2021

Assignment



**Subject: Computer Network And Programming**

**Name: MD AYNUL ISLAM**

**Chinese Name: 叶子**

**Student ID: 4420190030**

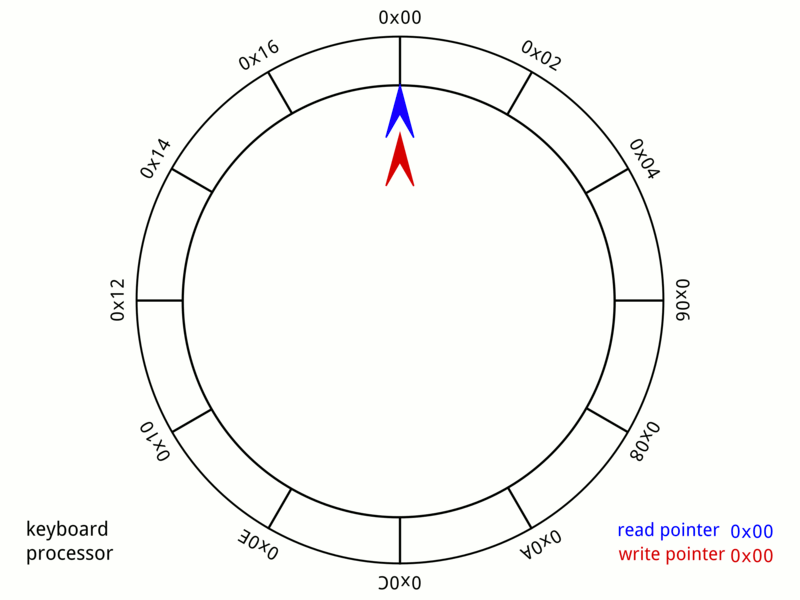
**Assignment 1**

Design a buffer class, to achieve operations including but not limited bellow and test codes.

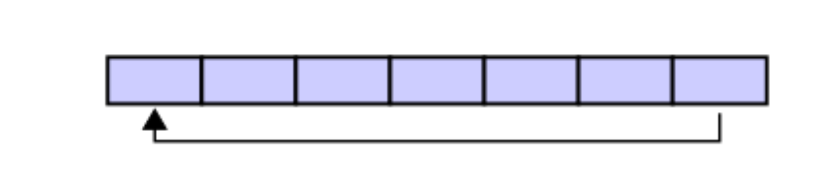
1. init a buffer
2. put element
3. get element
4. get the occupied size of the buffer
5. get the left size of the buffer
6. free the buffer

Answer:

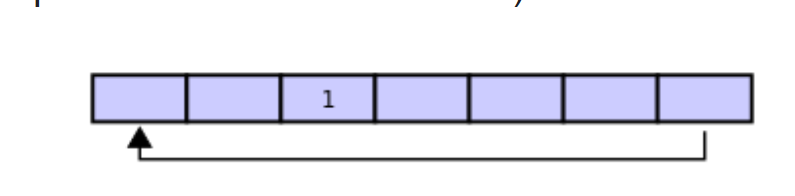
Circular buffer is a FIFO data structure that treats memory to be circular; that is, the read/write indices loop back to 0 after it reaches the buffer length. This is achieved by two pointers to the array, the “head” pointer and the “tail” pointer. As data is added (write) to the buffer, the head pointer is incremented and likewise, when the data is being removed (read) the tail pointer is incremented. The definition of head, tail, their movement direction and write and read location are all implementation dependent but the idea/goal remains the same. So, for the sake of this discussion, we will consider, that a write is done at head and read at tail.

