```
#ifndef NODE_H
#define NODE_H

class Node
{
public:
    Node(int number, bool isMedical);
    int getNumber();
    bool getIsMedical();
private:
    int number;
    bool isMedical;
};

#endif // NODE_H
```

```
#include "node.h"

//Model representing a node on the course

Node::Node(int number, bool isMedical){
    this->number = number;
    this->isMedical = isMedical;
}

int Node::getNumber(){
    return this->number;
}

//check if the node is a medical checkpoint
bool Node::getIsMedical(){
    return this->isMedical;
}
```

```
#include "checkpointmanager.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    if(argc<4){
        std::cout << "\n\nPlease run the application as follows"
                " \"./CheckpointManager node.txt entrants.txt course.txt"
                " times.txt\" without the qoutes\n\n\n\n";
        return 0;
    }
    QApplication a(argc, argv);
    CheckpointManager w;
    w.show();

    return a.exec();
}
```

```
#ifndef ENTRANT_H
#define ENTRANT_H
#include <iostream>
#include "course.h"

class Entrant
{
public:
    Entrant(int id, std::string string, Course* course);
    int getId();
    void reset();
    Node* nextNode();
    void incrementVisitedNodes();
    std::string getName();
private:
    Course* course;
    int id;
    std::string name;
    int visitedNodes;
};

#endif // ENTRANT_H
```

```
#include "entrant.h"
#include <iostream>

//Model representing an entrant on the course

Entrant::Entrant(int id, std::string name, Course* course){
    this->id = id;
    this->name = name;
    this->visitedNodes = 0;
    this->course = course;
}

int Entrant::getId(){
    return this->id;
}

//Called when an entrant reachs a new checkpoint
void Entrant::incrementVisitedNodes(){
    this->visitedNodes++;
}

//Resets the entrant, used whe reloading
void Entrant::reset(){
    this->visitedNodes = 0;
}

//Returns the next node, this entrant is expected at
Node* Entrant::nextNode(){
    return this->course->getCheckpointNo(this->visitedNodes);
}

std::string Entrant::getName(){
    return this->name;
}
```

```
#ifndef COURSE_H
#define COURSE_H

#include <iostream>
#include <vector>
#include "node.h"

class Course
{
public:
    Course(std::string id);
    void addCheckpoint(Node* checkpoint);
    Node* getCheckpointNo(int no);
    std::string getId();
private:
    std::string id;
    std::vector<Node*> checkpoints;
};

#endif // COURSE_H
```

```
#include "course.h"
#include <stdexcept>

//The courses class represents a model of a course

Course::Course(std::string id)
{
    this->id = id;
}


void Course::addCheckpoint(Node* checkpoint){
    this->checkpoints.push_back(checkpoint);
}

//Get checkpoint number X
Node* Course::getCheckpointNo(int no){
    try {
        return this->checkpoints.at(no);
    }catch (const std::out_of_range& oor) {
        return NULL;
    }
}

std::string Course::getId(){
    return this->id;
}
```

```
#ifndef CHECKPOINTMANAGER_H
#define CHECKPOINTMANAGER_H

#include <QWidget>
#include "node.h"
#include "entrant.h"
#include "course.h"

namespace Ui {
class CheckpointManager;
}

class CheckpointManager : public QWidget
{
    Q_OBJECT

public:
    explicit CheckpointManager(QWidget *parent = 0);
    std::string nodesFilePath;
    std::string entrantsFilePath;
    std::string coursesFilePath;
    std::string timesFilePath;

    ~CheckpointManager();
public slots:
    void checkpointChanged(int);
    void entrantChanged();
    void entrantExcludedChanged();
    void submitPressed();

private:
    int currentState; //Current state of the form (0 unfinished, 1 time,
    //2 medical, 3 excluded)

    void loadNodes();
    void loadEntrants();
    void hideElements();

    Course * getCourse(std::string courseId);
    Node * getNode(int nodeNumber);
    Entrant * getEntrant(int id);

    void removeEntrant(int id);

    void resetEntrants();

    void parseNodesFile();
    void parseEntrantsFile();
    void parseCoursesFile();
    void parseTimesFile();

    Ui::CheckpointManager *ui;
    std::vector<Node> nodes;
    std::vector<Entrant> entrants;
