

# Computer Science Large Practical 2015–2016

Dr Paul Patras  
School of Informatics

Issued on: Friday 25<sup>th</sup> September, 2015.

**Updated on: Tuesday 27<sup>th</sup> October, 2015.**

The CSLP coursework handout is structured as follows:

---

<b>Coursework Description</b>	<b>2</b>
1 Introduction . . . . .	2
2 Requirements . . . . .	2
2.1 Problem Domain . . . . .	3
2.2 Simulation Outline . . . . .	3
2.3 Simulation Components . . . . .	4
2.4 Command-line Arguments . . . . .	6
2.5 Input Formatting . . . . .	6
2.6 Output Formatting . . . . .	8
2.7 Statistics Analysis . . . . .	9
2.8 Experimentation . . . . .	10
2.9 Visualisation . . . . .	11
2.10 Testing . . . . .	11
3 Frequently Asked Questions . . . . .	12
<b>Part 1</b>	<b>13</b>
1 Introduction . . . . .	13
2 Description . . . . .	13
3 What to Submit . . . . .	14
4 How to Submit . . . . .	14
5 Deadline . . . . .	14
6 Frequently Asked Questions and Clarification . . . . .	14
<b>Part 2</b>	<b>15</b>
1 Introduction . . . . .	15
2 Description . . . . .	15
3 Additional Credit . . . . .	16
4 Assessment . . . . .	16
5 Assessment Criteria . . . . .	16
6 What to Submit . . . . .	17
7 How to Submit . . . . .	18
8 Deadline . . . . .	18
9 Marking . . . . .	18
10 Frequently Asked Questions and Clarification . . . . .	18

---

# 1 Introduction

The requirement for the Computer Science Large Practical is to develop a command-line application in the C programming language. The purpose of the application is to *execute stochastic simulations of an on-demand public transport system* for future cities, in order to gain insights into how the efficiency of this operation and the customers' satisfaction can be optimised.

— ◇ —

Stochastic simulation is an important tool in many fields such as logistics, computer networking, physics, medicine, etc. and is particularly useful to understand complicated processes. Unlike deterministic simulations, stochastic simulations with the same input may produce different outputs. This is due to the fact that some events are intrinsically random.

— ◇ —

Different to previous software development assignments you may have had, this practical will expose you to the design and implementation of a more complex system that incorporates realistic constraints specific to large processes. Although a set of requirements is given below, some of these are intentionally incomplete, to allow you the choice of implementation methodology. As this is a relatively large task, your source code must be clean, commented and reusable. This is an individual assignment and thus code sharing is not permitted.

— ◇ —

You are expected to produce a written report that explains the key building blocks of your design, discusses the results of the analyses you performed with different inputs, and summarises your most important findings. These can be accompanied by graphs plotted based on the numerical output of your simulations. There should be evidence that your implementation has been tested, including the submission of input files you have generated yourself. Take time to understand if the produced output is sensible, i.e. the numerical values obtained would make sense in a real setting.

It is important that you read and implement carefully the requirements of the assignment. In particular, submissions that do not accept correctly formatted input or produce output with incorrect format will receive fewer marks.

— ◇ —

In what follows the requirements of your application are detailed, including input formatting, the expected application behaviour, and the format of the output. There are two deadlines associated with this coursework. The first deadline is optional and is an opportunity for you to submit an early version of your code in order to obtain feedback prior to your final submission. The report and code you submit for the second deadline weight 100% of the final mark. Further details are given in parts 1 and 2.

## 2 Requirements

In this section the requirements of the application you must submit are discussed.

## 2.1 Problem Domain

To be able to simulate a real system, first it is required to construct a model that captures the most important aspects of the system that will be studied. Not all the features of a real system may be relevant to a specific problem. The challenge is to identify the important elements to be modelled and omit others to reduce complexity, while still obtaining information that helps understanding the system's behaviour.

Different inputs can be given to a well designed simulator to obtain estimates of different parameters of interest. Such simulation-based studies are quicker and significantly cheaper than approaches based on extensive measurements of the real system. In addition, simulations allow to investigate exceptional scenarios that are foreseeable, but for which a real data set is not yet available.

Our focus is on simulating an on-demand public transport system. We are interested in future city scenarios where minibuses will allow passengers to share their journeys with others who intend to travel similar itineraries, instead of following predefined routes. This approach is currently considered by European city councils as a potential major step towards eliminating the incentives for personal car ownership, thereby significantly reducing carbon emissions in densely populated areas. At the same time, such public ride sharing services would enable cutting down commute times and improve customer satisfaction, while replacing periodic bus schedules that often prove inefficient, since service demands vary geographically and in time.

