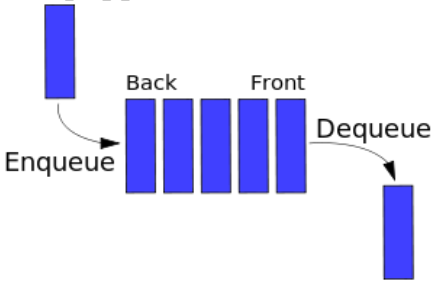
Queue

Queues are a type of container adaptor, specifically designed to operate in a FIFO context (***first-in first-out***), where elements are inserted into one end of the container and extracted from the other.

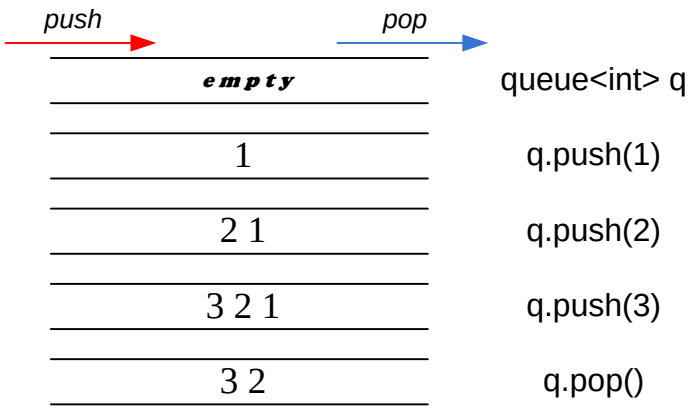
FIFO is an acronym for **first in, first out**, a method for organizing and manipulating a data buffer, where the oldest (first) entry, or 'head' of the queue, is processed first. It is analogous to processing a queue with first-come, first-served (FCFS) behaviour: where the people leave the queue in the order in which they arrive.

Queues are implemented as containers adaptors, which are classes that use an encapsulated object of a specific container class as its underlying container, providing a specific set of member functions to access its elements. Elements are pushed into the "**back**" of the specific container and popped from its "**front**".



The underlying container may be one of the standard container class template or some other specifically designed container class. This underlying container shall support at least the following operations:

- ***empty*** – test whether container is empty;
- ***size*** – return size;
- ***front*** – access first element
- ***back*** – access last element
- ***push*** – insert element to the back;
- ***pop*** – remove front element;

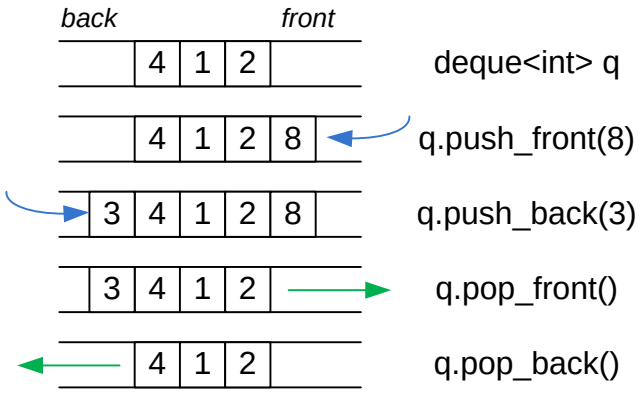


Push numbers into queue and pop then in FIFO order:

```
#include <cstdio>
#include <queue>
using namespace std;

queue<int> q;

int main(void)
{
  q.push(1); q.push(2); q.push(3); q.push(4); q.push(5);

  printf("Queue contains %d elements\n", q.size()); // 5
  printf("Front = %d, Back = %d\n", q.front(),q.back());
  // Front = 1, Back = 5
  while (!q.empty())
  {
    printf("%d ", q.front()); // 1 2 3 4 5 from front to back
    q.pop();
  }
  printf("\n");
  return 0;
}
```

Deque

Deque or ***Double Ended Queue*** is a generalized version of Queue data structure that allows insert and delete at both ends.



- ***empty*** – test whether container is empty;
- ***size*** – return size;
- ***clear*** – clear the deque;
- ***front*** – access first element
- ***back*** – access last element
- ***push_back*** – insert element to the back;
- ***push_front*** – insert element to the front;
- ***pop_back*** – remove back element;
- ***pop_front*** – remove front element;

```
#include <cstdio>
```

```
#include <queue>
using namespace std;

deque<int> q;

void print(void)
{
  for (int i = 0; i < q.size(); i++)
    printf("%d ", q[i]);
  printf("\n");
}

int main(void)
{
  q.push_back(1); q.push_back(2); q.push_front(3);
  q.push_front(4); q.push_back(5);
  printf("Deque contains %d elements\n", q.size());
  print();

  q.pop_back(); q.pop_front();
  print();

  printf("Front = %d, back = %d\n", q.front(),q.back());
  return 0;
}
```

Working with deque in Java.

```
import java.util.*;

public class Main
{
  static void print(Deque<Integer> q)
  {
    for(Integer i: q)
      System.out.print(i + " ");
    System.out.println();
  }

  public static void main(String[] args)
  {
    Scanner con = new Scanner(System.in);
    Deque<Integer> q = new LinkedList<Integer>();
    q.addLast(1); q.addLast(2); q.addFirst(3);
    q.addFirst(4); q.addLast(5);
    System.out.println("Deque contains " + q.size() + " elements");
    print(q);

    q.removeLast(); q.removeFirst();
    print(q);

    System.out.println("Front = " + q.getFirst());
    System.out.println("Back = " + q.getLast());
    con.close();
  }
}
```

E-OLYMP 6128. Simple deque Implement deque data structure.

