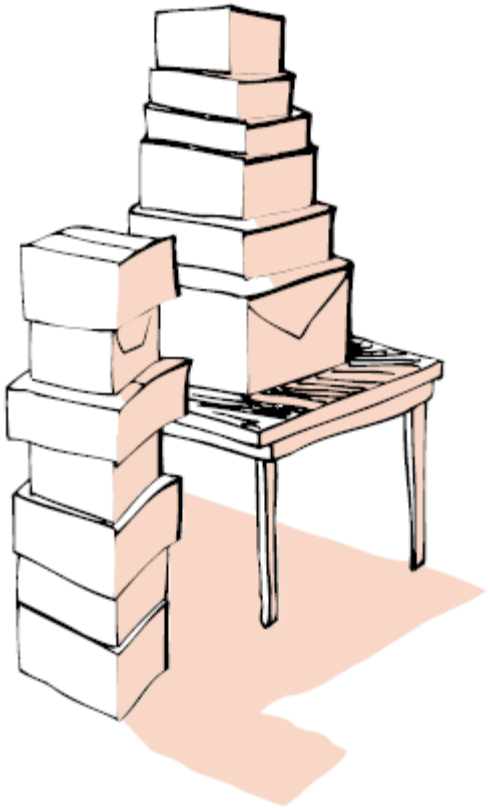
C Programming Basic – week 5

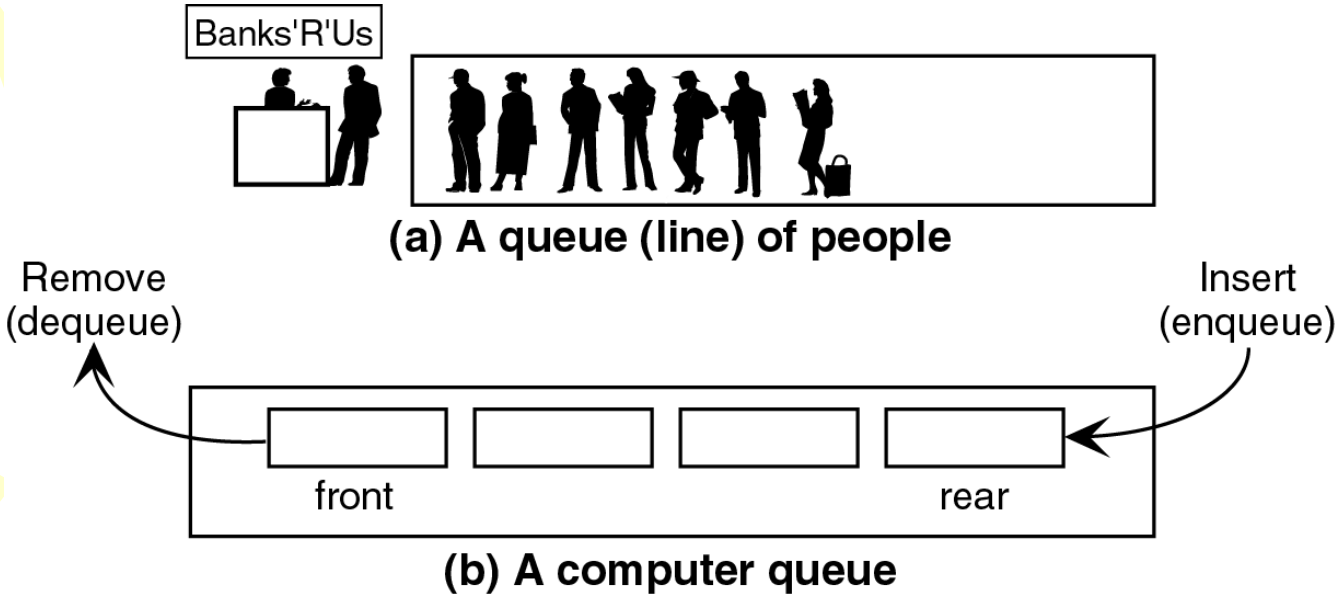
Topics

- Queue
  - Implementation using array
  - Implementation using linked list
- Exercises



