"Hardly Normal"

Curtin University, Department of Computing
Assignment, Semester 2, 2015

CRICOS Number 00301J
September 3, 2015

Preamble

In practical 8 you have already implemented a generic DSALinkedList, DSAQueue and DSAStack. You have also had a chance to use stacks and queues in a structured manner, in the form of the Infix to Postfix calculator scenario. In this assignment, you will be making use of this knowledge to implement a system that is somewhat more open ended. In this system, you will be deciding (for the most part) how to structure the program, how to use the abstract data types and your program will need to decide on the flow of objects in and out of them. Feel free to re-use the generic ADTs from your practicals.

The start of this assignment specification document does look somewhat daunting and confusing. I recommend that you highlight and/or take notes. Also note that you are not expected to build the optimal solution to this problem (in the general case that's still a research topic!). Jump to Section 5 for a breakdown of what your program is expected to do for each set of marks. Note that, as you can see from Section 5, you can get most of the marks for doing what is arguably the easiest part of the specification!

Requirements for passing the unit

Please note: As specified in the unit outline, it is necessary to have attempted the assignment in order to pass the unit. As a guide, you should score at least 15% to be considered to have attempted this assignment. We have given you the exact mark breakdown in Section 5. Note that the marks indicated in this section represent maximums, achieved only if you completely satisfy the requirements of the relevant section.

Plagiarism is a serious offence. This assignment has many correct solutions so plagiarism will be easy for us to detect (And we will). For information about plagiarism, please refer to
`http://academicintegrity.curtin.edu.au`

Late Submission

I know for the vast majority of you, I don't need to tell you what is in this section. But every semester, someone always tries to see what they can get away with here. For the benefit of fairness to the vast majority of you who do the right thing, we need to make the rules surrounding late submission clear

As specified in the unit outline, you must submit the assignment on the due date. **Acceptance of late submissions is not automatic and will require supporting documentation proving that the late submission was due to unexpected factors outside your control.** See the unit outline for details as to the procedure for requesting that an assessment be accepted after the due date.

Note that external pre-scheduled commitments including, but not limited to, work, travel, scheduled medical, sporting, family or community engagements are not considered unexpected factors outside your control. If you know you have, or are likely to have, such engagements and that they may affect your ability to complete the assignment, you will be expected to have planned your work accordingly. This may mean that you need to start and/or complete your assignment early to make sure that you are able to hand it in on time.

Also note that IT related issues are almost never a valid excuse. These include computer crashes, hard disk corruption, viruses, losing computers/storage media, networks going down (even if it is Curtin's network - outages of the entire Curtin network of more than 24 hours may be considered depending on circumstances) or the like. As IT professionals in training, you are expected to have suitable backups and alternative ways of getting your assignment completed in the event that any IT problems are encountered. You are also expected to submit your assignment ahead of time to allow for unforeseen issues.

In the event that you submit your assignment late **and are** deemed to have a valid excuse, you will be penalised 10% (that is, 10% out of 100%, not out of what you would have received) per calendar day that you are late, up to a maximum of seven (7) calendar days. Any work submitted after this time will not be marked and you will automatically fail the unit.

Note that the requirements for passing this unit are applied after penalties. An assignment normally scoring 15% that is submitted one day late, even with a valid excuse, will risk not satisfying the requirements for passing this unit.

Clarifications

This assignment specification may be clarified and/or amended at any time. Such clarifications and amendments will be announced in the lecture and on the unit's Blackboard page (not necessarily at the same time and not necessarily in that order). These clarifications and amendments form part of the assignment specification and may include things that affect mark allocations or specific tasks. It is your responsibility to be aware of these, either by attending the lectures, watching the iLecture and/or monitoring the Blackboard page.

1 Introduction

A large multinational electronics retailer is responsible for the logistics surrounding the bulk-purchase, storage and distribution of products to over 800 franchisee storefronts around the world. In order to properly manage their Distribution Centres, they need software that will allow them to keep track of their various products, determine where new stock deliveries should be sent, figure out where various carton deliveries can be found and so-on.