In this practical we will build a simulator of such an on-demand transport system, where customers could choose pick-up locations among a list of bus stops and desired departure times, as well as their destination stops. We will investigate the efficiency of the transportation process in terms of bus occupancy per unit of travel time when users can tolerate different pick-up delays. We will consider each minibus has a specified number of seats and the distances and road layouts between stops are known. Each bus will station at the last stop or at the garage when not in service and will be scheduled once identified as the nearest to newly placed user requests. Route planning decisions will be made based on configurable user waiting time limits, and the differences between the time requests are placed and the desired departure time. Routes can also be updated in real time as new users enter the system. Shorter durations between requests and desired departure times, as well as short maximum tolerable waiting times can potentially lead to journeys with fewer passengers and thus yield lesser efficiency. On the other hand planning with longer waiting intervals can prove more cost efficient, but may negatively impact customer satisfaction, which we will evaluate in terms of different metrics. The simulator will further allow us to understand whether a system is under provisioned, i.e. user requests for journeys cannot be accommodated or pick-ups cannot be satisfied within a given maximum admissible waiting time.

## 2.2 Simulation Outline

The underlying simulation algorithm is fairly straightforward and is outlined below.

```
time ← 0
while time ≤ max_time do
    determine the set of events that may occur after the current state
    delay ← choose a delay based on the nearest event
    time ← time + delay
    modify the state of the system based on the current event
end while
```

A couple of clarifications with respect to the above pseudo-code are appropriate. The set of possible events include (i) the arrival of a new customer request in the system, (ii) the departure of a bus from a location, (iii) the arrival of a bus at a stop, (iv) the disembarkation of a passenger, and respectively (v) the boarding of a new passenger.

Requests arrive in the system in a stochastic fashion. They comprise of a desired pick-up time, a source stop, and a target (destination) stop. The time between requests and the delay between a request and its corresponding pick-up time are chosen by sampling exponentially distributed random variables with given means. (When the mean rate is given as input, the mean delay is computed as the reciprocal of the rate.) You can draw from an exponential distribution with mean delay  $\lambda$ , by computing  $-\lambda \cdot \log(\text{rand}(0.0, 1.0))$ , where  $\text{rand}(0.0, 1.0)$  returns a random number between 0.0 and 1.0 (exclusive). The source and target stops for each request are chosen uniformly at random. All of these are independent of each other, and of other requests, and identically distributed (i.i.d.). That is the event of a new pick-up request from a given stop does not depend on previous events at the same or other stops.

When determining the set of events that may occur after the current state, each of these will have an associated delay. To advance the simulation, you will first need to find the closest one, i.e. the one with the smallest delay. Then you will update the simulation time with this delay, execute the corresponding event and update the system states accordingly.

This procedure is repeated until `max_time` expires.

## 2.3 Simulation Components

Your simulations must include the following elements:

- **Stops.** Passengers will only board or disembark at designated minibus stops. In the envisioned system users book journeys through e.g. a smartphone application and do not necessarily need to be present at a bus stop upon such a request. However, once a request enters the system, this has to be queued in view of scheduling. The requests queue at a stop could thus be virtually infinite, but for practical reasons in our system we will consider a maximum number of requests queued as at a stop does not exceed  $2^{16} - 1$ . We are only interested in the back-end system, thus developing mobile phone applications used by users is outside the scope of this practical.
- **Users.** We consider users place journey requests at exponentially distributed time intervals with the mean delay given as an input parameter. A request will comprise randomly chosen departure and destination stops, a desired boarding time that is an exponentially distributed delay after the request time, whose mean is also given as an input parameter. A journey request may be served within a maximum admissible delay after the desired departure time, which will be given as an input parameter as well.
- **Minibuses.** Minibuses are scheduled periodically, as new requests enter the system. Their capacity in terms of maximum number of passengers is given as input, but for practical reasons will not exceed  $C_{\max} = 24$  seats. We will consider the time to board and disembark a passenger is constant and is given as an input parameter.
- **Service Network.** The service network comprises a specified number of bus stops and the travel distances between each one of them and its neighbours, expressed in minutes. We will use a directed graph representation to model this, as shown in the example illustrated in Figure 1. The total number of stops and the graph representation of the distances

