId;        // Train identifier
    long                trainStatId;    // Current station
    long                trainTime;      // Current time
    long                trainOnBoardTime; // Time on board the train
    vector<Stop_t>      trainStops;     // Stops
};

//-----------------------------------------------------------------------------

// DECODE NUMBER

bool
DcdNum (
    register char       *p,             // Pointer to string
    long                *val)           // Value
{
    *val = 0;

    while (*p != '\0') {
        if (*p < '0' || *p > '9') return 0;
        *val *= 10;
        *val += *p - '0';
        p ++ ;
    }
    return 1;
}

//-----------------------------------------------------------------------------

// MAIN LINE

int
main (
    int                 argc,           // Argument count
    char                *argv[])        // Argument value
{
    bool                trainDef;       // Train defined flag
    Train_t             train;          // Current train
    Stop_t              stop;           // Current stop
    set<Stop_t>         stops;          // Stops
    set<Stop_t>::iterator   stopIter;    // Stop iterator
    map<long, Train_t>  trains;         // Trains
    map<long, Train_t>::iterator trainIter; // Train iterator
    vector<long>        arrivals;       // Arrival times
    vector<long>::iterator  arrivalIter; // Arrival time iterator
    set<long>           stations;       // Stations
    long                startTime;      // Starting time
    long                fromStatId;     // Starting station
    long                toStatId;       // Destination station

    exec sql begin declare section;
        long            trainId;        // Train identifier
        long            statId;         // Station identifier
        long            time;           // Time
        long            seqNo;          // Sequence number
        long            nextStatId;     // Next station identifier
        long        waitTime;       // Waiting time
        long        duration;       // Duration
    exec sql end declare section;

    // Get the parameter values

    if (argc != 4) {
        cerr << "USAGE: RouteTime startTime fromStationId toStationId"
            << '\n';
        return 1;
    }

    if ( ! DcdNum (argv[1], &startTime)) {
        cerr << "Invalid start time\n";
        return 1;
    }

    if ( ! DcdNum (argv[2], &fromStatId)) {
        cerr << "Invalid starting station identifier\n";
        return 1;
    }

    if ( ! DcdNum (argv[3], &toStatId)) {
        cerr << "Invalid destination station identifier\n";
        return 1;
    }

    exec sql connect to egpc;

    exec sql whenever sqlerror goto db_error;

    // Get the earliest applicable trains

    exec sql declare trainCur cursor for
        select  trainId, trainFirstStatId, trainStartTime
        from    train;
    exec sql open trainCur;
    for (;;) {
        exec sql fetch  trainCur
            into    :trainId, :statId, :time;
        if (SQLCODE != 0) break;

        trainDef = 0;
        train.trainStops.clear ();

        if (time > startTime) {
            trainDef = 1;
            train.trainId = trainId;
            train.trainStatId = statId;
            train.trainTime = time;
            train.trainOnBoardTime = -1;
        }

        exec sql declare routeCur cursor for
            select  routeSeqNo, routeNextStatId, routeNextWaitTime
            from    route
            where   routeTrainId = :trainId
            order   by routeSeqNo;
        exec sql open routeCur;
        for (;;) {
            exec sql fetch  routeCur
                into    :seqNo, :nextStatId, :waitTime;
            if (SQLCODE != 0) break;

            exec sql select trackDuration
                into    :duration
                from    track
                where   trackFromStatId = :statId and
                    trackToStatId = :nextStatId;
            if (SQLCODE != 0) goto db_error;

            if (trainDef) {
                stop.stopArrivalTime = time + duration;
                stop.stopDepartTime =
                    time + duration + waitTime;
                stop.stopStatId = nextStatId;
                stop.stopTrainId = trainId;
                train.trainStops.push_back (stop);
                stops.insert (stop);
            } else if (time + duration + waitTime > startTime) {
                trainDef = 1;
                train.trainId = trainId;
                train.trainStatId = nextStatId;
                train.trainTime = time + duration + waitTime;
                train.trainOnBoardTime = -1;
                break;
            }
            time  += duration + waitTime;
            statId = nextStatId;
        }
        exec sql close routeCur;

        if (trainDef && train.trainStops.size() > 0)
            trains[trainId] = train;
    }
    exec sql close trainCur;

    // If starting at destination, arrival time is the start time

    if (fromStatId == toStatId)
        arrivals.push_back (startTime);

    // Board all trains at the station

    for (
        trainIter = trains.begin();
        trainIter != trains.end();
        trainIter ++
    ) {
        if (trainIter->second.trainStatId == fromStatId)
            trainIter->second.trainOnBoardTime =
                trainIter->second.trainTime;
    }
    stations.insert(fromStatId);

    while (arrivals.size() < 3 && trains.size() > 0) {
        stopIter = stops.begin();
        trainIter = trains.find(stopIter->stopTrainId);

        // For diagnostic use

        //cout << "Train " << trainIter->second.trainId
        //  << " from " << trainIter->second.trainStatId
        //  << " at " << trainIter->second.trainTime
        //  << " to " << stopIter->stopStatId
        //  << " at " << stopIter->stopArrivalTime
        //  << '\n';

        // Board the train if the station is visited

        if (trainIter->second.trainOnBoardTime < 0) {
            if (
                stations.find(trainIter->second.trainStatId) !=
                    stations.end()
            ) {
                // For diagnostic use

                //cout << "Boarded:\ttrain "
                //  << trainIter->second.trainId
                //  << "\tfrom station "
                //  << trainIter->second.trainStatId
                //  << "\tat time "
                //  << trainIter->second.trainTime << '\n';
                trainIter->second.trainOnBoardTime =
                    trainIter->second.trainTime;
            }
        }

        // Already boarded the train

        if (trainIter->second.trainOnBoardTime >= 0) {

            // For diagnostic use

            //cout << "Visited:\tstation " << stopIter->stopStatId
            //  << "\tat time " << stopIter->stopArrivalTime
            //  << "\tvia train " << stopIter->stopTrainId
            //  << ((stopIter->stopStatId == toStatId)
            //      ? "\t**** " : "")
            //  << '\n';

            // If arriving at destination, record arrival time

            if (stopIter->stopStatId == toStatId)
                arrivals.push_back (stopIter->stopArrivalTime);

            // Visited the station

            stations.insert (stopIter->stopStatId);
        }
        // Update the train location and time

        trainIter->second.trainStatId = stopIter->stopStatId;
        trainIter->second.trainTime = stopIter->stopDepartTime;

        // Remove processed stop

        stops.erase(stopIter);
    }

    for (
        arrivalIter = arrivals.begin();
        arrivalIter != arrivals.end();
        arrivalIter ++
    )
        cout << (*arrivalIter - startTime) << '\n';

    return 0;

db_error:
    cerr << "Error: SQLCODE=" << SQLCODE << '\n';
    return 1;
}
```