You are responsible for developing the software that will allow them to track these logistics for all Distribution Centres. Your program will accept two files as input: A file describing the state of the Distribution Centre (the DC description file) and a file describing the task that your program should perform (the task file). Your program should then output to standard output information that answers that task, as described below. For tasks other than searching for stock, your program should also write a new DC description file that describes the state of the Distribution Centre after performing the given task. More details about the inputs and outputs of your program are described in Sections 4 and 6.

The Distribution Centre is based around Cartons of stock. In this document, we will refer to a "Carton" as a unit of this stock. Each Carton holds only one model of a product. Different Cartons can contain different models, each with different specifications. The Cartons are stored in a Distribution Centre. Each Distribution Centre has a variety of properties with respect to layout and capacity. Within these Distribution Centres, several tasks need to be performed.

2 Carton Deliveries

All Cartons have the following information:

- Consignment note of the Carton. This is a positive integer less than 1024 that is guaranteed to be unique out of all the Cartons in a given Distribution Centre. There is no significance to the value of this number apart from the fact that it is a unique identifier.
- Type of product in the Carton. This is an arbitrary ASCII string that will be common among all products of the same model. Examples include, but are not limited to, "LemonBook Air", "Stone Smart Watch", "Samesong SUHD5500", "Hawaii Accent P1" and so-on. You will not know ahead of time what the different models will be, nor how many different models there will be.
- Wholesaler name. This is an arbitrary ASCII string that will be common among all Cartons of products from a given wholesaler. Note that any one wholesaler may provide more than one model of product. Examples include, but are not limited to, "Lemon Computers", "Digitorn", "Worsethan Digital" and so-on. As for models, you will not know ahead of time the names or number of wholesalers.
- Warranty date. This will be input and output to and from your program as an ASCII string in the format yyyy-MM-dd. For example, the 13th of June 2013 will be represented as 2013-06-13. Some products have lifetime warranties; these will input and output as an expiry date of all zeros (0000-00-00). It is up to you to select an appropriate representation for this lifetime warranty period. Note that you will need to do some arithmetic with this date later on, therefore your program will need to internally represent this as something more informative than just a string.

3 Distribution Centres (DC)

Cartons are stored in Distribution Centres. Each Distribution Centre has several stock rooms, indexed for zero to the number or stock rooms minus 1, Each stock room has a fixed capacity. Any Carton may be stored in any stock room . . . although you should be strategic about mixing (or conversely, avoiding the mixing of) different types of Carton in different stock rooms. Each stock room will have one of three different layouts. These layout are layouts are also fixed; a stock room is **never** remodelled to another layout. The spots in each stock room are indexed from zero to capacity minus one. The types are:

- Dead-end. In such a layout, only the Carton most recently added can be removed. Distribution Centres tend to have a lot of dead-end stock rooms but you don't want to put old products here if they're likely to get buried else their manufacturer support period could expire. Of course, if all the space you have left is in a dead-end, you don't really have a choice. Dead-end storage banks are filled from the lowest index, thus the only Carton that is accessible is the one at the highest occupied index. There should be no gaps between Cartons in a dead-end stock room.

- Rolling corridor. In such a layout, only the Carton least recently added can be removed. This is good for Products with short warranty periods as you'll tend to remove the oldest (and, in many cases, the earliest to lose manufacturer support) Cartons first, assuming they entered in such an order. There will usually be less rolling corridor layout space than dead-end space. Rolling corridor stock rooms grow towards higher indexes like dead-end storage banks but only the 0th indexed Carton can be removed at any time. When a Carton is removed from the 0th index, the corridor "rolls" as the remaining Cartons move down one index, with the Carton previously at index 1 moving to index 0, index 2 moving to index 1 and so-on. Again, there should be no gaps between Cartons in a rolling corridor stock room.
- Yard. In such a layout, any Carton may be moved at any time. While the most flexible, there tends to be the least amount of yard space as compared to dead-end or rolling corridor space. Unlike either of the two previous stock rooms, Cartons can be added and removed arbitrarily from a yard and there may be gaps between Cartons.

This is looking a lot like a Stack, a Queue and a general Array, right? An example of a very small Distribution Centre with each of these three layouts is shown in Figure 1. Wholesaler names have been shortened for clarity. Note that the Distribution Centres can get quite a bit larger than this, with many different stock rooms. Your program should be able to handle at least a few thousand stock rooms, each with hundreds of spots for Cartons.

