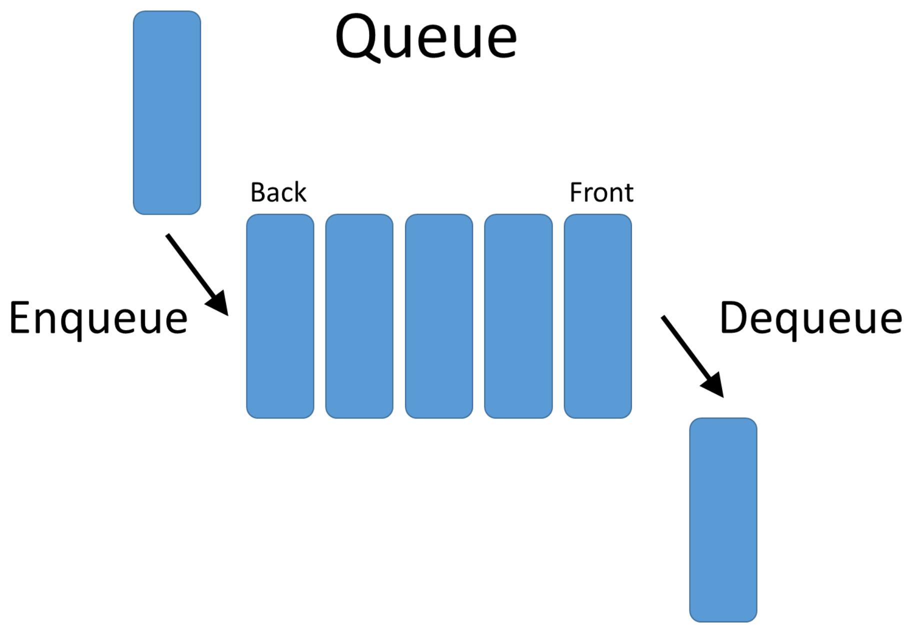
Assignment 3: Queue

Task 1: Queue Data structure

A queue is a container associated with objects inserted or removed based on the fast-in first-out (FIFO) principle.



In this assignment, you need to implement a Queue ADT (abstract data type) as defined in Table 1.

Table 1: Queue ADT definition.

Fn #	Function	Param.	Ret.	After functions execution	Comment
	[before each function execution.]			<>	Initially, the queue is empty
1	enqueue(item)	20		<20>	enqueue an element.
		23		<20, 23 >	enqueue an element.
		12		<20, 23 , 12>	enqueue an element.
		15		<20, 23 , 12, 15>	enqueue an element.
		19		<20, 23, 12, 15, 19>	enqueue an element.
2	dequeue()		20	< 23, 12, 15, 19>	dequeue an element.
3	length()	-	4	< 23, 12, 15, 19>	Return the number of elements in the queue.
4	front()	-	23	< 23, 12, 15, 19>	Return the front element.
5	back()		19	< 23, 12, 15, 19>	Return the back element.
6	is_empty()	-	False	< 23, 12, 15, 19>	If we have an empty queue, then this function will return true; otherwise, it will return false.
7	clear()	-		<>	Reinitialize the queue, i.e., make it (logically) an empty queue. <> means an empty queue.

- You have to implement using the concepts of **OOP in C++** programming language.
- The implementation must at least support the **int** data type of elements. You can also use templates in C++. If you implement using **templates**, you will get **bonus marks**.
- You need to provide both **Linked List based** and **Array based implementation**.
- You have to use the implemented list (**Linked-list based**) from Assignment-1 for the linked list based implementation of queue. You can modify your submitted implementation of Assignment-1 if required.

- You have to use the implemented list (**Array based**) from Assignment-1 for the array based implementation of the queue. You can modify your submitted implementation of Assignment-1 if required.
- The array-based implementation of the queue must grow **circularly**.
- The **average time complexity** of all the functions in both implementations has to be **O(1)**.
- **You may implement extra helper functions, but those will not be available for programmers to use**. So, while using the queue implementations, one can only use the methods listed in Table 1.
- The queue is **initially empty** and has **no fixed size**.
- You should have two separate headers and implementation cpp files for the two implementations and one main file to run the program as specified in the input output format, **totaling five files.**

Input format (for checking the implementation):

Several lines (indicating the tasks to perform on the queue), each containing one or two integers, Q (Fn #), 0<=Q<=6, and P (parameter) (Only for enqueue function). For each line, the program will execute Fn # Q. If Q is 0, the program will exit.

Output Format:

At first, the system will output in one line the items of the queue within the angle bracket, space separated, as shown in Table 1. Then, for each task, it will output two lines as follows:

The first line will output the queue items, space separated as shown in Table 1, and the second line will output the return value (if available).

Please output the necessary messages in the case of any corner scenario.

Task 2: Using the Queue

In this task, you have to implement the Stack ADT as defined in Assignment-2 using the Queue ADT.

- You have to implement using the concepts of **OOP in C++** programming language.
- The queue elements can be any of **int, char, double, or generic** (c++ templates).
- You should have separate header and implementation cpp file and one main file to run the program as specified in the input output format, **totaling three files.**

Input format (for checking the implementation):

Same as Task-1. The operation functions will be from Assignment-2

Output Format:

Same as Task-1

Special Instructions:

Write **readable, re-usable, well-structured, quality** code. This includes but is not limited to writing appropriate functions for implementation of the required algorithms, meaningful naming of the variables, suitable comments where needed, **proper indentation**, etc.

Please **DO NOT COPY** solutions from anywhere (your friends, seniors, the internet, etc.). Any form of plagiarism (irrespective of source or destination) will result in getting **-100% marks** in the assignment.

Submission Guidelines:

1. Create a directory with your 7-digit student id as its name.
2. The prefix of all of your source files must have your student id and task no (i.e 2105001_1_ll_queue.h, 2105001_1_ll_queue.cpp, 2105001_1_a_queue.h,

2105001_1_a_queue.cpp, 2105001_1_main.cpp, 2105001_2_stack.h, 2105001_2_stack.cpp, 2105001_2_main.cpp)

3. Put **all the source files that are required to run the program, including Assignment-1 files** (only .h and .cpp files), into the directory created in Step 1.
4. Zip the directory (compress in **.zip** format; .rar, .7z, or any other format is **not** acceptable)
5. Upload the zipped file to Moodle.
6. Failure to follow the above-mentioned submission guidelines will result in a **10% penalty.**

Submission Deadline: July 15, 2023, 11:55 p.m.

Evaluation Policy:

Task	Implementation	
1	Linked list based queue	30%
	Circular Array based queue	35%
2	Implementation of stack	35%
Bonus	Use templates to implement queue in Task 1	5%