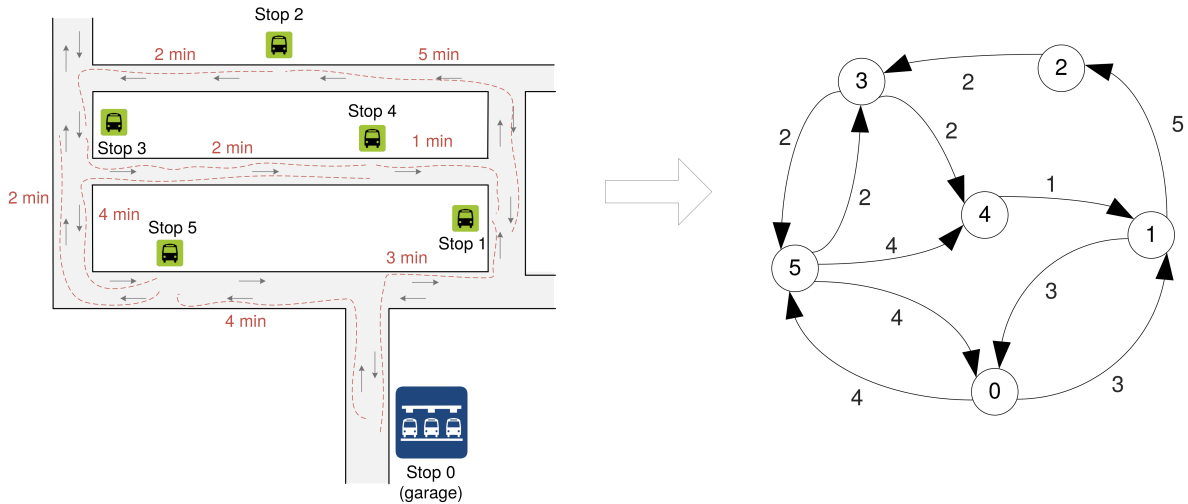


Figure 1: Simple example of a service network graph model. The vertices represent bus stop locations, the graph arcs the roads between two adjacent locations and the arc costs the duration (in minutes) required to travel between them.

between them (arc costs) will be provided as an input in matrix form. Note that by using directed graphs, we can model both one-way and two-way streets.

- Route Planning.** The decision to schedule a minibus can be triggered once a new request enters the system. However, the schedule can be delayed according to a maximum admissible delay that is given as a parameter, i.e. a user may be collected within that time interval starting from the desired departure time. Further, the route of a minibus that is already in service can be updated to accommodate a new request as long as this will not alter previously agreed departure times. For a planned journey, you must ensure that the minibus occupancy is not exceeded at any time. Based on a set of requests, you must compute the shortest route that picks up and drops off the largest possible number of passengers that intend to take similar journeys, i.e. the shortest paths from each start and destination pair overlap as much as possible.

There are two other important aspects here. First, there may not be a direct link between all stops that must be visited. Thus, when computing a path, you may have to keep track of the intermediary arcs between vertices that have no service demand and adjacent ones that do. Second, finding optimal routes that visit all nodes with active requests at minimum costs is required. How you implement this is entirely your choice. For instance you may choose to run shortest path algorithms such as Johnson–Diskstra, Floyd–Warshall, etc. and combine the solutions given by these into a single one or run these on sub-graphs that contains multiple start and destination pairs. The task is non-trivial, thus exploring different heuristics is also appropriate.

Note that when e.g. a small number of vertices need to be visited, it is often more efficient to travel multiple times through the same location even if the route previously serviced passengers who had placed requests there. Further, for small areas, even a brute-force approach that computes all the possible routes and selects the one with the small cost may be appropriate.

Thus when planning a service route, you are free to choose any existing algorithms, heuristics, or implement your own solution, but you are expected to document your choice and discuss its implication on the system’s performance in your written report.

From the description of the components above you should be able to identify the key requirements for the simulator you will develop. When implementing these, you are expected to include comments in your code that explain these requirements, as well as the design choices you made.

At simulation start time, all minibuses are considered to be at the garage (location 0) and there will be requests in the system. New requests can fire off at any time, as long as they do so with the rate specified in the input.

## 2.4 Command-line Arguments

The application you will develop is to be executed at the system console and should accept command-line arguments. It is mandatory that one argument is the name of the input script that describes your simulation. You can optionally add other parameters, e.g. for logging purposes or to allow disabling detailed output when repeating the simulations with different values of a certain input. Thus running your code at the console will be done as follows:

```
user:~$ ./simulator <input file name> [OPTIONS]
```

If you do add optional parameters, please document your code and display information about usage when the application is run without arguments.

The output of your simulations should be printed to the system standard output (stdout). The input and output formatting is specified next.

