Sabanci University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Summer 2019-2020

Take-Home Exam 3 – A Tour of Networks
Due: 7 August 2020 23:55 (SHARP)

DISCLAIMER:

Your program should be a robust one such that you have to consider all relevant user mistakes and extreme cases; you are expected to take actions accordingly!

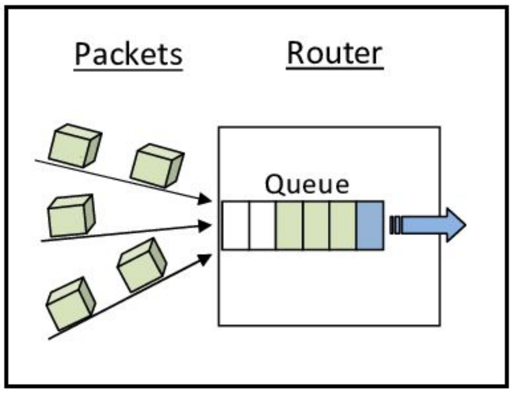
Only checking the sample run cases might not be sufficient as your solution will be checked against a variety of samples different from the provided samples; however checking these test cases are highly encouraged and recommended, as well.

You must NOT collaborate with your friends and discuss your solutions with each other. You have to write down the code on your own. Plagiarism will not be tolerated AND cooperation is not an excuse!

After this take-home exam deadline, all students will be called for a demo to explain how they approached and solved the problem.

Introduction

The aim of this take-home exam is to have you work with the queue data structure, class design issues and operator overloading. You are asked to implement a simulation of a computer network, where packets of data roam around different routers and try to reach their destination.



Inputs and Files to Read

Your program needs to read two files, one that includes information about routers, and the other one about data packets. The names of these files will be asked through standard input (cin). These two inputs will be the only inputs to your program. You may safely assume that the user enters the correct file names, hence you do not need to check the existence of the files.

The routers file starts with a line that has only a single number. This number indicates the number of intermediate routers. Apart from a router, there are two other devices: 'E' and 'T' (which stand for \`Entry\` and \`Terminal\` respectively). The second line of the file shows the forward connections of the router 'E'. It means that 'E' can forward a packet to one of these routers only. All the next lines show the connections of the routers starting from 1 to the number of intermediate routers. Please note that there is no line about the router 'T', because it is the **terminal** node and it does not make a forward connection naturally.

In a nutshell, the first line is always a number in the routers file. Starting from the second line, comes the routers. In each of these lines, the router name is followed by a " - " (space dash space) combination. In the rest of the line, the forward connected router identifiers are given with a single space break. In this routers file, a router will always be forward-connected to a router with a higher number (or to 'T') to ensure that the packages are not routed backwards (however, in real life, this is not always the case -more on this on CS408-).

The packets file is much simpler in format. In each line, there is a packet name. You may assume that this packet name does not contain any whitespace characters (space, tab, endline etc.). Other than that, <u>it may contain a combination of letters, numbers, special characters and so on</u>. The order of these packets is important for our task, as explained in the next section.

Task and Data Structure

Your task for this take-home exam is to implement an algorithm such that these packets are sent from the router 'E' to the router 'T'. Of course, there are more details to this algorithm. First of all, routers process packets in a "FIFO: first-come first-served" manner (here, processing refers to removing the packet from the queue and sending it to a forward router). This means that the first packet to arrive at that router should be sent away first. In order to ensure this order, each router should possess a queue that stores the packets. The details of how this queue should be implemented is going to be explained in the later part of this section, as well.

When the router decides to process a packet, it should pick which router it should send it to. This router should be one of the routers that this router is connected to, of course. In computer networks, this is bound to a rule in order to ensure a balanced traffic in the network. To simulate the same technique, we will send the packet to the router that has the least populated queue, among the connected routers of the current router. If two or more connected routers have the same queue size, the packet is sent to the one that occurs the first in the respective line of the current router. The same things apply for the terminal node as well.