Each Distribution Centre may have a different number of each stock room. Each stock room in a given Distribution Centre may have a different capacity. Cartons can move from place to place within the Distribution Centre as long as there is space. For example, to get a Carton from the middle of a dead end stock room, you can move all the cartons in front of it into another stock room that has space.

Index within Stock Room	0	1	2	3	4	5	6	7	8	9
	Stock Room **0** (type: dead-end, capacity: 10)									
details	1014 HDMI Cable Digitorn 0000-00-00	793 HDMI Cable Rick Sith 0000-00-00	403 Aux. Cord Rick Sith 0000-00-00							
In or Out			Out	In						
	Stock Room **1** (type: rolling corridor, capacity: 8)									
details	390 Ink Epsilon 2015-06-10	806 Ink Lovecraft 2015-08-08	195 Ink Epsilon 2015-08-20	481 Ink Lovecraft 2016-01-20	247 Ink Lovecraft 2016-01-20					
In or Out	Out					In				
	Stock Room **2** (type: yard, capacity: 3)									
details	975 Laptop Lenubu 2015-04-01		377 Laptop Samesong 2015-04-02							
In or Out	Out	In	Out							

Figure 1: An example of a small Distribution Centre, and the Cartons that are in it. The positions where new Cartons could be added and existing Cartons are marked with "In" and "Out" respectively.

3.1 DC Description File

The DC description file is a colon separated, ASCII, Unix-linebreak file with two distinct sections, one after the other. The sections are separated by a line with a single '%' character. The first section describes the geometry of the Distribution Centre and gives the consignment numbers of the Cartons within it. The second section describes each Carton in the Distribution Centre. An example of a description file for the Distribution Centre shown in Figure 1 is shown here.

```
# DC geometry section

D:1014:793:403:::::::
R:390:806:195:481:247:::
Y:875::377

%

# Carton description section

975:2015-04-01:Laptop:Lenubu
390:2015-06-10:Ink:Epsilon
195:2015-08-20:Ink:Epsilon
806:2015-08-08:Ink:Lovecraft
377:2015-04-02:Laptop:Samesong
481:2016-01-20:Ink:Lovecraft
403:0000-00-00:Aux. Cord:Rick Sith
```

```
793:0000-00-00:HDMI Cable:Rick Sith
247:2016-01-20:Ink:Lovecraft
1014:0000-00-00:HDMI Cable:Digitorn

# End of file
```

Your program should ignore all blank lines and all lines that start with '#'. Your program may optionally output such lines if you like but they are not required. The example files that we will provide you with may contain lines starting with '#' as comments and blank lines as spacers, to help you to understand what's happening. These are for your benefit, not that of your program!

Your program should also detect if something is wrong with the file. For example, a consignment note that is mentioned in the first part of the file that is missing from the second or vice versa. In these cases your program should output a sensible error message and quit.

3.1.1 DC Geometry

Each row in this section represents a stock room. The index of the stock room, starting from zero, is in the order in which these rows appear. In other words, the first row represents the 0th stock room.

The first element in each row is a single letter, one of 'D', 'R' or 'Y', to indicate that the room is a dead-end, rolling corridor or yard respectively. Stock rooms may appear in any order and stock rooms of the same type are not necessarily grouped together.

The subsequent elements in each row will either be empty (ie. consist of colons with nothing between them – "::"), or will consist of the consignment note of the Carton in that spot. The spots are zero-indexed.

3.1.2 Carton Description

Each row in this section represents a Carton. It contains the information listed in Section 2, in the order they appear in that section, separated by colons. The dates will appear in the form yyyy-MM-dd, zero-padded. For example, the 20th of January 2016 will appear as 2016-01-20. These rows will appear in no particular order. In particular, they are not guaranteed to appear in consignment note order or the same order as in the DC geometry description.

4 Tasks

The software that you will write will need to manage the performance of the following tasks, given the description of a Distribution Centre and the Cartons currently in it.

4.1 Receiving a Carton Delivery

Figure out a sensible place (stock room and, for a yard type room, the index) to add a new Carton. Alternatively, the program should report that the Distribution Centre is full. "Sensible" in this context requires you to have a technique for assigning Cartons to spots keeping in mind the issues highlighted above. You don't need to find the optimal solution - that's largely still a research problem. But you do need to have shown some thought here and justified your resulting algorithm in the report. Your algorithm will need to be able to handle any situation. It is expected that your algorithm will be sufficiently complex that you will need at least half a page of text, plus at least one diagram, to properly justify it.