    std::vector<Course> courses;
};

#endif // CHECKPOINTMANAGER_H
```

```
#include "checkpointmanager.h"
#include "ui_checkpointmanager.h"
#include <QMessageBox>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>

#include <iostream>

using namespace std;

CheckpointManager::CheckpointManager(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::CheckpointManager)
{
    ui->setupUi(this);
    QStringList cmdline_args = QCoreApplication::arguments();

    this->nodesFilePath = cmdline_args.at(1).toStdString();
    this->entrantsFilePath = cmdline_args.at(2).toStdString();
    this->coursesFilePath = cmdline_args.at(3).toStdString();
    this->timesFilePath = cmdline_args.at(4).toStdString();

    //Parse the needed files for an event
    parseNodesFile();
    parseCoursesFile();
    parseEntrantsFile();
    parseTimesFile();

    //Load nodes and entrant's into the combobox
    loadNodes();
    loadEntrants();

    hideElements();

    //Resize window to the smallest size without ruining content
    resize(sizeHint());
}

//Hides and resets all UI elements
void CheckpointManager::hideElements(){
    ui->lblCheckpointType->setVisible(false);
    ui->lblCheckpointValue->setVisible(false);

    ui->lblEntrant->setVisible(false);
    ui->inptEntrant->setVisible(false);

    ui->lblArrival->setVisible(false);
    ui->inptArrivalTime->setVisible(false);

    ui->lblEntrantExcluded->setVisible(false);
    ui->inptExcluded->setVisible(false);
    ui->inptExcluded->setCurrentIndex(0);

    ui->lblDTime->setVisible(false);
    ui->inptDTime->setVisible(false);

    ui->lblNote->setText(QString(""));
    ui->lblNote->setVisible(false);

    ui->inptSubmit->setVisible(false);

    this->currentState = 0;
}
```

```
//Loads Nodes into the inptNode Combobox
void CheckpointManager::loadNodes(){
    ui->inptNode->clear();
    ui->inptNode->addItem("Select a checkpoint", QVariant(-1));
    for(vector<Node>::iterator node = nodes.begin(); node!=nodes.end(); ++node){
        //Add the node to the combobox and set QVariant to help indentify the node
        ui->inptNode->addItem(
        QString::number((*node).getNumber()), QVariant((*node).getNumber()));
    }
}

//Loads entrants into the inpEntrant Combobox
void CheckpointManager::loadEntrants(){
    ui->inptEntrant->addItem("Select an entrant", QVariant(-1));
    for(vector<Entrant>::iterator entrant =
            entrants.begin(); entrant!=entrants.end(); ++entrant){
        //Add the entrant to the combobox and set QVariant to help indentify
        //the node
        ui->inptEntrant->addItem(QString((*entrant).getName().c_str()),
                QVariant((*entrant).getId()));
    }
}

//Called when inptNode's value is chnaged (E.g. a new node is selected)
void CheckpointManager::checkpointChanged(int index){
    int nodeNumber = ui->inptNode->itemData(index).toInt();
    if(nodeNumber == -1){
        //If its been set to the place holder, hide all elements
        hideElements();
    }else{
        //Determine node type
        if(getNode(nodeNumber)->getIsMedical()){
            ui->lblCheckpointValue->setText(QString("Medical checkpoint"));
        }else{
            ui->lblCheckpointValue->setText(QString("Time checkpoint"));
        }
        entrantChanged();

        ui->lblCheckpointType->setVisible(true);
        ui->lblCheckpointValue->setVisible(true);
    }
    resize(sizeHint());
}

//Called when inptEntant's index is changed
void CheckpointManager::entrantChanged(){
    hideElements();

    ui->lblCheckpointType->setVisible(true);
    ui->lblCheckpointValue->setVisible(true);

    ui->lblEntrant->setVisible(true);
    ui->inptEntrant->setVisible(true);

    int entrantId =
    ui->inptEntrant->itemData(ui->inptEntrant->currentIndex()).toInt();
    if(entrantId != -1){
        parseTimesFile();
        int nodeId = ui->inptNode->itemData(ui->inptNode->currentIndex()).toInt();

        Entrant * entrant = this->getEntrant(entrantId);
        if(entrant != NULL){
            Node * node = this->getNode(nodeId);