► Implement commands for double queue.

E-OLYMP 6129. Deque with error protection Implement deque data structure with error protection.

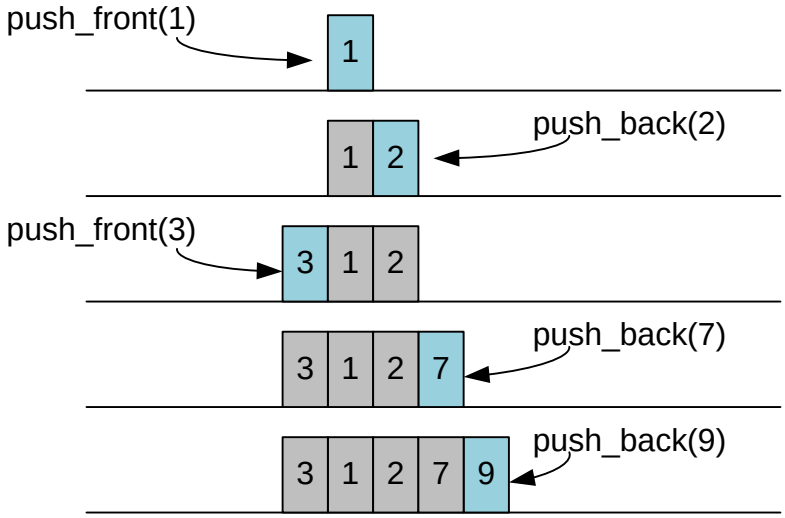
► If you run operation **pop_front**, **pop_back**, **front**, **back** and the deque is empty, the program should print line ***error*** instead of integer value.

E-OLYMP 8355. Book shelf Simulate the movements of books on the shelf.

► The shelf where the books are placed represents itself a double ended queue. In this problem you must simulate the next operations:

push_front(*x*) – put the book number *x* from the left side;
push_back(*x*) – put the book number *x* from the right side;
pop_front() – remove the book from left side;
pop_back() – remove the book from right side;

Consider the order in which the books are placed on the shelf (from sample input).



The books are removed from the shelf in the following order:

pop_front() – book number 3;
pop_back() – book number 9;
pop_front() – book number 1;
pop_front() – book number 2;
pop_back() – book number 7;

E-OLYMP 3161. Deques on 6 Megabytes Write a program that operates with a big number of deques. Deque is a "queue with two ends".

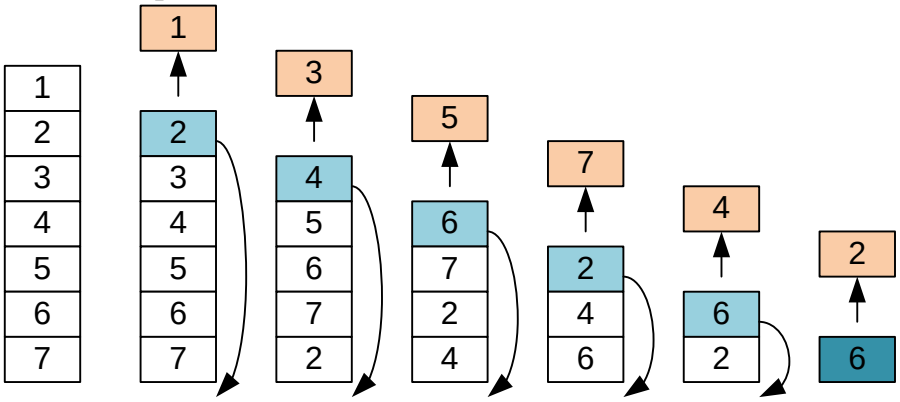
► Since the parameter A is no more than 150000, there will be no more than 150000 deques. Let's declare an array of 150000 deque structures. Then simulate the deques operations.

E-OLYMP 10143. Throwing cards away Given is an ordered deck of *n* cards numbered from 1 to *n* with card 1 at the top and card *n* at the bottom. The following

operation is performed as long as there are at least two cards in the deck: throw away the top card and move the card that is now on the top of the deck to the bottom of the deck. Your task is to find the sequence of discarded cards and the last, remaining card.

► For the given number *n*, initialize the deck of cards – fill the double - sided queue with the numbers 1, 2,…, *n*. Then, while the queue contains more than one element, make *n* – 1 iterations: print and remove the top card, and then put the top card down the deck.

Let's simulate the operations with a deck of cards for *n* = 7.



Card number 6 is the remaining card.

E-OLYMP 10142. Power of time Simulate *n* processes.

► Declare deque and start simulation like it is written in the problem statement.