The task file for adding a Carton will be a single-row Colon Separated Value file (ie. one line). The first element will be 'A' (for "Add"), followed by the four pieces of data described in Section 2. For example:

```
A:235:0000-00-00:HDMI Cable:Digitorn
```

Your program should print to standard output the index of the stock room in which the Carton should be added as an integer, a colon and then the index of the spot in the stock room in which the Carton should be placed. For a dead-end or rolling corridor, this should be the next available space. For a yard this can be any empty spot. This should all be on the one line. Alternatively, your program should output "FULL" if the Distribution Centre is full.

Your program should also write a DC description file, in the same format as the input, with a filename that matches that of the input DC description file appended with "-output". This file should describe the state of the Distribution Centre after the addition of this Carton. If the Distribution Centre is full, this file should describe the same Distribution Centre as the input file.

As an example, a program might respond to the aforementioned query, applied to the Distribution Centre described in Figure 1, by deciding to place the HDMI Cable into the Stock room 0 dead end on the premise that this item has a lifetime warranty so it doesn't matter if it gets buried. This would yield the output:

```
0:3
```

Plus the file:

```
D:1014:793:403:235::::::
R:390:806:195:481:247:::
Y:875::377

%

975:2015-04-01:Laptop:Lenubu
390:2015-06-10:Ink:Epsilon
195:2015-08-20:Ink:Epsilon
806:2015-08-08:Ink:Lovecraft
377:2015-04-02:Laptop:Samesong
481:2016-01-20:Ink:Lovecraft
403:0000-00-00:Aux. Cord:Rick Sith
793:0000-00-00:HDMI Cable:Rick Sith
247:2016-01-20:Ink:Lovecraft
1014:0000-00-00:HDMI Cable:Digitorn
235:0000-00-00:HDMI Cable:Digitorn
```

Note that there are many possible algorithms. For example, there might be a plausible case to put the incoming HDMI Cables in the remaining space of stock room 2.

4.2 Shipping Stock of a Given Product

Find the best next Carton to remove of a given Product and the number of movements you'll need to perform (if any) to get it out (or if it's impossible because you don't have the space). You will need to strike a balance between the easiest Carton to get of that Product, the Carton that will cease manufacturer support soonest but might be a bit hard to get to (for items that have a limited warranty period) and the Carton that might be causing the most inconvenience (because it's blocking something else). Again, you don't need to find the optimal solution - that's largely still a research problem. But you do need to have shown some thought here and justified your resulting algorithm in the report. It is expected that your algorithm will be sufficiently complex that you will need at least half a page of text, plus a diagram, to properly justify it.

The task file for removing a Carton will consist of a single-row Colon Separated Value file (ie. one line). The first element will be 'R' (for "Remove"). The second element will be a string that describes the type of the supply to be located.

```
R:Aux. Cord
```

Your program should print to standard output the index of the stock room from which the Carton should be taken as an integer, a colon, then the index of the spot in which the Carton to be removed will be found. For a dead-end this must be the highest occupied index. For a rolling corridor this must be zero. For a yard this will be the index in the yard of the Carton to be removed. This should all be on the one line.

On the next line, your program should print a line of colon-separated elements describing the Carton that should be removed, as listed in Section 2. For example, the aforementioned request, applied to the Distribution Centre in Figure 1, will yield the following:

```
0:2
```

If there are no Cartons of the required type your program should instead output "NOT FOUND". If the Carton of the required Product is found but it's "buried" within a dead-end or rolling corridor stock room and there isn't enough space to shuffle Cartons around to get it out (or your algorithm otherwise can't find a way to do so), your program should instead output "STUCK".

