EE 205
Programming Assignment 3
Team: RTS

Team Members

Riley Cammack

Sasha Yamada

Sean Teramae

Task 1

Simulate a router on Internet. This router is connected to 3 networks and thus has 3 network cards. Each network card behaves like a queue.

To keep it simple, we will assume that the router has 1 input queue and 2 output queues. (In reality, each network interface has in and out queue so the router has a switching fabric inside).
To keep it simple, we will assume that all time is in seconds, in integers; and that generating a random number between integer X and integer Y is uniformly distributed in the range from X to Y.
Packets arrive to 1st queue randomly every seconds. When a packet reaches the head of the queue, it is subject to processing and that takes some time . Packets leaving 1st queue randomly go to either 2rd or 3rd queue.
If a packet arrives and finds the destination queue full, the packet is dropped and a message sent back to the sender. (Therefore, packets can be dropped from any of the three queues).
Packets come in randomly to the first queue with the time between arrivals, which is uniformly distributed between q1a and q1b seconds. Processing time for a packet at the 1st queue, is uniformly distributed between q1p and q1e seconds. Queue 1 capacity is Q1.
Processing time of the 2nd queue is uniformly distributed between q2p and q2e seconds. Queue 2 capacity is Q2.
Processing time of the 3rd queue is uniformly distributed between q3p and q3e seconds. Queue 3 capacity is Q3.
Code this situation and then test it with different times. To make it real, you can print out the contents of the queues after "sleeping" for that many seconds, so that numbers show up in "real time" and you can watch your queue grow or shrink.
Experiment with different situations – for example, make frequent arrivals to queue 1, then make packets go to queue 2 a lot more (i.e. clog the network to the right of queue 2), etc.
Calculate the average, min and max time and standard deviation of time spent waiting in each queue, and report on number of packets being dropped (total number and average).

Specify the properties of each queue in a plain text, configuration file called **config.txt** that provides one line per queue with 5 different positive integers, space-delimited, in this order, with the following variables:

q1a q1b q1p q1e Q1
q2p q2e Q2
q3p q3e Q3

Commentary:

For this homework, we began by conceptualizing the algorithm for simulating these process. First off, we begin by loading the setup variables that we would use in the simulation. This includes all ranges for load and process times, queue sizes, and packet value ranges; we call this period the setup. Next, we begin the loop by first simulating an insert at time = 0. Insertion into queue 1 is first checks to see if the queue is full. If the queue is full, the packet is dropped and an error message is given. If the queue is not full, the packet is added to the queue and a new action is created to process the new packet. Either way though, another insert action is generated at the end of attempting to insert the packet. Next, the packet is processed in the process one function. This determines whether to send the packet to queue 2 or 3. The system then checks if the chosen queue is full. If the queue is full, the packet is dropped and an error is displayed. If the queue is not full, a new action is generated to process the packet at the chosen queue and the packet is dequeued from queue 1.

We store actions such as inserting and processing in a singly linked list that is sorted by time. Time is a large factor in this loop. The actions generated are given an execution time and are executed if the time of the system is equal to the given execution time. Our global variable, time, is used as a reference to ensure that we keep track of what actions need to happen and when. Our loop simulates the process of a data packet traversing through the many nodes. The end of a loop represents the end of a second, and is what we used to increment the time. Once one run through of a loop is completed, we print the status of all three queues and the action call list. This helps to visualize the current status of the system at the end of each second.

Hand Examples:

1) insert in Q1
Q1: 25/
Q2:
Q3:

2) Process Q1
Q1:
Q2:
Q3:

3) Insert in Q2/Q3
Q1:
Q2: 25/
Q3:

4) Insert in Q1
Q1: 45/
Q2: 25/
Q3:

5) Insert in Q1
Q1: 45/70/
Q2: 25/
Q3:

6) Insert in Q1
Q1: 45/70/
Q2: 25/
Q3:

7) Process Q1
Q1: 70/
Q2: 25/
Q3:

