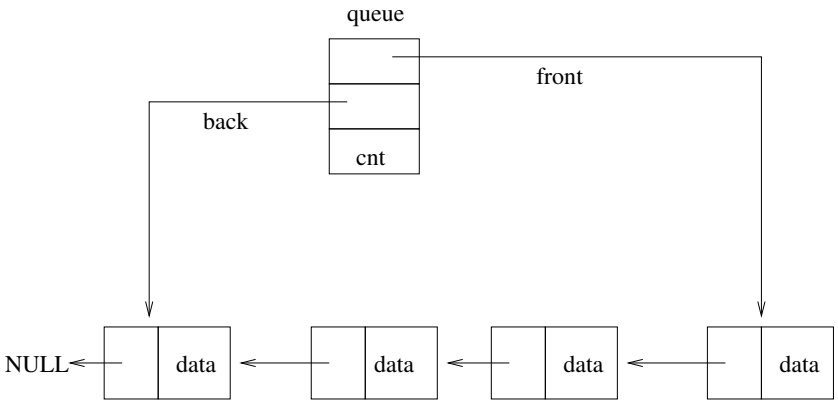
# The ADT queues

A queue is another linear data structure that follows the First-In-First-Out (FIFO) principle. This means the first element added to the queue is the first to be removed, much like a real-world line of people waiting for service at counters (post office, bank ATM, etc). The behaviour of a queue ADT is defined by the following primary operations:

- enqueue: adds a new element to the rear (back) of the queue
- dequeue: removes and returns the element from the front (head) of the queue
- peek (or front): returns the front element without removing it
- empty: checks if the queue contains any element
- full: checks if the queue has reached its maximum capacity (relevant for fixed-size implementations like arrays

We implement a queue as a linear linked list. Each element of a queue is a node that contains data and a pointer to a similar node. These nodes are made into a chain using the pointers. We also have a main component of a queue that consists of two pointers: a back pointer pointing to the back of the queue and a front pointer pointing to the front of the queue. Though not strictly necessary, it is beneficial to include an integer counter to keep track of the number of nodes in the queue. Below is a schematic diagram illustrating a queue implementation. Any new element will be added to the left of the element pointed to by the back pointer. Additionally, the element pointed by the front pointer will be removed first.




We use a queue to illustrate its role in managing printer jobs. Many users send their printing jobs to a public printer. The printer will put them into a queue by arrival time and print the jobs in order. Let us assume that five different users send five different files. We read the job ID, user, and file name from the keyboard. These three components make the data part of the node. Once the queue is formed, we print its elements from the front. The job at the front is of rank 1st, and so on. The file of rank 1st is printed first, then dequeued from the queue. Now the 2nd one in the original queue becomes rank 1st. If any user sends another file for printing, then it will be enqueued at the back of the queue. Below is a code that implements file management on a printer. The code is self-explanatory.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct   // component of a node
{
 char owner[30];
 int job_id;
 char file_name[100];
} data;
typedef struct node            // element of queue
{
 data item;
 struct node *next;        // self-referential
} node;
typedef struct   // definition of queue
{
 node *front,*back;
 int cnt;
} queue;
typedef enum {false, true} boolean;
void initialize(queue *); // reset or initialize queue
void enqueue(queue *,data *);   // add a new element at the back
boolean  empty(queue *);        // boolean function returns true or false
data dequeue(queue *);  // delete an existing element at the front
void printq(queue *);  // printing the elements of queue
void fmt(char *,int);  // auxiliary function for printing
int main()
{
 queue q;
 data d;
 initialize(&q);      // reset queue
 printf("Enter job id (-ve val to stop):");       // input of queue elements
 scanf("%d",&d.job_id);
 while(d.job_id>0)
 {
  printf("Enter owner:");
  scanf("%s",d.owner);
  printf("Enter file name:");
  scanf("%s",d.file_name);
```