## 2.5 Input Formatting

Your input file will contain a set of global parameters, including minibus capacity, passenger boarding and disembarkation duration, mean journey request arrival rate, mean desired collection interval, and maximum admissible waiting delay. Then we define the number of minibuses, the number of bus stops, and the graph representation of the stop locations with the distances between them (road map), where the first location is considered to be the garage. Except for the road map, each object will be described in one line of input. The graph corresponding to the road map will be given in matrix form.

### Input Specification

The input parameters will be given in the input script according to the following specification:

```
busCapacity <unsigned int>
boardingTime <unsigned int>
requestRate <float>
pickupInterval <float>
maxDelay <unsigned int>
noBuses <unsigned int>
noStops <unsigned int>
map
0      <int> <int> ... <int>
<int> 0      <int> ... <int>
:      :      :      ... <: >
<int> <int> <int> ... 0
```

**stopTime** <float>

We will work with minibus capacities specified in terms of the maximum number of passengers. The time required to board and disembark a passenger is expressed in seconds. The user request rate is expressed as the average number of journey requests per hour, and the average duration between the time of request and the desired pick-up time (`pickupInterval`) is expressed in minutes. Since a user's desired departure time may not be accommodated, a practical system would indicate the earliest time available for collection, which may or may not be within the maximum delay a passenger could tolerate (`maxDelay`) expressed in minutes. If a request cannot be served within that delay, the event must be logged.

While the order of the parameters is not strict, the description of the road layout connecting the stops must follow the definition of the number of stops and is introduced by the **map** keyword. Except for the map description, all parameters should be given each on a single line.

The cost matrix that specifies the duration in minutes between each stop has the (0,0) element representing the bus garage, while elements  $(i,j)$  are zero when  $i = j$  and we use -1 to indicate there is no direct link between two locations. Remember we will be using directed graphs, i.e. there may only be a one way link between two vertices.

In addition to the above, it is essential that you specify when the simulation should terminate. This will be given in hours. Appropriate conversion to minutes/seconds may be required.

You may use the **#** character to comment out a line in the input file and you should ignore blank lines. An example is given below.

## Example

```
# capacity of a minibus in terms of number of passengers
busCapacity 12
# passenger boarding/disembarkation duration (in seconds)
boardingTime 10
# Average number of journey requests per hour
requestRate 30.0
# Average time between request and desired pick-up (in minutes)
pickupInterval 30.0
# Maximum time a user can wait beyond the desired pick-up time (in minutes)
maxDelay 15
# Number of minibuses
noBuses 5
# Number of bus stops
noStops 6
# Road layout and distances (in minutes) between bus stops
map
  0   3  -1  -1  -1   4
  3   0   5  -1  -1  -1
 -1  -1   0   2  -1  -1
 -1  -1  -1   0   2   2
 -1   1  -1  -1   0  -1
  4  -1  -1   2   4   0
# Simulation duration (in hours)
stopTime 24
```

## 2.6 Output Formatting

Your simulator should output information about events in a human readable format. On the other hand the output must be easily parsable, such that specific information can be extracted e.g. for visualisation purposes. Each event will be output on a single line in the following format: `<time> -> <event> <details>` We will output time in the format **days:hours:minutes:seconds**. Since some of the input parameters (e.g. distances between locations, passenger boarding and disembarkation times) are given in minutes or seconds, you should make the appropriate conversions and work with second granularity. You must account for the following events:

- new journey request scheduled / can not be accommodated
- minibus left / arrived at location
- passenger boarded / disembarked
- minibus occupancy changed

### Output Specification

The output formatting for the expected events is given below. Note that parts of the functionality you implement will be automatically tested. Thus strictly abiding to the output format given below is essential.

```
<time> -> new request placed from stop <unsigned int> to stop <unsigned int> for departure at <time> scheduled
    for <time>
<time> -> new request placed from stop <unsigned int> to stop <unsigned int> for departure at <time> cannot
    be accommodated
<time> -> minibus <unsigned int> arrived at stop <unsigned int>
<time> -> minibus <unsigned int> left stop <unsigned int>
<time> -> minibus <unsigned int> boarded passenger at stop <unsigned int>
<time> -> minibus <unsigned int> disembarked passenger at stop <unsigned int>
<time> -> minibus <unsigned int> occupancy became <unsigned int>
```