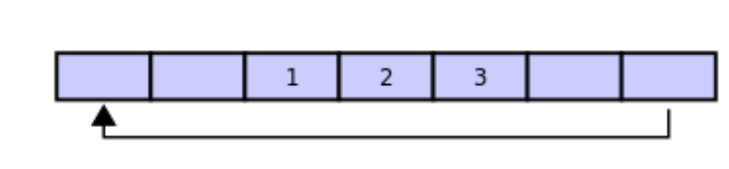


A circular buffer first starts out empty and has a set length. In the diagram below is a 7-element buffer:

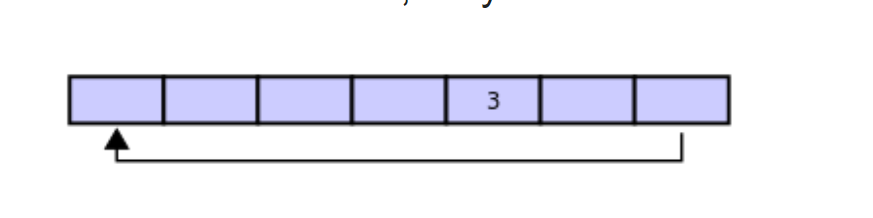


Assume that 1 is written in the center of a Circular Buffer (the exact starting location is not important in a Circular Buffer):

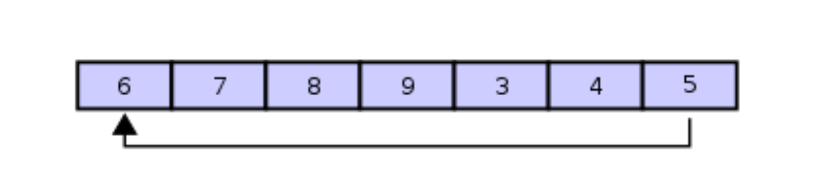


Then assume that two more elements are added to the Circular Buffer — 2 & 3 — which get put after 1:

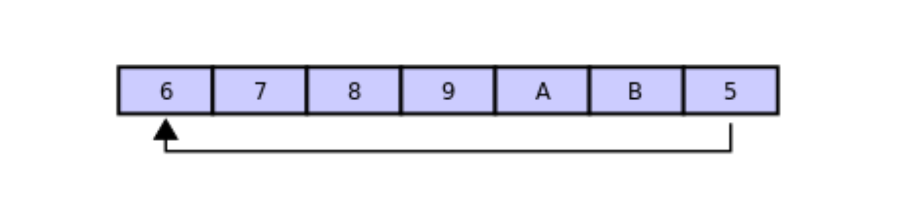
If two elements are removed, the two oldest values inside of the Circular Buffer would be removed. Circular Buffers use FIFO (First In, First Out) logic. In the example 1 & 2 were the first to enter the Circular Buffer, they are the first to be removed, leaving 3 inside of the Buffer.



If the buffer has 7 elements, then it is completely full:

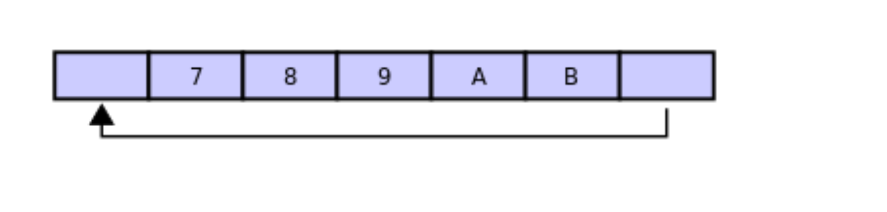


A property of the circular buffer is that when it is full and a subsequent write is performed, then it starts overwriting the oldest data. In the current example, two more elements — A & B — are added and they *overwrite* the 3 & 4:



Alternatively, the routines that manage the buffer could prevent overwriting the data and return an error or raise an [exception](https://en.wikipedia.org/wiki/Exception_handling). Whether or not data is overwritten is up to the semantics of the buffer routines or the application using the circular buffer.

Finally, if two elements are now removed then what would be returned is **not** 3 & 4 but 5 & 6 because A & B overwrote the 3 & the 4 yielding the buffer with:



1.

Initialize the CircularBuffer with a max\_size if set, otherwise

max\_size will elementsdefault to 10

|  |
| --- |
| **def** **\_\_init\_\_**(self, max\_size=**10**):  self.buffer = [None] \* max\_size  self.head = **0**  self.tail = **0**  self.max\_size = max\_size |

2.