Again, in real life, all the routers do these operations in parallel. As we do not know how to implement parallel programs yet, we need to do this simulation sequentially. Your program should start by supplying all of the packets to the router 'E'. Then, it should handle router 'E' (process all packets in the queue until empty), and then router '1', and then router '2', etc. ... At the end of your program, packets are expected to be in the router 'T'. Your program will print the packets that have reached the router 'T', in the order they reached (queue system ensures this already). Some routers may not have any forward-routers, unfortunately. Such routers will lose all the packets they receive. Due to this, we cannot guarantee that all packets will arrive at 'T'.

During the whole process, packets need to remember which routers they have visited. This information should be reflected in the final output of your program. For this purpose, you will have a vector in your packet structure. The packet structure you **must** use is the following:

```
struct Packet{
      string id;
      vector<string> routerHistory;
};
```

This packet struct should be the main item that your queue implementation stores in nodes. You already know that queues can be implemented in several ways. In this take-home exam, you **must** implement a dynamic linked-list based queue. You can start with the DynIntQueue class that we had shared with you under lecture resources. However, this class works with a single integer only. You are expected to use a Packet struct in the Queue class. Following is the QueueNode definition that you **must** use:

```
struct QueueNode{
      Packet value;
      QueueNode *next;
};
```

You can still add functions or constructors to your QueueNode struct.

In order to push you for better overall class design and practice on operator overloading, there are more restrictions on what you are expected of while implementing your Queue class:

1) **Default Constructor Obligation:** Your class must have a default constructor, which constructs an empty queue.

2) **Destructor Obligation:** Your class must have a destructor, which deallocates all the QueueNodes in the Queue.

3) **Must implement operator << for Enqueue operation:** Your class must not have a regular public queue method. Rather than that, you must overload the operator << which will have our Queue object on the left-hand side and a Packet object on the right-hand side (i.e. `myQueue << myPacket`). The Packet should be added to the end of the queue.

4) **Must implement operator >> for Dequeue operation:** Your class must not have a regular public dequeue method. Rather, you must overload the operator >> which will have our Queue object on the left-hand side and a Packet object on the right-hand side (i.e. `myQueue >> newPacket`). The first element of the queue should be removed and the information should be moved to the Packet object on the right-hand side of the operator.

5) **Must implement operator << for ostream:** Your program must implement a << operator for *ostream << queue* type of operations. This must be used for the whole final output operation, with `cout` (which is an object of *ostream*). Any other public procedures other than this operator **must not** be used for displaying the content of the queue.

If not prohibited by any of these rules, you can add other fields or methods to your class. However, we would like to remind you that any workaround which destroys the philosophy of object-oriented design will hurt your grade. If you cannot make sure whether your class design is solid/acceptable or not, you may ask your TAs about it.

These strict rules will be manually checked up during grading and also be asked to you during the demos.

Last but not least, you can store the forward-connected routers' list however you wish. A vector of strings is possibly the easiest way.

Sample Runs

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are the standard input (cin) taken from the user (i.e., like ***this***). You have to display the required information in the same order and with the same words/spaces as here; in other words, there must be an exact match!

We will be automatically grading your take-home exam using GradeChecker, so it is very important to satisfy the exact same output given in the sample runs. You can utilize GradeChecker (http://sky.sabanciuniv.edu:8080/GradeChecker/) to check whether your code is working in the expected way. To be able to use GradeChecker, you should upload all of your files used in the take-home exam **without zipping them**. Just a reminder, you will see a character ¶ which refers to a newline in your expected output.

