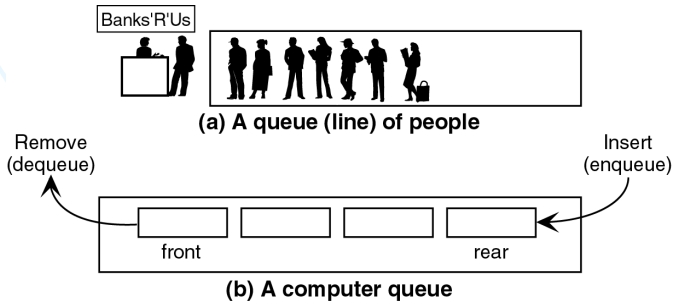
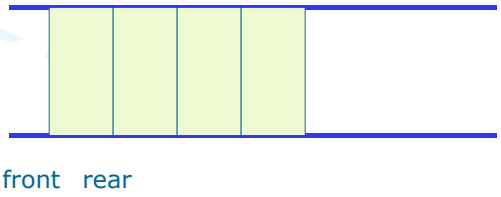
Queue

- A queue is a waiting line
- Both ends are used: one for adding elements and one for removing them.
- Data is inserted (enqueued) at the rear, and removed (dequeued) at the front



Data structure FIFO

- Queue items are removed in exactly the same order as they were added to the queue
  - FIFO structure: First in, First out



Operations on queue

- *Queue* CreateQ(*max_queue_size*) ::=
  create an empty queue whose maximum size is *max_queue_size*

- *Boolean* IsFullQ(*queue, max_queue_size*) ::=
  if(number of elements in *queue* == *max_queue_size*)
  return *TRUE*
  else return *FALSE*

Operations on queue

- *Queue* EnQ(*queue, item*) ::=
  if (IsFullQ(*queue*)) *queue_full*
  else insert *item* at rear of *queue* and return *queue*
- *Boolean* IsEmptyQ(*queue*) ::=
  if (*queue* ==CreateQ(*max_queue_size*))
  return *TRUE*
  else return *FALSE*
- *Element* DeQ(*queue*) ::=
  if (IsEmptyQ(*queue*)) **return**
  else remove and return the *item* at front of queue.

Implementation using array and structure

```
#define MaxLength 100
typedef ... ElementType;
typedef struct {
ElementType Elements[MaxLength];
//Store the elements
int Front, Rear;
} Queue;
```

Initialize and check the status

```
void MakeNull_Queue(Queue *Q){
  Q->Front=-1;
  Q->Rear=-1;
}
int Empty_Queue(Queue Q){
 return Q.Front==-1;
}
int Full_Queue(Queue Q){
return (Q.Rear-Q.Front+1)==MaxLength;
}
```

Enqueue

```
void EnQueue(ElementType X,Queue *Q){
if (!Full_Queue(*Q)){
  if (Empty_Queue(*Q)) Q->Front=0;
  Q->Rear=Q->Rear+1;
  Q->Element[Q->Rear]=X;
}
else printf("Queue is full!");
}
```

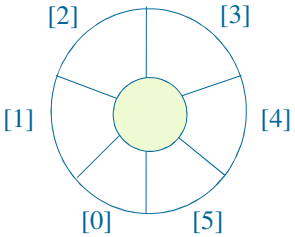
Dequeue

```
ElementType DeQueue(Queue *Q){
  ElementType e;
  if (!Empty_Queue(*Q)){
     e = Q->Elements[Q->Front];
     Q->Front=Q->Front+1;
     if (Q->Front > Q->Rear)
     MakeNull_Queue(Q);
     // Queue become empty
    return e;
}
else printf("Queue is empty!");
}
```

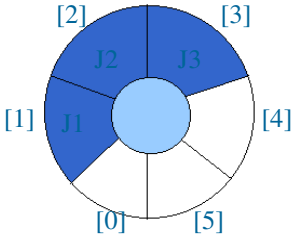
Implementation 2: regard an array as a circular queue

front: one position counterclockwise from the first element
rear: current end

EMPTY QUEUE



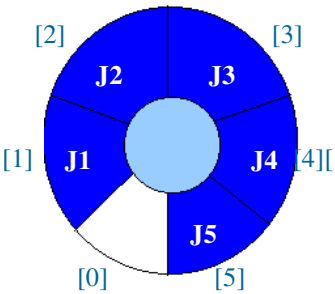
front = 0
rear = 0



front = 0
rear = 3

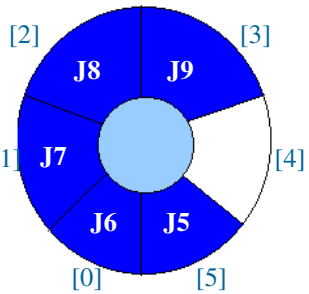
Problem: one space is left when queue is full

FULL QUEUE



front =0
rear = 5

FULL QUEUE



front =4
rear =3

Queue is full or not?

```
int Full_Queue(Queue Q){
 return (Q.Rear-Q.Front+1) %
  MaxLength==0;
}
```

Dequeue

```
void DeQueue(Queue *Q){
if (!Empty_Queue(*Q)){
//if queue contain only one element
  if (Q->Front==Q->Rear) MakeNull_Queue(Q);
  else Q->Front=(Q->Front+1) % MaxLength;
  }
else printf("Queue is empty!");
}
```

Enqueue

```
void EnQueue(ElementType X,Queue *Q){
if (!Full_Queue(*Q)){
  if (Empty_Queue(*Q)) Q->Front=0;
  Q->Rear=(Q->Rear+1) % MaxLength;
  Q->Elements[Q->Rear]=X;
} else printf("Queue is full!");
}
```

Implementation using a List

- Exercise: A Queue, is a list specific. Implement operations on queue by reusing implemented operations of list.