            ui->lblArrival->setVisible(true);
            //Set default arrival time to now, to help user
            ui->inptArrivalTime->setTime(QTime::currentTime());
            ui->inptArrivalTime->setVisible(true);
```

```
            if((*entrant).nextNode()->getNumber()==(*node).getNumber()){
                //If entrant is on the right route
                if((*node).getIsMedical()){
                    //If medical checkpoint, show medial info
                    ui->lblEntrantExcluded->setVisible(true);
                    ui->inptExcluded->setVisible(true);

                    ui->lblNote->setVisible(true);

                    ui->lblDTime->setVisible(true);
                    ui->inptDTime->setVisible(true);
                    this->currentState = 2;
                }else{
                    //Set the time of Dtime to current time to help user
                    ui->inptDTime->setTime(QTime::currentTime());
                    this->currentState = 1;
                }
            }else{
                //Entrant has gone the wrong way
                ui->lblNote->setText(QString("Note: This entrant will be "
                "excluded for going the wrong way"));
                ui->lblNote->setVisible(true);
                this->currentState = 3;
            }

            ui->inptSubmit->setVisible(true);
        }else{
            //Remove null pointer's
            ui->inptEntrant->removeItem(ui->inptEntrant->currentIndex());
        }
    }
    resize(sizeHint());
}

//Called when the entrantExcluded combobox is changed.
void CheckpointManager::entrantExcludedChanged(){
    if(ui->inptExcluded->currentIndex()==1){
        //Show label
        ui->lblNote->setText(QString("Note: This entrant will be excluded for "
        "medical reasons"));
        ui->lblNote->setVisible(true);
    }else{
        //Hide label
        ui->lblNote->setVisible(false);
    }
    resize(sizeHint());
}

//Called when submit is pressed. In charge of saving to file
void CheckpointManager::submitPressed(){
    //Open time file
    int fd = open(timesFilePath.c_str(), O_RDWR);
    struct flock fl;
    if (fd == -1) {
        //Change for file descriptor lock
        QMessageBox msgBox;
        msgBox.setText("Error!");
        msgBox.setInformativeText("The times file is currently being used by "
        "someone else, please wait and try submit again.");
        msgBox.setIcon(QMessageBox::Critical);
        msgBox.exec();
    }else{

        fl.l_type = F_WRLCK;
        fl.l_whence = SEEK_SET;
        fl.l_start = 100;
        fl.l_len = 10;
        //Check if file is currently locked
```

```
if (fcntl(fd, F_SETLK, &fl) == -1) {
    if (errno == EACCES || errno == EAGAIN) {
        QMessageBox msgBox;
        msgBox.setText("Error!");
        msgBox.setInformativeText("The times file is "
        "currently being used by someone else, please wait "
        "and try submit again.");
        msgBox.setIcon(QMessageBox::Critical);
        msgBox.exec();
    } else {
        QMessageBox msgBox;
        msgBox.setText("Error!");
        msgBox.setInformativeText("Unexpected file error, try again.");
        msgBox.setIcon(QMessageBox::Critical);
        msgBox.exec();
    }
} else {
    //Open file for appending
    FILE* timesFileToWrite = fopen(timesFilePath.c_str(), "a+");

    int entrantId =
    ui->inptEntrant->itemData(ui->inptEntrant->currentIndex()).toInt();
    int nodeId =
    ui->inptNode->itemData(ui->inptNode->currentIndex()).toInt();

    Entrant * entrant = this->getEntrant(entrantId);
    Node * node = this->getNode(nodeId);
    QTime arrivalTime = ui->inptArrivalTime->time();

    //Time checkpoint
    if(this->currentState==1){
        fprintf(timesFileToWrite,"T %d %d %02i:%02i\n",
                node->getNumber(), entrant->getId(), arrivalTime.hour(),
                arrivalTime.minute());
    }
    //Medical Checkpoint
    if(this->currentState==2){
        QTime dTime = ui->inptDTime->time();
        fprintf(timesFileToWrite,"A %d %d %02i:%02i\n", node->getNumber(),
                entrant->getId(), arrivalTime.hour(),