### Example

An example of valid output is shown next.

```
00:01:20:00 -> new request placed from stop 2 to stop 10 for departure at 00:01:32:00 scheduled
    for 00:01:33:00
00:01:25:10 -> new request placed from stop 3 to stop 8 for departure at 00:01:34:00 scheduled
    for 00:01:36:00
00:01:28:00 -> minibus 1 left stop 1
00:01:33:00 -> minibus 1 arrived at stop 2
00:01:33:10 -> minibus 1 boarded passenger at stop 2
00:01:33:10 -> minibus 1 occupancy became 4
00:01:33:10 -> minibus 1 left stop 2
00:01:33:30 -> minibus 2 arrived at stop 5
00:01:33:40 -> minibus 2 disembarked passenger at stop 5
00:01:33:40 -> minibus 2 occupancy became 0
00:01:35:10 -> minibus 1 arrived at stop 3
00:01:35:30 -> new request placed from stop 0 to stop 5 for departure at 00:01:37:00 cannot be accommodated
```



```
00:01:36:00 -> minibus 1 boarded passenger at stop 3
00:01:36:10 -> minibus 1 occupancy became 5
00:01:36:10 -> minibus 1 left stop 3
...
```

Take the time to analyse your output carefully. While some output can appear syntactically correct, it may reveal some errors in your implementation approach.

## 2.7 Statistics Analysis

In addition to reporting the events timeline, your application should return summary statistics at the end of the simulation time. The analyses you are expected to perform are detailed next.

### Average Trip Duration

Your simulator should report the average duration of trips performed by a minibus over the duration of your simulation. This will provide an indication of how the maximum time a user would be willing to wait beyond the desired departure time impacts route lengths and how well algorithms solving the shortest path computation task work. You should report the average trip duration in the following format:

```
average trip duration <average duration (minutes:seconds)>
```

### Trip Efficiency

To be able to assess how efficient such on demand public transport operation is, it will be appropriate to compute the number of passengers transported per unit of travel time. For this purpose you will need to trace for each route the duration, as well as the minibus occupancy between boarding and disembarkation events. We will assume that all buses leave the origin with zero occupancy. The average efficiency will be displayed as a floating point number, in the following format:

```
trip efficiency <average occupancy per time unit (passengers aboard per minute)>
```

These statistics may also prove useful for increasing the size of the minibus fleet.

### Percentage of Missed Requests

Similarly, to gain insight into the ability of the minibus fleet to accommodate user requests, we will count the number of instances a new request cannot be served within the maximum admissible delay of the desired departure time. This will also provide an indication of customer satisfaction and will be reported as the average number of missed requests over the total number of requests during the simulation, as follows:

```
percentage of missed requests <fraction of missed over total requests>
```

## Average Waiting Time

A minibus with passengers already travelling may often be required to wait at a stop for a new user to arrive and board, which is another aspect that impacts customer satisfaction. Thus we will report the average duration a passenger waits on board (except for boarding and disembarkation of others) during a journey. The output will be presented in seconds as follows:

```
average passenger waiting time
    <average duration waiting at stops during a trip> seconds
```

## Average Trip Deviation

Finally, we are interested in monitoring how much a journey deviates from the shortest path from a passenger's departure point to their intended destination. For this purpose we will compute for each passenger the difference between the total time of the shortest path and the actual duration that passenger is actually spending on a bus. It is straightforward to observe that there are several input parameters and design choices that impact this statistic. We will use the following semantic:

```
average trip deviation
    <sum of differences between shortest and actual trips over trip count>
```

You should delimit the start and end of the statistics with a '---' sequence.

## Example

An example of valid summary statistics is shown below.

```
---
average trip duration 14:30
trip efficiency 7.20
percentage of missed requests 0.10
average passenger waiting time 30 seconds
average trip deviation 1.60
---
```

## 2.8 Experimentation

To acquire a better understanding of the impact of certain parameters on the metrics of interest, your simulator should allow experimenting with different values of the maximum admissible pick-up delay and the number of minibuses. For that, in the input file you should be able to specify a set of values instead of a single one. This set will be defined using the **experiment** keyword, followed by the values. For example, the line

```
maxDelay experiment 10 20 30
```

instructs the simulator to run three simulations with the same parameters and topology, but varying the maximum allowed delay for pick-up. Similarly, we can try experimenting with different fleet sizes. For example, in the following input script

```
noBuses experiment 10 15 20
```

For a road map given as input, it would be appropriate to attempt to find the optimal value of one parameter (e.g. pick-up delay or number of minibuses) that maximises trip efficiency or minimises the average waiting time.