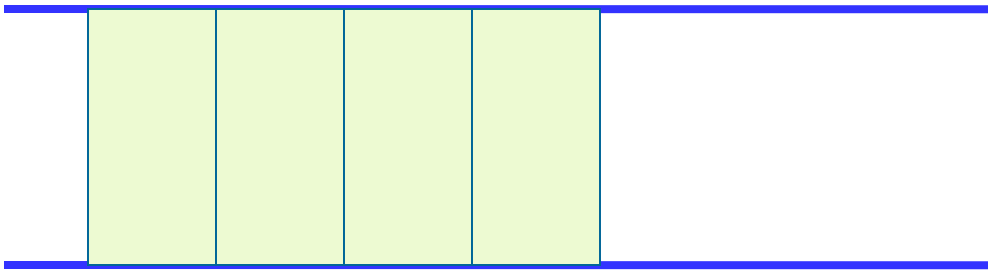
Queue

- 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



front rear

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

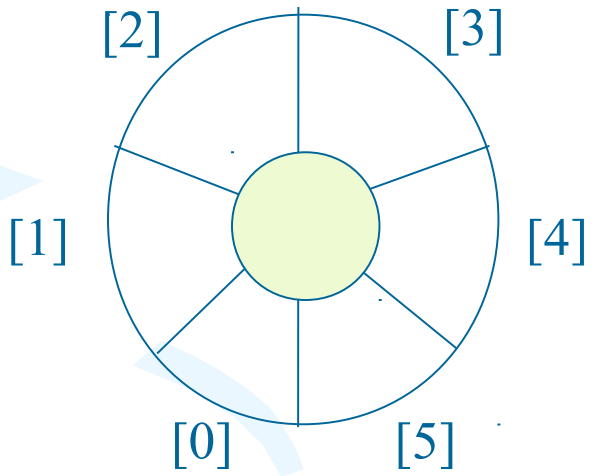
```
void DeQueue(Queue *Q){
  if (!Empty_Queue(*Q)){
     Q->Front=Q->Front+1;
     if (Q->Front > Q->Rear)
  MakeNull_Queue(Q);
     // Queue become empty

}
else printf("Queue is empty!");
}
```

Implementation 2: regard an array as a circular queue

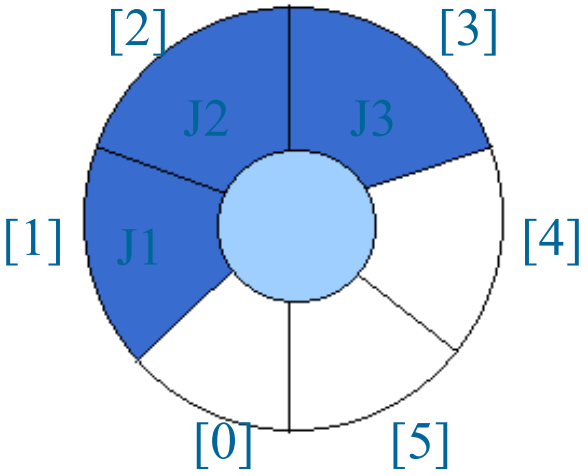
front: one position counterclockwise from the first element