Your program should also write a DC description file, in the same format as the input, with a filename that matches that of the input DC description file appended with "-output". This file should describe the state of the Distribution Centre after the removal of this Carton. This should also reflect any shuffling of Cartons within the Distribution Centre that might be required. If your program was unsuccessful in either finding or extracting the required Carton, the output file should describe the same Distribution Centre as the input file. Again, for the aforementioned request and the Distribution Centre in Figure 1, your program should output a file containing:

```
D:1014:793::::::::
R:390:806:195:481:247:::
Y:875::377

%

975:2015-04-01:Laptop:Lenubu
390:2015-06-10:Ink:Epsilon
195:2015-08-20:Ink:Epsilon
806:2015-08-08:Ink:Lovecraft
377:2015-04-02:Laptop:Samesong
```

481:2016-01-20:Ink:Lovecraft
793:0000-00-00:HDMI Cable:Rick Sith
247:2016-01-20:Ink:Lovecraft
1014:0000-00-00:HDMI Cable:Digitorn

4.3 Searching for Cartons Given Particular Information

Your program should be able to search for Cartons given particular information. This should be done using a search tree of some sort and/or a sorting algorithm of some sort. At most only 1/4 of the marks will be allocated to any solutions that simply do a brute-force search through the Distribution Centre for the Cartons in question. You will need to explain and justify this algorithm and your organisation of your data to support this searching in your report. It is expected that your algorithm will be sufficiently complex that you will need at least a page of text, plus a diagram, to properly justify it. The task file for searching for one or more Cartons will consist of a single-row Colon Separated Value file (ie. one line). The first element will be 'S' (for "Search"). There will then be four elements, one for each of the pieces of information in Section 2. Note that some of these elements will be blank if we don't care about them. This task will return zero or more Cartons. For each Carton, your program should print to standard output, in colon-separated format, the index of the stock room in which it is located, its index within that stock room and then the four elements that describe the Carton as listed in Section 2. The Cartons should be returned in ascending order of stock room and then, for Cartons in the same stock room, ascending order of index within that room. For example, if we want to find all Cartons with "HDMI Cables" in them, the file will be:

S:::HDMI Cable:

This should yield the following on standard output for the Distribution Centre in Figure 1:

0:0:1014:0000-00-00:HDMI Cable:Digitorn
0:1:793:0000-00-00:HDMI Cable:Rick Sith

Note that any search on the warranty expiry date should also match all earlier expiry dates. For example, if the Distribution Centre contains a Carton of Ink with an expiry date of the 3rd of December 2015, it should be returned in response to the following task file.

S::2016-01-01:Ink:

This should yield the following on standard output for the Distribution Centre in Figure 1:

1:0:390:2015-06-10:Ink:Epsilon
1:1:806:2015-08-08:Ink:Lovecraft
1:2:195:2015-08-20:Ink:Epsilon

Of course, as a search task, the Distribution Centre does not change so you don't need to output the DC description file.

5 Marking

Managing such a system in an optimal manner, is actually still the topic of continued research. We certainly don't expect you to come up with an optimal system, after all this is just an assignment to give you practice dealing with data structures and algorithms. The marks we assign are so that we can measure how well you understand and implement these data structures and algorithms.

Please note: As specified in the unit outline, it is necessary to have attempted the assignment in order to pass the unit. As a guide, you should score at least 15% to be considered to have attempted this assignment. We have given you the exact mark breakdown in this section. Note that the marks indicated in this section represent maximums, achieved only if you completely satisfy the requirements of the relevant section.

5.1 Marking Breakdown

Marks will be assigned to the following implemented capabilities in your program (note that we mean you write ONE program that can do all of these things, not several programs that do one of these things each).

You may attempt to answer this assignment in any order you wish. The order shown below is merely a suggestion for a sequence that might be natural. Later in the semester, we will provide you with an example of an input file that will exercise each of these capabilities. Note that when we mark your program, we will be running it with a different input file (that we won't be giving you until we hand your marked assignment back!) that is similarly correct and similarly restricted to the capability noted. We encourage you to generate your own additional input files for testing.

5.1.1 Class Diagram: 5%

You will hand in a class diagram showing the final system that you actually implemented (or, if your implementation is incomplete, what you would have implemented). This diagram must be a valid PNG or JPEG file and named "classdiagram.[png|jpg]". Ensure that your diagram is readable!

5.1.2 Justifications of Add/Remove/Search Implementations: 20%

The justifications outlined in Section 4. Again, if you write a sensible algorithm and justification here, you can get at least some of these marks even if your program does not compile.