Sample Run 1

```
Please enter the name of the file for routers: routers.txt
Please enter the name of the file for packets: packets.txt
Packets are displayed in the order they arrive in the terminal router,
along with their router visit history:
P4: E, 1, 7, T
P10: E, 1, 7, T
P16: E, 1, 7, T
P22: E, 1, 7, T
P28: E, 1, 7, T
P34: E, 1, 7, T
P12: E, 3, 5, 10, T
P30: E, 3, 5, 10, T
P6: E, 3, 5, 9, 10, T
P24: E, 3, 5, 9, 10, T
P1: E, 1, 6, 9, 10, T
P7: E, 1, 6, 9, 10, T
P13: E, 1, 6, 9, 10, T
P19: E, 1, 6, 9, 10, T
P25: E, 1, 6, 9, 10, T
P31: E, 1, 6, 9, 10, T
P37: E, 1, 6, 9, 10, T
P23: E, 2, 6, 9, 10, T
P29: E, 2, 6, 9, 10, T
P35: E, 2, 6, 9, 10, T
P18: E, 3, 5, 11, T
P36: E, 3, 5, 11, T
P2: E, 2, 8, 11, T
P5: E, 2, 8, 11, T
P8: E, 2, 8, 11, T
P11: E, 2, 8, 11, T
P14: E, 2, 8, 11, T
P17: E, 2, 8, 11, T
P20: E, 2, 8, 11, T
P26: E, 2, 8, 11, T
P32: E, 2, 8, 11, T
P3: E, 3, 4, 8, 11, T
P9: E, 3, 4, 8, 11, T
P15: E, 3, 4, 8, 11, T
P21: E, 3, 4, 8, 11, T
P27: E, 3, 4, 8, 11, T
P33: E, 3, 4, 8, 11, T
Press any key to continue . . .
```

Sample Run 2

```
Please enter the name of the file for routers: routers2.txt
Please enter the name of the file for packets: packets.txt
Packets are displayed in the order they arrive in the terminal router,
along with their router visit history:
P4: E, 1, 7, T
P10: E, 1, 7, T
P16: E, 1, 7, T
P22: E, 1, 7, T
P28: E, 1, 7, T
P34: E, 1, 7, T
P12: E, 3, 5, 10, T
P30: E, 3, 5, 10, T
P6: E, 3, 5, 9, 10, T
P24: E, 3, 5, 9, 10, T
P1: E, 1, 6, 9, 10, T
P7: E, 1, 6, 9, 10, T
P13: E, 1, 6, 9, 10, T
P19: E, 1, 6, 9, 10, T
P25: E, 1, 6, 9, 10, T
P31: E, 1, 6, 9, 10, T
P37: E, 1, 6, 9, 10, T
P29: E, 2, 6, 9, 10, T
P18: E, 3, 5, 11, T
P36: E, 3, 5, 11, T
P2: E, 2, 8, 11, T
P5: E, 2, 8, 11, T
P8: E, 2, 8, 11, T
P11: E, 2, 8, 11, T
P14: E, 2, 8, 11, T
P17: E, 2, 8, 11, T
P20: E, 2, 8, 11, T
P26: E, 2, 8, 11, T
P32: E, 2, 8, 11, T
P23: E, 2, 6, 8, 11, T
P35: E, 2, 6, 8, 11, T
Press any key to continue . . .
```

Sample Run 3

```
Please enter the name of the file for routers: routers3.txt
Please enter the name of the file for packets: packets.txt
Packets are displayed in the order they arrive in the terminal router,
along with their router visit history:
P5: E, 3, T
P14: E, 3, T
P23: E, 3, T
P32: E, 3, T
P7: E, 1, 2, 3, T
P25: E, 1, 2, 3, T
P21: E, 5, T
P27: E, 5, T
P33: E, 5, T
P3: E, 5, 7, T
P6: E, 5, 7, T
P9: E, 5, 7, T
P12: E, 5, 7, T
P15: E, 5, 7, T
P18: E, 5, 7, T
P24: E, 5, 7, T
P30: E, 5, 7, T
P36: E, 5, 7, T

Press any key to continue . . .
```

Some Important Rules

Although some of the information is given below, please also read the take-home exam submission and grading policies from the lecture notes of the first week. In order to get a full credit, your program must be efficient, modular (with the use of functions), well commented and indented. Besides, you also have to use understandable identifier names. Presence of any redundant computation, bad indentation, meaningless identifiers or missing/irrelevant comments may decrease your grade in case that we detect them.