8) Insert in Q2/Q3
Q1: 70/
Q2: 25/
Q3: 45/

9) Process Q2
Q1: 70/
Q2:
Q3: 45/

Results:

```
Insert One @ [t = 0]

Waiting... (t = 0)
0( 3 ), 1( 5 )
Average wait for Q1 | Q2 | Q3:  10 | 2 | 5
Max wait for Q1 | Q2 | Q3:      10 | 2 | 5
Min wait for Q1 | Q2 | Q3:      5 | 2 | 5
Std Deviation for Q1 | Q2 | Q3: 0 | 0 | 0
Packets Dropped (Total and average): 0 | 0

Q1:     42 /
Q2:
Q3:
```

```
Waiting... (t = 2)
0( 3 ), 1( 5 )
Average wait for Q1 | Q2 | Q3:  9.66667 | 3 | 6
Max wait for Q1 | Q2 | Q3:      11 | 4 | 7
Min wait for Q1 | Q2 | Q3:      5 | 2 | 5
Std Deviation for Q1 | Q2 | Q3: 1.24722 | 0.816497 | 0.816497
Packets Dropped (Total and average): 0 | 0

Q1:     42 /
Q2:
Q3:
```

```
Insert One @ [t = 3]
1( 5 ), 0( 5 ), 1( 7 )
Average wait for Q1 | Q2 | Q3:  8.75 | 2.75 | 6.25
Max wait for Q1 | Q2 | Q3:      11 | 4 | 7
Min wait for Q1 | Q2 | Q3:      5 | 2 | 5
Std Deviation for Q1 | Q2 | Q3: 1.92029 | 0.829156 | 0.829156
Packets Dropped (Total and average): 0 | 0

Q1:     42 / 28 /
Q2:
Q3:
```

```
Process One @ [t = 5]
0( 5 ), 1( 7 ), 3( 12 )
Average wait for Q1 | Q2 | Q3:  8.33333 | 2.5 | 6.5
Max wait for Q1 | Q2 | Q3:      11 | 4 | 7
Min wait for Q1 | Q2 | Q3:      5 | 2 | 5
Std Deviation for Q1 | Q2 | Q3: 1.69967 | 0.763763 | 0.763763
Packets Dropped (Total and average): 0 | 0

Q1:     28 /
Q2:
Q3:     42 /
```

```
Insert One @ [t = 5]
1( 7 ), 0( 8 ), 1( 11 ), 3( 12 )
Average wait for Q1 | Q2 | Q3:  8.42857 | 2.57143 | 6.42857
Max wait for Q1 | Q2 | Q3:      11 | 4 | 7
Min wait for Q1 | Q2 | Q3:      5 | 2 | 5
Std Deviation for Q1 | Q2 | Q3: 1.59079 | 0.728431 | 0.728431
Packets Dropped (Total and average): 0 | 0

Q1:     28 / 70 /
Q2:
Q3:     42 /
```

```
Process One @ [t = 7]
0( 8 ), 1( 11 ), 3( 12 ), 3( 14 )
Average wait for Q1 | Q2 | Q3:  8.44444 | 2.77778 | 6.33333
Max wait for Q1 | Q2 | Q3:      11 | 4 | 7
Min wait for Q1 | Q2 | Q3:      5 | 2 | 5
Std Deviation for Q1 | Q2 | Q3: 1.42292 | 0.785674 | 0.816497
Packets Dropped (Total and average): 0 | 0

Q1:     70 /
Q2:
Q3:     42 / 28 /
```

```
Insert One @ [t = 23]
Error, Queue is full! Packet dropped.
1( 23 ), 0( 24 ), 1( 26 )
Average wait for Q1 | Q2 | Q3:  7.96429 | 2.82143 | 6.07143
Max wait for Q1 | Q2 | Q3:      11 | 4 | 7
Min wait for Q1 | Q2 | Q3:      5 | 2 | 5
Std Deviation for Q1 | Q2 | Q3: 1.54647 | 0.804166 | 0.883523
Packets Dropped (Total and average): 1 | 0.0434783

Q1:     98 / 25 /
Q2:
Q3:
```