5.1.3 Read, Interpret and Write DC Description Files: 30%

Your system can read and interpret a distribution centre file, **store the relevant information into sensible data structures** and write it back out again. Note that this is simply implementing the functionality that would be expected if one were to assign a task of removing a carton type that does not exist. Your program should also be able to detect issues with the input file and exit with an appropriate error message.

5.1.4 Implementation of Adding: 15%

Your system can handle receiving in new cartons of stock to an existing distribution centre. This includes doing something sensible to select a stock room, keeping in mind the issues highlighted above.

5.1.5 Implementation of removal: 15%

Your system can handle removing a carton from an existing distribution centre. This includes figuring out the optimal carton to remove based on the combination of tradeoffs highlighted above.

5.1.6 Implementation of Search: 15%

Your system can efficiently search for cartons as described, using an efficient data structure and/or search algorithm.

5.2 General Guidelines

For each feature described above (apart from the class diagram of course!), 50% of the marks will be assigned to the feature actually working correctly and 50% of the marks will be assigned to correct use of data structures and coding style.

5.2.1 System Correctness

For the capability to be considered to be "working", it needs to parse the input file properly, perform the relevant task according to the rules outlined above and generate the correct files, according to the prescribed format and naming convention. Note that in particular, we will deduct marks if, for a given input file, your program "locks up" or does nothing for an extended period of time.

Also note that you will need to make sure that your program is "bulletproof". At no point should your program throw an exception to the command line. Catching "Exception" in your main class is also not acceptable - you need to actually know what you're catching and do something sensible about it - or print a sensible error message to standard error. If you receive an input that does not make sense, your program should print a short, relevant error message to standard error and then exit cleanly. We will not be expecting complex algorithmic error checking (for example, we will not expect you to detect when two Cartons come in with the same consignment note).

Examples of input lines that do not make sense include (but are not limited to):

- Lines that do not follow the prescribed format. For example, a line that says "Doggo".
- Invalid initial characters.
- Missing or extra elements.

Note that there are many possible correct solutions. Just because your program does not get the same exact output as your friend does not mean that it is not correct. Also note that because we are not expecting optimal solutions, we do understand that different programs may run out of capacity at slightly different times.

5.2.2 Correct Use of Data Structures and Coding Style

Following the coding guidelines includes an acceptable level of commenting - not excessive and not insufficient. Consult with your practical tutor and senior tutors if you are unsure about what an acceptable level of commenting entails. In other words, it is not enough to just have your program exhibit these capabilities (and exhibit them correctly!), they must also be implemented according to the coding guidelines outlined in the rest of the unit, must use the data structures in a correct manner. Of course, if you implement something that is unconventional but "clever" or unusual, and yet well structured, you will still get the marks for it!

Please note that you will receive NO marks in this section if your program provides the right answer but you've "hard coded" a solution and aren't actually implementing the data structures. This is more of an issue for the earlier parts of the problem mentioned above, where it is possible to get a correct output without actually implementing big chunks of the necessary infrastructure. For example, simply scanning the input file for the Cartons in question

In later parts of the assignment, this becomes less of an issue as we will be testing your program with inputs designed such that if you implemented the data structures correctly, things will work. If you hard code or only partially implement chunks of the system, your system will fail. Again, I stress, we're not looking for the "optimal" solution and we aren't looking for a system that can manage the distribution centre at its absolute maximum capacity - in the general case that's open research. We are; however, looking for you using the correct data structures and algorithms, in the correct manner, in the correct places.

6 Your Submission

You are required to submit your assignment by 5:00pm on the due date as indicated in the unit outline (on the day of your lecture). You will be required to submit **BOTH** an electronic copy as well as a printed copy of your assignment.

Paper submissions should be placed in the "Data Structures and Algorithms" pigeon hole at the entry to the Building 314 level 2 labs. Upload your electronic submission (cover page, class diagram and source code) electronically via Blackboard, under the Assessments section.

We require an electronic submission so that we can test your program. We also require a paper submission because you are required to sign the cover page in ink and because we know that computers go wrong and you don't want to be disadvantaged because the system ate your electronic version. If there is a discrepancy, we will consider your paper version to be authoritative so please make sure that what you submit on paper and what you submit electronically match up.