When we grade your take-home exams, we pay attention to these issues. Moreover, in order to observe the real performance of your code, we are going to run your programs in Release mode and **we may test your programs with very large test cases**. Hence, take into consideration the efficiency of your algorithms other than correctness.

How to get help?

You may ask your questions to TAs or to the instructor. Information regarding the office hours of the TAs and the instructor are available at SUCourse+.

YOU SHOULD USE GRADE CHECKER FOR THIS TAKE-HOME EXAM!

You should use GradeChecker (http://sky.sabanciuniv.edu:8080/GradeChecker/) to check your expected grade. Just a reminder, you will see a character ¶ which refers to a newline in your expected output.

GradeChecker and the automated grading system use a different compiler than MS Visual Studio does. Hence, you should check the "***Common Errors***" page to see some extra situations to consider while doing your take-home exam. If you do not consider these situations, you may get a lower score (even zero) even if your program works correctly with MS Visual Studio.

Common Errors Page: http://sky.sabanciuniv.edu:8080/GradeChecker/commonerrors.jsp

GradeChecker can be pretty busy and unresponsive during the last day of the submission. Due to this fact, leaving the take-home exam for the last day generally is not a good idea. You may wait for hours to test your take-home exam or make an untested submission, sorrily..

GradeChecker and Sample Runs together give a good estimate of how correct your implementation is, however we may test your programs with different test cases and **your final grade may conflict with what you have seen on GradeChecker.** We will also **manually** check your code (comments, indentations and so on), hence do not object to your grade based on the GradeChecker results; but rather, consider every detail on this documentation. **So please make sure that you have read this documentation carefully and covered all possible cases, even some other cases you may not have seen on GradeChecker or Sample Runs**. The cases that you *do not need* to consider are also given throughout this documentation.

Submit via SUCourse+ ONLY! **Grade Checker is not considered as a submission**. Paper, e-mail or any other methods are not acceptable, either.

The internal clock of SUCourse+ might be a couple of minutes skewed, so make sure you do not leave the submission to the last minute. In the case of failing to submit your take-home exam on time:

"No successful submission on SUCourse+ on time = A grade of 0 directly."

What and where to submit (PLEASE READ, IMPORTANT)

You should test your program using GradeChecker. We will use the same UNIX based C++ compiler that Grade Checker uses for grading your take-home exam.

It'd be a good idea to write your name and lastname in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts).

Submission guidelines are below. Since the grading process is automatic, you are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be *zero*. The lack of even one space character in the output will result in your grade being zero, so please test your programs yourself and with the GradeChecker tool explained above.

- Name your cpp file that contains your program as follows:

  "SUCourse+UserName_the3.cpp"

  Your SUCourse+ username is actually your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SU e-mail address is **atam@sabanciuniv.edu**, then the file name must be: **"atam_the3.cpp"**

- Please make sure that this file is the latest version of your take-home exam program.

- You should upload all the .txt files to SUCourse+ as well (if any given),

- Do not zip any of the documents. Upload all of them as separate files.

- Submit your work **through SUCourse+ only**! You can use the GradeChecker only to see if your program can produce the correct outputs both in the correct order and in the correct format. It will not be considered as the official submission. You must submit your work to SUCourse+.

- If you would like to resubmit your work, you should first remove the existing file(s). This step is very important. If you do not delete the old file(s), we will receive both files and the old one may be graded.

You may visit the office hours if you have any questions regarding submissions.

Plagiarism

Plagiarism is checked by automated tools and we are very capable of detecting such cases. Be careful with that...

Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do NOT send any part of your code to your friends by any means or you might be charged as well, although you have done your take-home exam by yourself. Take-home exams are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

In case of plagiarism, the rules on the Syllabus apply.

Good Luck!
Tolga Atam, Duygu K. Altop