Lastly, you should also allow experimenting with multiple parameters at the same time. When running a set of such experiments, you **should** disable the output of detailed information about the events and instead only display summary statistics. **To delimit the different experiments, precede the output of each with the following heading:**

```
Experiment #<experiment no.>: <param1> <param1-value> <param2> <param2-value> ...  
---
```

For example, a fragment of valid output would look like this:

```
...  
Experiment #4:  maxDelay 10 noBuses 15  
---  
average trip duration 14:30  
trip efficiency 7.20  
percentage of missed requests 0.10  
average passenger waiting time 30 seconds  
average trip deviation 1.60  
---  
Experiment #5:  maxDelay 10 noBuses 20  
---  
average trip duration 12:00  
...
```

## 2.9 Visualisation

As you have noticed, one requirement is to produce output in a format that is easy to parse. This is particularly important when experimenting with different values of a parameter, as you may wish to use this output to plot various metrics. For instance you could plot the trip efficiency as a function of pick-up delay. Similarly, you may also examine the time evolution of minibus occupancies.

To plot your results you can load your output into spreadsheets (e.g. OpenOffice Calc, Google Spreadsheet, Microsoft Excel, etc.) or you can use specialist plotting programs such as GNUplot, R or matplotlib.

## 2.10 Testing

To demonstrate that your application has been thoroughly tested, you will be required to submit a number of test inputs of your own, both valid and invalid. You should add comments in the scripts to explain what you are testing and in your report you should explain the purpose of each test and the results expected.

**NB:** Parts of the functionality of your code will be subject to automated testing with different previously seen and unseen inputs. Thus, even if your output may be semantically correct, it will be regarded as invalid if not abiding to the specified format. Also, refrain from prompting

the user for interactive input; your programme should have enough information to run simulations after parsing an input script. Likewise, if you decide to add additional arguments to the command line, these should be strictly optional and documented.

### 3 Frequently Asked Questions

- *Why am I expected to develop my application in C? Can I use Java/Python/Visual Basic instead?*

No. Part of the challenge of this practical is to improve your skills with a procedural language that is widely used as a system programming language. This is also something you can expect when taking a job as a software developer in a company that has clear incentives to use a particular language. C is currently ranked among the most popular programming languages according to the TIOBE index.<sup>1</sup>

- *Can I develop my application on my Windows/MAC/Linux laptop/desktop?*

Yes. Just make sure that it compiles and/or runs on DiCE as well.

- *Shouldn't we take into account different request arrival rates at different locations and changes in the rates triggered by some events (e.g. holidays)?*

Arguably. In this practical, however, we are analysing snapshots of limited durations and we assume people have on average a similar behaviour. We could potentially evaluate even more complex scenarios by using different inputs and further extending the functionality of the simulator. Nevertheless, the application build according to our requirements will still provide important insights into the system's performance.

- *What is the proposed maximum number of bus stops?*

It is foreseeable that the number of bus stops may limit e.g. the possibility of performing exhaustive search to find an optimal route. For this practical we will assume the number of locations is specified as an `uint16_t` value, i.e. the maximum number is 65,535.

- *What about ticket costs, fuel consumption, and refilling times?*

Although these are some of the many practical aspects one could consider, in your simulations you are not required to include such constraints.

---

<sup>1</sup><http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

# Part 1

## Computer Science Large Practical 2015–2016

Dr Paul Patras  
School of Informatics

Issued on: Friday 25<sup>th</sup> September, 2015.

Deadline: Thursday 22<sup>nd</sup> October, 2015 at 16:00.

### 1 Introduction

This part of the CSLP is optional and it is primarily intended to allow you to receive useful feedback on your progress, such that you can work out whether you are on track to complete the second part in time.

Any work you choose to submit should be your own, although you may make use of any part of any publicly-available source code that you find useful. If you do re-use code available on online repositories, clearly state which parts of the code are not your own. Code sharing among colleagues is not allowed.

### 2 Description

Part one of the CSLP requires you to submit your C simulator as a preview version of the software you are expected to deliver for Part 2 of the CSLP assignment. It is understood that at this stage the functionality of your application will be incomplete and that it will not have been comprehensively tested. However, this preview version will prove that you have started the work on the practical and became familiar with the concept of discrete-event stochastic simulation. You will only be assessed on the execution of a single simulation. The experimentation part will not be tested.

In this instance you will not be provided with exhaustive qualitative feedback as this would arrive too late to be useful to you. However your application will be tested and a you will receive a brief report of how well you are doing at this point. It is also recommended that you included a text file with your submission, where you ask specifically on which aspects of your project you wish to receive feedback on.

### 3 What to Submit

You are to submit the directory which contains your development project. This should be a working C application that can be compiled and executed. A simple README file documenting how your application is compiled and run is sufficient.