```
  enqueue(&q,&d);    // add element at the back
  printf("Enter job id (-ve val to stop):");
  scanf("%d",&d.job_id);
 }
 printf("Original queue:\n");
 printq(&q);     // print elements of queue
 if(!empty(&q))     // if not empty, delete an element
 {
  d=dequeue(&q);
  printf("The dequeued item is the following:\n");
  printf("Rank: 1st Owner: %s Job id: %d  File name: %s\n",d.owner,d.job_id,d.file_nam
  }
 printf("Queue after a single dequeue operation:\n");
 printq(&q);     // print the queue again
 return 0;
}
void initialize(queue *que)
{
 que->cnt=0;
 que->front=NULL;
 que->back=NULL;
}
void enqueue(queue *que, data *d)
{
 node *p;
 p=(node *) calloc(1,sizeof(node)); // allocate space for new element
 p->item=*d;
 p->next=NULL;
 if(que->cnt==0)
 {
  que->cnt +=1;
  que->front=p;
  que->back=p;
 }
 else
 {
  que->cnt +=1;
  que->back->next=p;
  que->back=p;
```

```
 }
}
boolean empty(queue *que)
{
 return (boolean) que->cnt==0;
}
data dequeue(queue *que)
{
 data da;
 node *p;
 p=que->front;
 da=p->item;
 que->front=p->next;
 que->cnt -=1;
 if(que->cnt==0)
   {
     que->back=NULL;
   }
 free(p);   // free allocated space
 return da;   // return data
}
void printq(queue *que)   // print queue
{
 data da;
 node *p;
 int k;
 char str[3];
 if(que->cnt==0)
 {
  printf("Empty queue!\n");
  return;
 } printf("%-8s%-14s%-10s%-20s\n","Rank","Owner","Job id","File name");
 k=0;
 p=que->front;
        while(k<que->cnt)
 {
  da=p->item;
  k++;
  fmt(str,k);
```

```
  printf("%d%-7s",k,str);
  printf("%-14s%-10d%-20s\n",da.owner,da.job_id,da.file_name);
  p=p->next;
 }
}
void fmt(char *ch,int k)
{
  if(k==1)
  {
   strcpy(ch,"st");
  }
  else if(k==2)
  {
   strcpy(ch,"nd");
  }
  else if(k==3)
  {
   strcpy(ch,"rd");
  }
  else
  {
   strcpy(ch,"th");;
  }
}
```

## Listing 78: C Program "queue.c"

Below is a sample input-output. Note that since we use `%s` for reading character strings, these character strings corresponding to the owner and file name should not have blank spaces inside the strings.

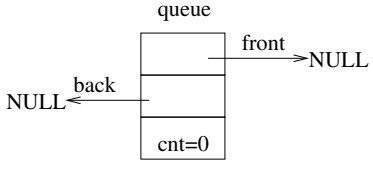
```
Enter job id (-ve val to stop):8
Enter owner:sghorai
Enter file name:numer.pdf
Enter job id (-ve val to stop):12
Enter owner:keshav
Enter file name:banach.pdf
Enter job id (-ve val to stop):17
Enter owner:sjha
Enter file name:number.pdf
```