We will not consider you to have submitted your assignment until we have received both the paper and electronic versions. Do not leave it to the last minute to print your assignment in case there are problems getting access to the relevant printing facilities. The printers in the labs being broken is not an excuse for a late submission.

You **MUST** ensure that your program compiles and runs on the lab computers. Do **NOT** assume that just because it runs on your computer at home, that it will run on the lab computers. Submissions that do not compile and run on the lab computers will receive a mark of **ZERO** for the program (which means your only marks will come from the report). You should already know how to remotely access the lab computers; living far from campus is not an excuse to have not done this.

Your program will be compiled with the following command line. Once again, you **MUST** ensure that your program compiles with this command line on the lab computers. If it fails to do this, for whatever reason, you will receive **ZERO** for the program.

- javac DSAAssignment.java

Your program will be run with the following command line. Once again, you **MUST** ensure that your program runs and produces your intended output with this command line on the lab computers. If it fails to do this, for whatever reason, you will receive **ZERO** for the program.

- java DSAAssignment "DC description file" "task file"

Where "DC description file" and "task file" are replaced by the filenames of the input files described in Sections 3 and 4. It is expected that the output, as described in Sections 3 and 4 will be displayed on standard output. Your program should NOT output anything else on standard output. This includes debugging output, which should be switched off or removed in the version that you submit.

You will need to submit the following:

6.1 Electronic Submission

- Please submit your entire assignment as a single GZip'd Tarball with the name DSA2015S2-assignment-########.tar.gz. (Replace "########" with your actual student number.) This file should contain all of the files for your submission, including your cover sheet, class diagram and source code. This should be submitted via Blackboard. This file should contain the following:
  - A Java program, consisting of one or more .java files. Note that these files must conform to the Java**6** specification and must be able to compile and run on the lab machines
  - A report in **plain ASCII text format** (Unix or DOS linebreaks are OK) with the extension of txt. **DO NOT** submit a .pdf, .doc, .odt, or any other file format.
  - Images in .png or .jpg format. This should include your class diagram and any diagrams you may need to refer to in your report. Make sure that your class diagram is named "classdiagram.[png|jpg]" and any supporting diagrams are named as referenced in your report.
- DO NOT USE RAR, ZIP, 7Z OR ANY OTHER FORMAT. Only .tar.gz files will be accepted. Failure to comply with this simple requirement **will** result in a grade of 0
- TRY DOWNLOADING AND OPENING THE FILE YOU HAVE SUBMITTED to ensure that Blackboard uploaded it correctly. You are responsible for making sure that your assignment is properly uploaded by testing in this manner.

- PRESS SUBMIT ONLY WHEN YOU ARE READY! "Save" doesn't finalise the submission, so you can use "Save" to allow later updates. But be aware that you must press Submit before the deadline **you have been warned**.

6.2 Paper Submission

- Your submission must be on A4 sized, white paper of at least 80gsm (i.e. the type of paper usually used in an office printer).
- You should include the standard Department of Computing cover page as the first page in your submission.
- All text should be easily legible, in black (except on diagrams where it may be a suitably contrasting colour).
- You must use a font no smaller than 10pt (including on diagrams), which is clearly readable. Examples of suitable fonts include "Computer Modern", "Times New Roman", "Arial", "Helvetica", "Verdana", "Courier", "Lucida Console" and their variations and equivalents. This includes source code. Do not use "fancy" fonts. Do not use characters from languages other than English.
- In the interests of saving paper, you may print double sided and single-spaced with margins down to 5mm
- Number each page consecutively. Make sure that the number of pages you submit reflects the number of pages on your cover page. Your cover page is page 1 and counts as a page.
- Make sure that you securely fasten your assignment together with staples or similar. Do NOT use temporary fasteners, binder clips or fasteners that will come loose as papers are shuffled, such as paper clips. A clear plastic envelope to contain your entire assignment when you submit it is a good idea but make sure that your pages are fastened to each other inside the envelope. Do not submit loose pages within an envelope.
- In case any pages do come adrift, ensure that your student number appears on each page. For this purpose ONLY you may use a font as small as 6pt.

End of Assignment Specification

Ensure to periodically check the Blackboard page for any updates or clarifications published. Remember, they form part of the assignment specification and can affect marking and required functionality. We recommend you check this daily. Do not rely on being informed about an announcement via email.

Good luck! :-)