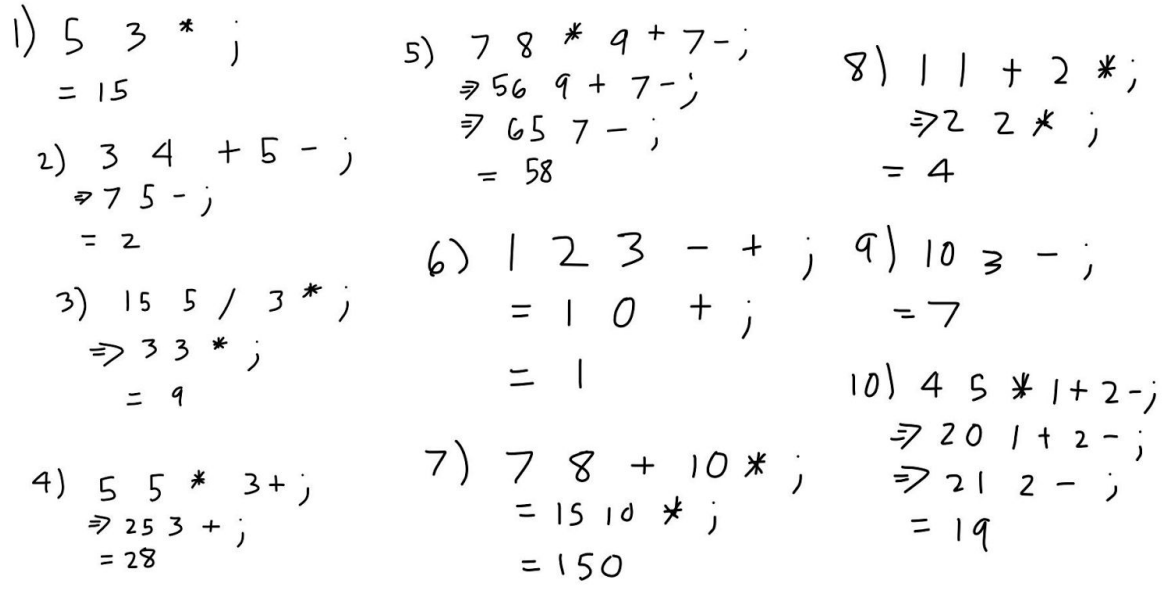
Task 2:

Do the reverse Polish Calculator from earlier lab, but using stack. The link to the description how to use the stack is in the text of that problem.

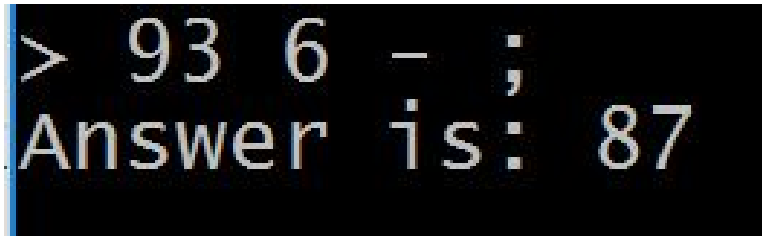
Commentary:

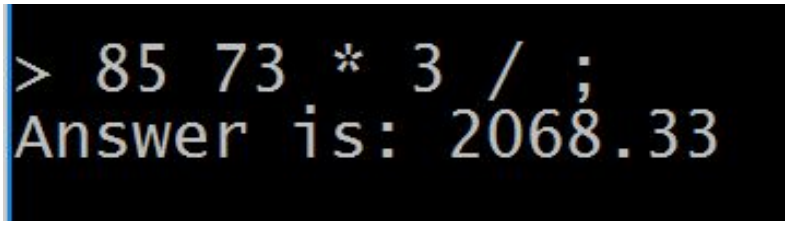
For this problem, we began by analyzing what libraries and functions we would need to complete the task. We immediately began with the stack as the center of the task. We hypothesized that the numbers would be pushed onto the stack incrementally and the processing would occur when we found an operator. The operators we supported were the standard four functions: addition, subtraction, multiplication, and division, as well as the modulus operator. Upon finding an operator in the buffer, we begin the calculations with the last two numbers found in the stack. The last two numbers are then popped from the stack but replaced with the new value from the result of the two numbers and the operand. The answer is given at the end of the loop when all the calculations have been done. If there is an unidentifiable operator or unknown character in the input string, there is an error message thrown and the result for that equation is an error. For any given equation with N amount of numbers, there must be exactly N-1 operators; and there must be at least 2 numbers at the start of an equation.