                arrivalTime.minute());
        if(ui->inptExcluded->currentIndex()==0)
            fprintf(timesFileToWrite,"D %d %d %02i:%02i\n",
                    node->getNumber(), entrant->getId(), dTime.hour(),
                    dTime.minute());
        else
            fprintf(timesFileToWrite,"E %d %d %02i:%02i\n",
                    node->getNumber(), entrant->getId(), dTime.hour(),
                    dTime.minute());
    }
    //Excluded
    if(this->currentState==3){
        fprintf(timesFileToWrite,"I %d %d %02i:%02i\n", node->getNumber(),
                entrant->getId(), arrivalTime.hour(), arrivalTime.minute());
    }
    fclose(timesFileToWrite);

    fl.l_type = F_UNLCK;
    fl.l_whence = SEEK_SET;
    fl.l_start = 100;
    fl.l_len = 10;
    //Unlock file
    if (fcntl(fd, F_SETLK, &fl) == -1){
        QMessageBox msgBox;
        msgBox.setText("Error!");
        msgBox.setInformativeText("Unexpected error while unlocking file");
        msgBox.setIcon(QMessageBox::Critical);
        msgBox.exec();
    }else{
```

```
                QMessageBox msgBox;
                msgBox.setText("Time added successfully!");
                msgBox.setInformativeText("The entrant time has been added.");
                msgBox.exec();

                //Reset all elements so we can add another time.
                hideElements();
                parseTimesFile();
                ui->inptEntrant->setCurrentIndex(0);
                ui->inptNode->setCurrentIndex(0);
                entrants.empty();
                parseEntrantsFile();
                parseTimesFile();
            }
        }
    }

}

//Returns a node points from a node id
Node * CheckpointManager::getNode(int nodeNumber){
    for(vector<Node>::iterator node = nodes.begin(); node!=nodes.end(); ++node){
        if((*node).getNumber()==nodeNumber)
            return &(*node);
    }
    return NULL;
}

//Returns a course pointer from a courseId
Course * CheckpointManager::getCourse(string courseId){
    for(vector<Course>::iterator course = courses.begin();
            course!=courses.end(); ++course){
        if((*course).getId().compare(courseId)==0)
            return &(*course);
    }
    return NULL;
}

//Returns an entrant from an entrantId
Entrant * CheckpointManager::getEntrant(int id){
    for(vector<Entrant>::iterator entrant = entrants.begin();
            entrant!=entrants.end(); ++entrant){
        if((*entrant).getId()==id)
            return &(*entrant);
    }
    return NULL;
}

//Resets an entrant
void CheckpointManager::resetEntrants(){
    for(vector<Entrant>::iterator entrant = entrants.begin();
            entrant!=entrants.end(); ++entrant){
        (*entrant).reset();
    }
}


//Removes an entrant from the list and comobox when theyre no longer needed
void CheckpointManager::removeEntrant(int id){
    for(vector<Entrant>::iterator entrant = entrants.begin();
            entrant!=entrants.end(); ++entrant){
        int entrantId = (*entrant).getId();
        if(entrantId==id){
            ui->inptEntrant->removeItem(ui->inptEntrant->findData(QVariant(entrantId)));
            entrants.erase(entrant);
            break;
        }
    }
}
```

```
//Parsenodes file
void CheckpointManager::parseNodesFile(){
    ifstream nodesFile(nodesFilePath.c_str(), ios::in);

    if (!nodesFile){
        QMessageBox msgBox;
         msgBox.setText("Error!");
         msgBox.setInformativeText("Error parsing nodes file. The "
         "program will now exit.");
         msgBox.setIcon(QMessageBox::Critical);
         msgBox.exec();
         exit (EXIT_FAILURE);
    }

    while(nodesFile){
        string nodeNumber;
        string nodeType;
        getline(nodesFile, nodeNumber, ' ');
        getline(nodesFile, nodeType);
        if((nodeType.compare("JN")!=0) && (!nodeNumber.empty()) ){
            bool isMedical = (nodeType.compare("MC")==0);
            int nodeNo = atoi(nodeNumber.c_str());
            nodes.push_back(Node(nodeNo, isMedical));
        }
    }
    nodesFile.close();
}