Insert an item at the back of the CircularBuffer

|  |
| --- |
| **def** enqueue(self, item):  **if** self.is\_full():  **raise** **OverflowError**(  "CircularBuffer is full, unable to enqueue item")  self.buffer[self.tail] = item  self.tail = (self.tail + 1) % self.max\_size |

3.

Return the item at the front of the Circular Buffer and remove it

|  |
| --- |
| **def** dequeue(self):  **if** self.is\_empty():  **raise** **IndexError**("CircularBuffer is empty, unable to dequeue")  item = self.buffer[self.head]  self.buffer[self.head] = None  self.head = (self.head + 1) % self.max\_size  **Return** item |

4. For get the occupied size of the buffer

{int(n) - int(cb.buffer.count(None))}

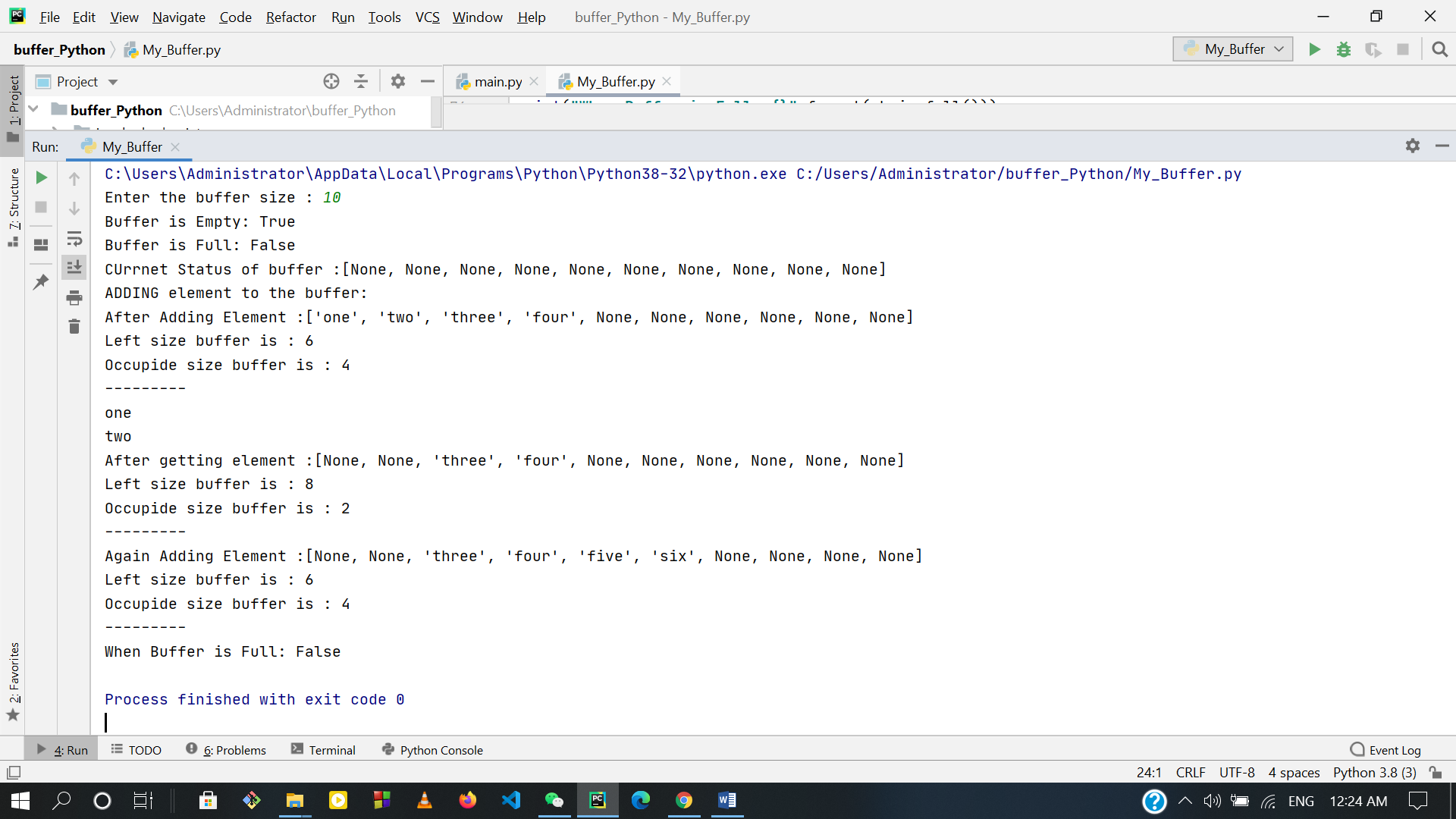
5. For get the left size of the buffer

{cb.buffer.count(None)}

**Source Code Description:**

|  |
| --- |
| **class** **CircularBuffer**(object):  **def** **\_\_init\_\_**(self, max\_size):  self.buffer = [None] \* max\_size  self.head = **0**  self.tail = **0**  self.max\_size = max\_size  **def** **\_\_str\_\_**(self):  items = ['{!r}'.format(item) **for** item **in** self.buffer]  **return** '[' + ', '.join(items) + ']'  **def** **size**(self):  **if** self.tail >= self.head:  **return** self.tail - self.head  **return** self.max\_size - self.head - self.tail  **def** **is\_empty**(self):  **return** self.tail == self.head  **def** **is\_full**(self):  **return** self.tail == (self.head - **1**) % self.max\_size  **def** **enqueue**(self, item):  **if** self.is\_full():  **raise** **OverflowError**(  "CircularBuffer is full, unable to enqueue item")  self.buffer[self.tail] = item  self.tail = (self.tail + **1**) % self.max\_size  **def** **front**(self):  **return** self.buffer[self.head]  **def** **dequeue**(self):  **if** self.is\_empty():  **raise** **IndexError**("CircularBuffer