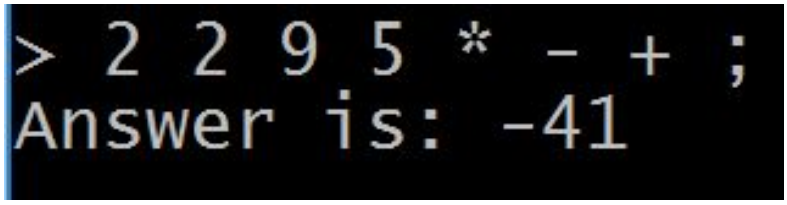
Hand Examples:

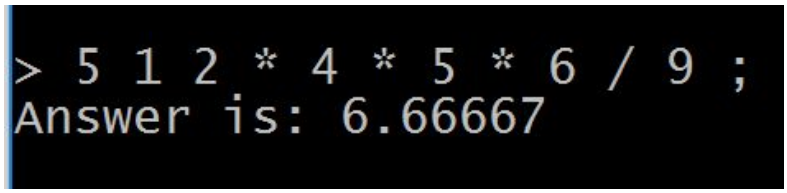


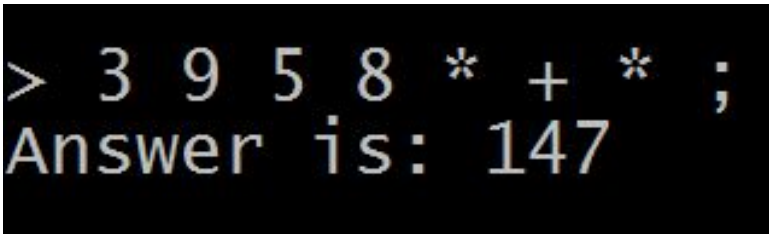
Results:

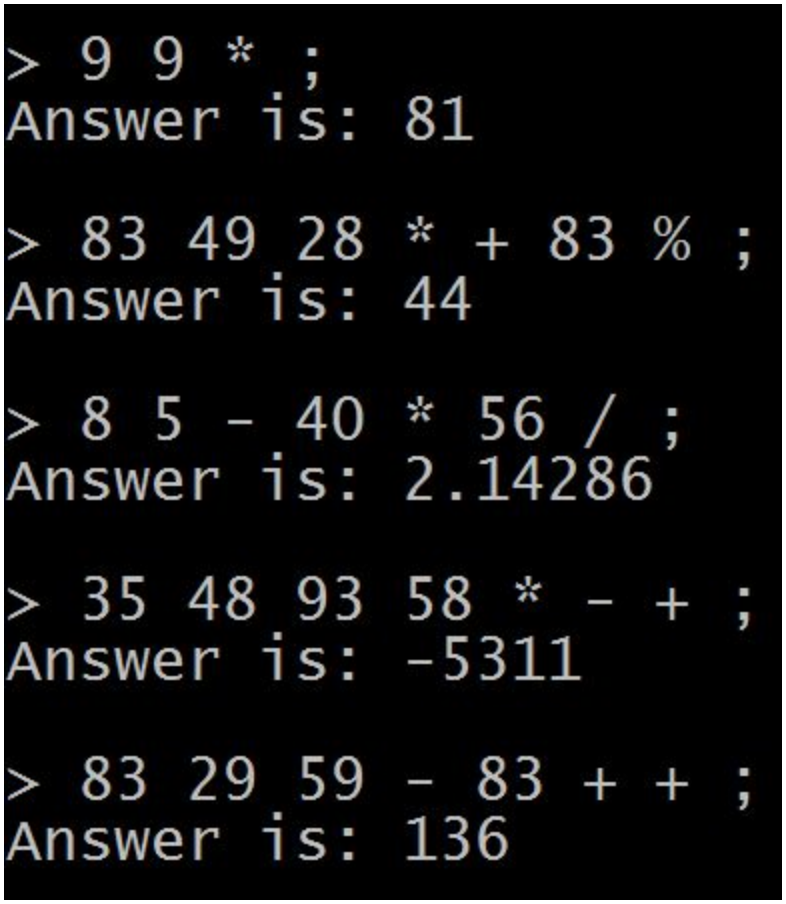

> 93 6 - ;
Answer is: 87


> 85 73 * 3 / ;
Answer is: 2068.33


> 2 2 9 5 * - + ;
Answer is: -41


> 5 1 2 * 4 * 5 * 6 / 9 ;
Answer is: 6.66667


> 3 9 5 8 * + * ;
Answer is: 147



Task 3:

Personally, I would choose to do Tamagotchi because it sounds like more fun than the other choices.