//Parse entrants file
void CheckpointManager::parseEntrantsFile(){
    ifstream entrantsFile(entrantsFilePath.c_str(), ios::in);

    if (!entrantsFile){
        QMessageBox msgBox;
         msgBox.setText("Error!");
         msgBox.setInformativeText("Error parsing entrants file. The program "
         "will now exit.");
         msgBox.setIcon(QMessageBox::Critical);
         msgBox.exec();
         exit (EXIT_FAILURE);
    }

    while(entrantsFile){
        string entrantId, courseId, entrantName;
        getline(entrantsFile, entrantId, ' ');
        getline(entrantsFile, courseId, ' ');
        getline(entrantsFile, entrantName);
        if(!entrantId.empty()){
            int entrantIdNo = atoi(entrantId.c_str());
            Course* entrantCourse = getCourse(courseId);
            entrants.push_back(Entrant(entrantIdNo, entrantName, entrantCourse));
        }
    }
    entrantsFile.close();
}

//Parse courses file
void CheckpointManager::parseCoursesFile(){
    ifstream coursesFile(coursesFilePath.c_str(), ios::in);

    if (!coursesFile){
        QMessageBox msgBox;
         msgBox.setText("Error!");
         msgBox.setInformativeText("Error parsing courses file. The program will "
         "now exit.");
         msgBox.setIcon(QMessageBox::Critical);
```

```
         msgBox.exec();
         exit (EXIT_FAILURE);
    }

    while(coursesFile){
        string courseId;
        string noNodes;

        getline(coursesFile, courseId, ' ');
        getline(coursesFile, noNodes, ' ');
        if(!courseId.empty()){
            int intNoNodes = atoi(noNodes.c_str());
            Course* newCourse = new Course(courseId);
            string nodeNo;
            for(int i = 0; i<(intNoNodes); i++){
                //If not the last node, look for ' ' delimineters
                if(i<(intNoNodes-1))
                    getline(coursesFile, nodeNo, ' ');
                else
                    getline(coursesFile, nodeNo);
                    //Last node, so just read till read of line

                Node* nodeToAdd = getNode(atoi(nodeNo.c_str()));
                if(nodeToAdd!=NULL){
                    (*newCourse).addCheckpoint(nodeToAdd);
                }
            }
            this->courses.push_back((*newCourse));
        }
    }
    coursesFile.close();
}

//Parses a times file
void CheckpointManager::parseTimesFile(){
    resetEntrants();

    ifstream timesFile(timesFilePath.c_str());

    if (!timesFile){
        std::ofstream newfile (timesFilePath.c_str(), ios::in);
        if(!newfile.is_open()){
            QMessageBox msgBox;
             msgBox.setText("Error!");
             msgBox.setInformativeText("Error parsing times file. The program "
             "will now exit.");
             msgBox.setIcon(QMessageBox::Critical);
             msgBox.exec();
             exit (EXIT_FAILURE);
        }
        newfile.close();
    }else{

        while(timesFile){
            string nodeType, nodeId, entrantId, time;
            //Read information from the line
            getline(timesFile, nodeType, ' ');
            getline(timesFile, nodeId, ' ');
            getline(timesFile, entrantId, ' ');
            getline(timesFile, time);
            if(!nodeType.empty()){
                int intEntrantId = atoi(entrantId.c_str());
                if((nodeType.compare("T")==0)||(nodeType.compare("D")==0)){
                    Entrant * entrant = getEntrant(intEntrantId);
                    if(entrant!=NULL){
                        (*entrant).incrementVisitedNodes();
                        if((*entrant).nextNode()==NULL){
                            //Remove finished/excluded entrants
                            removeEntrant((*entrant).getId());
```

```
                        }
                    }
                }else if((nodeType.compare("I")==0)&&(nodeType.compare("E")==0)){
                    removeEntrant(intEntrantId);
                }
            }
        }
    }
    timesFile.close();
}


//Deconstuctor
CheckpointManager::~CheckpointManager(){
    delete ui;
}
```