At this point you are not required to submit any further documentation. A written report is neither required.

### 4 How to Submit

First, get your code on the School of Informatics DiCE computer system, copying it from your own laptop as necessary. If your project is in a folder called `Simulator` then you should submit it for the cslp part 1 using the command:

```
submit cslp 1 simulator
```

### 5 Deadline

The deadline for this practical exercise is

**Thursday 22<sup>nd</sup> October, 2015 at 16:00**

### 6 Frequently Asked Questions and Clarification

- *I haven't managed to implement parts of application even those not associated with experimentation. Can I still submit what I do have?*

Yes. I will do my best to provide some feedback on the parts which are working. In addition there is some qualitative feedback which may be worthwhile to you.

- *I want to explain which parts of my application are not finished. Can I submit documentation along with Part 1?*

Yes, if you wish to then you can do this. If doing so then please also write this in your README file.

- *If I don't complete Part 1 before the deadline, can I modify that part in the time before the deadline for Part 2?*

Absolutely. You could even discard everything you have done for Part 1 and start again if you wish, though that would generally be inadvisable.

# Part 2

## Computer Science Large Practical 2015–2016

Dr Paul Patras  
School of Informatics

Issued on: Friday 25<sup>th</sup> September, 2015.

**Updated on: Tuesday 27<sup>th</sup> October, 2015.**

Deadline: Monday 21<sup>st</sup> December, 2014 at 16:00.

### 1 Introduction

This is an assessed practical exercise. It carries 100% of the mark for the CSLP. It represents a total of roughly 100 hours of work. The work which you submit for assessment should be your own although you may make use of any part of any publicly-available source code which you find useful. You should mark clearly the parts of the code that are not your own. This means you are re-using code, which is a good thing and distinguishes it from plagiarism, which is a bad thing.

### 2 Description

Part two of the CSLP requires you to submit the finished version of your C application. Your application should be able to simulate the existing example input scripts available on the course web page, **and** it should be able to simulate previously-unseen input scripts you have generated yourself. All parts will be tested, including the experimentation part.

You will not be evaluated on the code alone. It is essential that you prepare a **written report** explaining your findings from implementing the simulator and the insights gained into the performance of the simulated system. These should be accompanied by plots you produced where appropriate.

— ◇ —

Below is a summary of the key functionality expected:

- It should be possible to read and parse an input script of the correct format.
- Your application should produce a valid sequence of events resulting from correct simulation of the input script.

- Your application should validate the input and produce warnings and/or errors as appropriate.
- After all simulation events have been output, the summary statistics for the simulation should also be output.
- Experiments should function as described. You should allow disabling detailed output for experimentation.
- You should have a test suite with some sample bus stop locations and road map descriptions which are non-trivially different from each other.
- You should document all your design choices, findings, experiments and results in the written report.

### 3 Additional Credit

The requirements listed in the section above illustrate the core functionality which is required from your application. A well engineered solution which addresses all of the above requirements should expect to attract a very good or excellent mark. Additional credit will be awarded for additional useful features which are *not* on the above list. Thus, if you have time remaining before the submission deadline and you have already met all the requirements listed above, then you can attract additional marks by being creative, conceiving of new features which could be added to the application, and implementing these.

If you have added additional features to your implementation in order to attract extra credit then you should be sure to document these. The purpose of this part of the practical exercise is to allow you to exercise your own creativity and deliver a solution which is uniquely your own. Note that the highest mark you can achieve is still 100, but this additional credit may help if you lose points due to some error found in the way you implemented standard requirements.

Examples of features you might like to consider adding could be the following:

- Allowing other types of distributions for the request arrival rates;
- Allowing minibuses of different capacities;
- Allowing users to place journey requests for more than one passenger.

If you include any additional features, first make sure that the simulator will still operate given basic input. You can either design the extra input to be compatible with the existing input, or you can provide a command-line flag to switch between basic and extended functionality, while clearly documenting this in your README file.

### 4 Assessment

Part two of the CSLP is worth 100% of the mark for the course. Your mark will be expressed as a percentage given as an integer between 0 and 100. Thus it is not possible to score more than 100 and it is not possible to score less than zero.

### 5 Assessment Criteria

This practical exercise will be assessed in terms of the completeness, correctness and efficiency of the simulator, the quality of the C code using features of the language well, as well as the design choices, findings, results and conclusions presented in the written report.



