HW4

(deadline : 2019/11/12)

手寫題：

2. What would be the value of queues Q1 and Q2, and stack S after the following algorithm segment:

```
 1 S  = createStack
 2 Q1 = createQueue
 3 Q2 = createQueue
 4 enqueue (Q1, 5)
 5 enqueue (Q1, 6)
 6 enqueue (Q1, 9)
 7 enqueue (Q1, 0)
 8 enqueue (Q1, 7)
 9 enqueue (Q1, 5)
10 enqueue (Q1, 0)
11 enqueue (Q1, 2)
12 enqueue (Q1, 6)
13 loop (not emptyQueue (Q1))
   1 dequeue (Q1, x)
   2 if (x == 0)
     1 z = 0
     2 loop (not emptyStack (S))
       1 popStack (S, y)
       2 z = z + y
     3 end loop
     4 enqueue (Q2, z)
   3 else
     1 pushStack (S, x)
   4 end if
14 end loop
```

4. What would be the contents of queue `Q1` and queue `Q2` after the following code is executed and the following data are entered?

```
1 Q1 = createQueue
2 Q2 = createQueue
3 loop (not end of file)
   1 read number
   2 enqueue (Q1, number)
   3 enqueue (Q2, number)
   4 loop (not empty Q1)
      1 dequeue (Q1, x)
      2 enqueue (Q2, x)
   5 end loop
4 end loop
```

The data are 5, 7, 12, 4, 0, 4, 6.

12. Write an algorithm called `queueToStack` that creates a stack from a queue. At the end of the algorithm, the queue should be unchanged; the front of the queue should be the top of the stack, and the rear of the queue should be the base of the stack.

程式題：

20. One way to evaluate a prefix expression is to use a queue. To evaluate the expression, scan it repeatedly until you know the final expression value. In each scan read the tokens and store them in a queue. In each scan replace an operator that is followed by two operands with their calculated values. For example, the following expression is a prefix expression that is evaluated to 159:

```
- + * 9 + 2 8 * + 4 8 6 3
```

We scan the expression and store it in a queue. During the scan when an operator is followed by two operands, such as + 2 8, we put the result, 10, in the queue.

After the first scan, we have

```
- + * 9 10 * 12 6 3
```

After the second scan, we have

```
- + 90 72 3
```

After the third scan, we have

```
- 162 3
```

After the fourth scan, we have

```
159
```

22. Using the C ADT, write a program that simulates the operation of a telephone system that might be found in a small business, such as your local pizza parlor. Only one person can answer the phone (a single-server queue), but there can be an unlimited number of calls waiting to be answered.

Queue analysis considers two primary elements, the length of time a requester waits for service (the queue wait time—in this case, the customer calling for pizza) and the service time (the time it takes the customer to place the order). Your program should simulate the operation of the telephone and gather statistics during the process.

The program requires two inputs to run the simulation: (1) the length of time in hours that the service is provided and (2) the maximum time it takes for the operator to take an order (the maximum service time).

Four elements are required to run the simulation: a timing loop, a call simulator, a call processor, and a start call function.

a. **Timing loop**: This is simply the simulation loop. Every iteration of the loop is considered 1 minute in real time. The loop continues until the service has been in operation the requested amount of time (see input above). When the operating period is complete, however, any waiting calls must be answered before ending the simulation. The timing loop has the following subfunctions:

   - Determine whether a call was received (call simulator)
   - Process active call
   - Start new call

   This sequence allows a call to be completed and another call to be started in the same minute.

b. **Call simulator**: The call simulator uses a random-number generator to determine whether a call has been received. Scale the random number to an appropriate range, such as 1 to 10.

   The random number should be compared with a defined constant. If the value is less than the constant, a call was received; if it is not, no call was received. For the simulation set the call level to 50%; that is, on the average, a call is received every 2 minutes. If a call is received, place it in a queue.

c. **Process active call**: If a call is active, test whether it has been completed. If completed, print the statistics for the current call and gather the necessary statistics for the end-of-job report.

d. **Start new call**: If there are no active calls, start a new call if there is one waiting in the queue. Note that starting a call must calculate the time the call has been waiting.

   During the processing, print the data shown in Table 4-2 after each call is completed. (*Note*: you will not get the same results.)

Clock time	Call number	Arrival time	Wait time	Start time	Service time	Queue size
4	1	2	0	2	3	2
6	2	3	2	5	2	4

TABLE 4-2 Sample Output for Project 22

At the end of the simulation, print out the following statistics gathered during the processing. Be sure to use an appropriate format for each statistic, such as a float for averages.

a. Total calls: calls received during operating period
b. Total idle time: total time during which no calls were being serviced
c. Total wait time: sum of wait times for all calls
d. Total service time: sum of service time for all calls
e. Maximum queue size: greatest number of calls waiting during simulation
f. Average wait time: total wait time/number of calls
g. Average service time: total service time/number of calls

Run the simulator twice. Both runs should simulate 2 hours. In the first simulation, use a maximum service time of 2 minutes. In the second run, use a maximum service time of 5 minutes.