```
Enter job id (-ve val to stop):20
Enter owner:biswas
Enter file name:m430.tex
Enter job id (-ve val to stop):25
Enter owner:suprio
Enter file name:stoch.dat
Enter job id (-ve val to stop):-2
Original queue:
Rank    Owner         Job id    File name
1st     sghorai       8         numer.pdf
2nd     keshav        12        banach.pdf
3rd     sjha          17        number.pdf
4th     biswas        20        m430.tex
5th     suprio        25        stoch.dat
The dequeued item is the following:
Rank: 1st   Owner: sghorai  Job id: 8   File name: numer.pdf
Queue after a single dequeue operation:
Rank    Owner         Job id    File name
1st     keshav        12        banach.pdf
2nd     sjha          17        number.pdf
3rd     biswas        20        m430.tex
4th     suprio        25        stoch.dat
```

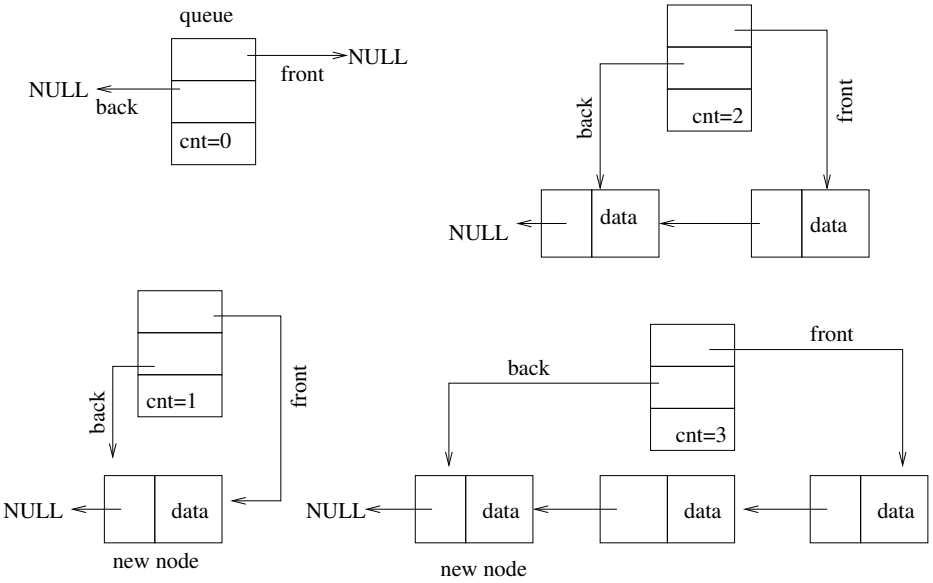
We can write the front function as follows:

```
data front(queue *que)
{
 return que->front->item;
}
```

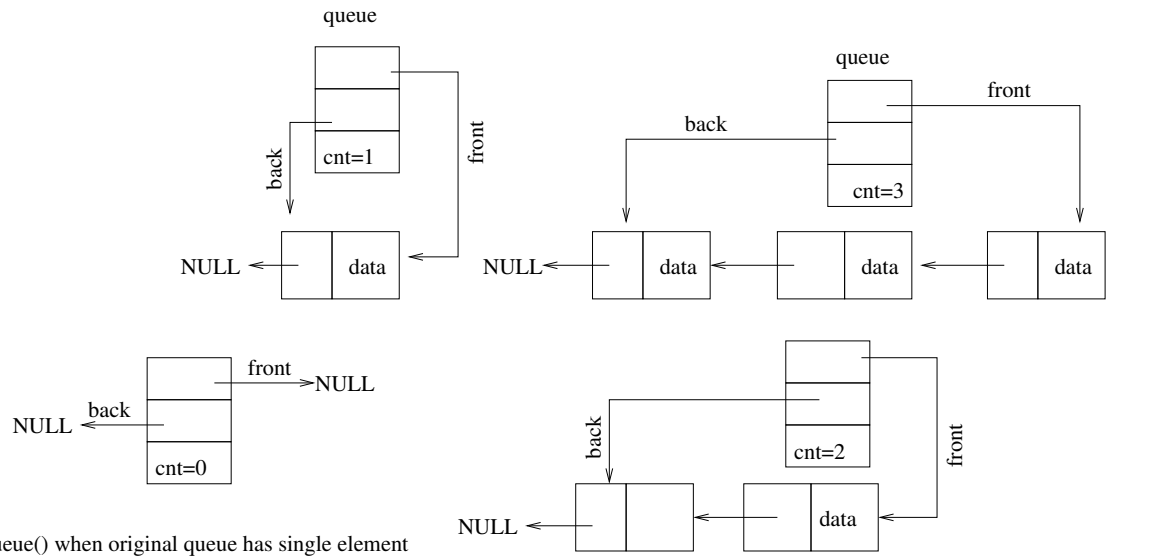
We illustrate the `reset()`, `enqueue()`, and `dequeue()` operations pictorially in the following figures.




(a) reset()




(a) enque() when orginal queue empty (b) enque() when original queue is not empty




(a) dequeue() when original queue has single element

(b) dequeue() when original queue has more than one elements

Return to Lecture Topics