rear: current end

EMPTY QUEUE



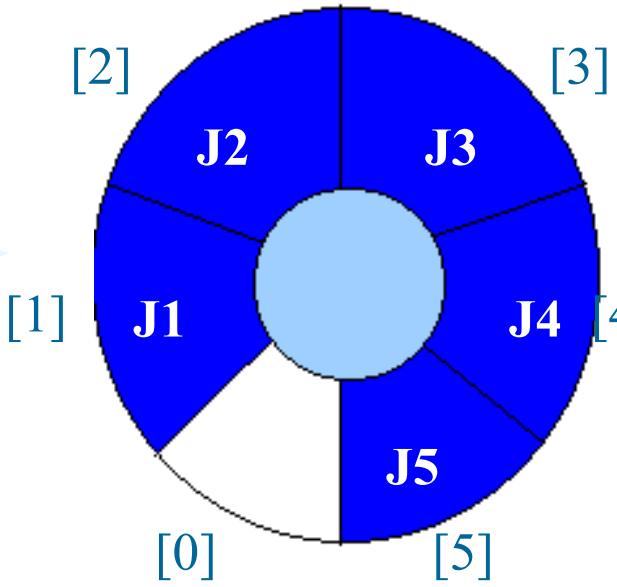
front = 0
rear = 0



front = 0
rear = 3

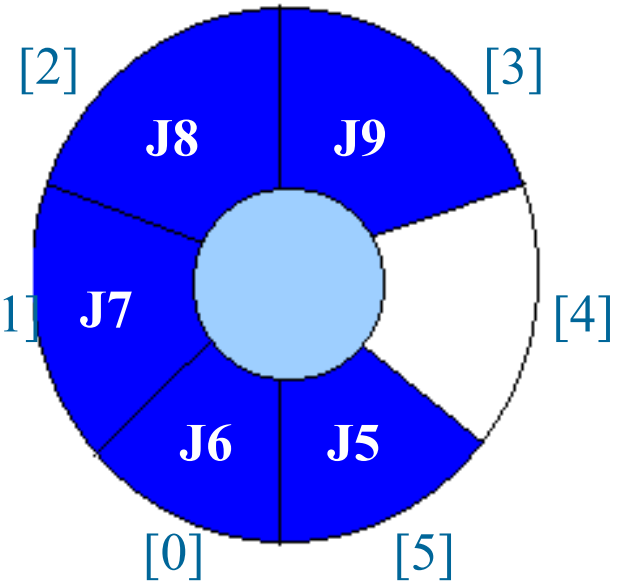
Problem: one space is left when queue is full

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==Q.Rear);
}
```

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;
}
```

DeqQueue

```
void DeQueue(Queue *Q){
if (!Empty_Queue(Q)){
  Position T;
  T=Q->Front;
  Q->Front=Q->Front->Next;
  free(T);
}
else printf("Error: Queue is empty.");
}
```

Exercise 5.1

- 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.

Exercise 5.2

- 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 5.3

- 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.

Exercise 5.4

- 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.

Homework 1

- Bank serves for withdrawal and deposit
- Write a program that serves client by port number (each port is actually a queue)
- Program has menu to add client to the bank
  - Enter time (9 – 9h->10h, 10 – 10-11h)
  - Time for one client is 15min
- Output: Client is put into a port with waiting time
- The program should calculate waiting time of clients
  - Total number of clients
  - Total waiting time
  - Average waiting time per client

Instruction:

- Client information could be in this format in a file:
- Time Number of clients
- 9:00 2
- 9:10 1
- 9:25 3
- 9:40 2

Interface

- BIDV- Hà Thành 17 Tạ Quang Bửu
- Number of queue:2
- 9:00 2
- 1st client goes to Queue 1 : 0
- 2nd client goes to Queue 2 : 0
- 9:10 1
- 1st client goes to Queue 1 : 5 (serve at 9:15)
- 9:25 3
- 1st client goes to Queue 2 : 0
- 2nd client goes to Queue 1: 5
- 3rd client goes to Queue 2: 15
- …

- Average: XYZ clients – total and average waiting time

Another implementation using array

- Queue CreateQ(*max_queue_size*) ::=
  # define MAX_QUEUE_SIZE 100

  typedef struct {
  int key; /* other fields */
  } element;
  element queue[MAX_QUEUE_SIZE];
  int rear = -1;
  int front = -1;
  Boolean IsEmpty(queue) ::= front == rear
  Boolean IsFullQ(queue) ::= rear == MAX_QUEUE_SIZE-1

Enqueue

```
void enq(int *rear, element item)
{
/* add an item to the queue */
   if (*rear == MAX_QUEUE_SIZE_1) {
      queue_full( );
      return;
   }
   queue [++*rear] = item;
}
```

Dequeue

- element deq(int *front, int rear)
  {
  if (*front == rear)
  return queue_empty( );
  /* return an error key */
  return queue [++ *front];
  }

Enqueue

```
void addq(int front, int *rear, element item)
{
    *rear = (*rear +1) % MAX_QUEUE_SIZE;
     if (front == *rear) /* reset rear and print
                                    error */
     return;
   }
     queue[*rear] = item;
}
```

Dequeue

```
element deleteq(int* front, int rear)
{
    element item;

     if (*front == rear)
         return queue_empty( );
        /* queue_empty returns an error key */

    *front = (*front+1) % MAX_QUEUE_SIZE;
     return queue[*front];
}
```