is empty, unable to dequeue")  item = self.buffer[self.head]  self.buffer[self.head] = None  self.head = (self.head + **1**) % self.max\_size  **return** item  **def** **Size\_off\_BUFF**(self):  **if** self.tail >= self.head:  **return** self.tail - self.head  # Examples  n = input('Enter the buffer size : ')  cb = CircularBuffer(int(n))  **print**("Buffer is Empty: {}".format(cb.is\_empty()))  **print**("Buffer is Full: {}".format(cb.is\_full()))  **print**("CUrrnet Status of buffer :" + (str(cb)))  **print**("ADDING element to the buffer: ")  cb.enqueue("one")  cb.enqueue("two")  cb.enqueue("three")  cb.enqueue("four")  **print**("After Adding Element :" + (str(cb)))  **print**(f'Left size buffer is : {cb.buffer.count(None)}')  **print**(f'Occupide size buffer is : {int(n) - int(cb.buffer.count(None))}')  **print**('---------')  **print**(cb.dequeue())  **print**(cb.dequeue())  **print**("After getting element :" + (str(cb)))  **print**(f'Left size buffer is : {cb.buffer.count(None)}')  **print**(f'Occupide size buffer is : {int(n) - int(cb.buffer.count(None))}')  **print**('---------')  cb.enqueue("five")  cb.enqueue("six")  **print**("Again Adding Element :" + (str(cb)))  **print**(f'Left size buffer is : {cb.buffer.count(None)}')  **print**(f'Occupide size buffer is : {int(n) - int(cb.buffer.count(None))}')  **print**('---------')  **print**("When Buffer is Full: {}".format(cb.is\_full())) |

**Program result:**

****

Reference:

1. More about circular Buffer, [click here](https://en.wikipedia.org/wiki/Circular_buffer).
2. For converts The code snippets into pretty-printed HTML format, [click here.](http://hilite.me/)
3. Read About Buffer Protocol. [Click here.](https://docs.python.org/3/c-api/buffer.html)
4. For Code sample. [click here.](https://stackoverflow.com/questions/4151320/efficient-circular-buffer)