- Your application should implement all the required functionality. This includes input parsing and validation, correct simulation and correct output, generating summary statistics, experiment implementation, code documentation, and testing (including sample input scripts).
  - Not implementing all the requirements will lose marks.
- Your application should be appropriately structured and easy for an experienced programmer to understand.
  - Writing idiomatic code will gain marks.
- Additionally, all else being equal, an application whose code contains examples of poor programming style (such as unused variables, dead code, blocks of commented-out code etc) should expect to attract fewer marks than an application which does not have these problems.
  - Poor programming style will lose marks.
- Efficiency is important in simulators which may need to deal with millions of simulated events. The main loop of the simulation should be carefully coded to avoid unnecessary object allocations and other instructions.
  - A more efficient simulator will get more marks.
- Your written report should be well written and organised, and should provide a complete summary of your work, the insights gained, plots generated and key findings.
  - A sloppy report will lose marks.

Additionally, if your simulator is fully functional and you manage to implement additional features that you document, you will get more marks.

## 6 What to Submit

You are to submit the directory which contains your C development and the written report. Your work will be assessed based on your report and by compiling and executing your application, so you must ensure that all source code files needed to compile and run your application are submitted.

Your directory should contain a folder called `doc` where you should put a **PDF version** of your written report. You should also include a `README` file in your directory, where you should provide:

- instructions for running your application:
  - include information about the development platform (DiCE, Windows, Ubuntu, OS X) and compiler, but remember that your application should compile and run on DiCE;
  - include any information about any parts of your application which are not finished or have non-obvious functionality, and
  - include notes on any additional features of your application of which you are duly proud;
- any input scripts you have created for testing purposes (please indicate the `README` file the location of such scripts).

Since part of the functionality of your simulator will be automatically tested, also include a Makefile with your submission. Running ‘make’ in the root directory of the submitted project should produce a running application.

You would additionally be well-advised to include some text files storing the output of your simulator operating over your test files.

## 7 How to Submit

First, get your code on the School of Informatics DiCE computer system, copying it from your own laptop as necessary. If your project is in a folder called `Simulator` then you should submit it for the cslp part 2 using the command:

```
submit cslp 2 simulator
```

## 8 Deadline

The deadline for this practical exercise is

**Thursday 17<sup>th</sup> December, 2015 at 16:00**

## 9 Marking

Your submitted coursework will be marked within 14 working days of submission. (This is the period of time approved by the School of Informatics for large practical classes.) For 2015/2016 this means Friday 15<sup>th</sup> of January, 2016.

We will try to meet this deadline but – for good reasons – students sometimes get deadline extensions meaning that their submitted coursework is not available to mark starting at Thursday 17<sup>th</sup> December, 2015 at 16:00. This can delay the return of marked coursework.

## 10 Frequently Asked Questions and Clarification

- *Should I put comments in my code?*
  - Yes. All else being equal, a submission with comments will attract more marks than one without comments.
- *How long should be my written report?*
  - There is no minimum page limit. Your report should demonstrate you are familiar with discrete-event stochastic simulations, you can interpret the output, understand the system’s performance and you can present your results in a clear and concise manner.
- *How much does the written report weight?*
  - The written report accounts for 25% of the final assessment.
- *How will this practical be marked?*
  - Submissions for the Computer Science Large Practical will be evaluated using the following process:
    1. Accompanying documentation is read for instructions on how to use the application.

- Submissions with insufficient documentation will lose marks here.
- 2. The C project is compiled and inspected for errors and warnings.
  - Submissions with errors will lose marks.
- 3. The application is run on previously-seen sample input scripts.
  - Submissions which fail to execute will lose marks here.  
(Refer to Section 2.5 in the coursework description for input specification).
  - Submissions which produce incorrect output will lose marks here.  
(Refer to Section 2.6 in the coursework description for output specification).
- 4. The application is run on previously-unseen sample input scripts.
  - Submissions which fail to execute will lose marks here (as above).
  - Submissions which produce incorrect output will lose marks here (as above).
- 5. The submitted simulation scripts are inspected as evidence of developer testing.
  - Submissions which have had insufficient testing will lose marks here.
- 6. Other additional features of the application will be explored.
  - Submissions with useful additional features will gain marks here.
- 7. The C source code will be inspected for good programming style.
  - Submissions with insufficient structure will lose marks here.
  - Submissions which do not use language features well will lose marks here.
  - Submissions with too few comments will lose marks here.
  - Submissions with blocks of commented-out code will lose marks here.
- 8. Additional features implemented will be verified.
  - Submission will be inspected for evidence of additional features.
  - Implementing additional features can attract a maximum of 10 marks.