Implementation using a List

```
typedef ... ElementType;
typedef struct Node{
 ElementType Element;
 Node* Next; //pointer to next element
 };
 typedef Node* Position;
 typedef struct{
  Position Front, Rear;
  } Queue;
```

Initialize an empty queue

```
void MakeNullQueue(Queue *Q){
  Position Header;
  Header=(Node*)malloc(sizeof(Node));
  //Allocation    Header
  Header->Next=NULL;
  Q->Front=Header;
  Q->Rear=Header;
}
```

Is-Empty

```
int EmptyQueue(Queue Q){
return (Q.Front==NULL);
}
```

EnQueue

```
void EnQueue(ElementType X, Queue *Q){
  Q->Rear->Next=
  (Node*)malloc(sizeof(Node));
  Q->Rear=Q->Rear->Next;
  Q->Rear->Element=X;
  Q->Rear->Next=NULL;
}
```

Dequeue

```
ElementType DeQueue(Queue *Q){
ElementType e;
if (!Empty_Queue(*Q)){
  e=Q->Elements[Q->Front];
//if queue contain only one element
  if (Q->Front==Q->Rear) MakeNull_Queue(Q);
  else Q->Front=(Q->Front+1) % MaxLength;
  return e;
  }
else printf("Queue is empty!");
}
```

HW: Library implementation

- Implement two Queue library, one use array, one use linked list.

Exercise 4-3: Queues Using Lists

- We assume that you write a mobile phone's address book.
- Declare a structure "Address" that can hold at least "name", "telephone number" and "e-mail address".
- Write a program that copies data of an address book from the file to other file using a queue. First, read data of the address book from the file and add them to the queue. Then retrieve data from the queue and write them to the file in the order of retrieved. In other words, data read in first should be read out first and data read in last should be read out last.

Exercises

- Make a queue that holds integers. The size of the queue is fixed to 10.
- Read integers separated by spaces from the standard input, and add them to the queue. When the program reads the 11th integer, the queue is already full. So the program removes the first integer and adds the 11th integer. Print the removed integer to the standard output.
- Process all the integers in this way.

Exercise:Booking management

- A plane has 50 rows of seat: A B C D E F
- By using a queue, write a Air ticket Booking management program with a menu for adding, canceling, modifying requests about ticket from client.
- A ticket request has the following fields:
  - Flight Number
  - Name of client
  - Booking Time
  - Quantity.
  - Seat type: W/C/N (W: C,F C: A, D, N: B,E)
- The result is: Accept or Refuse/Wait. If accept – the system reserve a seat for each ticket end inform the users. The time field can be the system time at the moment of input.

Queue Exercise 4EF

- **Simulate a computer that process computing request from OS's programs.**
- **Configuration Input:**
  - **Number of parallel process it can run**
  - **Memory capacity**
- **Program has the menu:**
  - **Create new program (with a given amount of necessary memory and ID)**
  - **Kill a program**
  - **Show the status of running and waitting processes.**

GUI example

- number of parallel process: 2
- memory capacity (MB): 100
- 1. Create new program run 1st time:
  - the memory size of program? 40
  - Program ID? 1 → successful. Process created

  2. Show the status:

ID	Memory
1	40

  1. Create new program run 2th time:
  - the memory size of program? 70
  - Program ID? 2 → in Queue as there is not enough memory (60 vs 70)

  1. Create new program run 2th time:
  - the memory size of program? 50
  - Program ID? 3 → successful. Process created

  2. Show the status:

ID	Memory	Queue
1	40	
3	50	
2		70

- 3. Kill process: Process ID ? 1 - Success

GUI example

2. Show the status:

ID	Memory	Queue
3	50	
2		70

- 1. Create new program run 3st time:
  - the memory size of program? 60
  - Program ID? 4 → in Queue

  3. Kill process: Process ID ? 3 – Success

  2. Show the status:

ID	Memory	Queue
2	70 (vì bộ nhớ trống đã đủ, tiến trình từ queue sẽ vào bộ nhớ)	
4		60

Exercises: Simulating service counters at the bank

- **Usually there are customer service desks at banks - performing withdrawal, deposit, etc.**
- **Write a simulation program of this activity at a bank whose parameter is the number of counters (actually each counter is a queue). The program provides functionalities to add a customer, asking for information about arrival time. Each customer is served for 15 minutes**
- **The program will print the results: Instruct guests to enter the counter (specify the counter number) and how long the waiting time for guests.**
- **The program statistics the time guests have to wait in a day:**
  - **Total number of guests**
  - **Total waiting time of the guests**
  - **Average waiting time**

Guide

- You can create customer data to the bank branch in the file according to the following structure:
- Time Number of customer
- 9:00 2
- 9:10 1
- 9:25 3
- 9:40 2

Interaction bw program and users

- BIDV Bank – Branhc Hà Thành 17 Tạ Quang Bửu
- Number of counters to simulate:2
- Reading simulation data file….
- 9:00 2
- The first guest is served at counter 1 and must wait: 0
- The second guest is served at counter 2 and must wait: 0
- 9:10 1
- The first guest is served at counter 1 and must wait: 5 (served at 9:15)
- 9:25 3
- The first guest is served at counter 2 and must wait: 0
- The second guest is served at counter 1 and must wait : 5
- The third guest is served at counter 2 and must wait : 15
- …
- Total: XYZ customers – total waiting time, average waiting time.