Data Structures

- theory that tells us how to best organise data for easy storage and access

Queue

- FIFO: first in first out
- join a queue at rear end
- leave/depart the queue from the front
- push: add elements at rear end
- pop: deletes element from front end
- by definition, size of queue is unbounded
- Input Buffer works like a queue

```
 __ __ __ __ __ __ __ __
 ^(front)                ^(rear)
```

```
#include<stdlib.h>
#include<stdio.h>

#define SIZE 5

struct queue
{
   int a[SIZE];
   int rear;
};

void push(struct queue *ptr , int a);
int pop(struct queue *ptr);
int is_empty(struct queue *ptr);
int is_full(struct queue *ptr);

int main()
{
   struct queue q;
   q.rear = -1;
   char c;
   int a;
   while(1)
   {
      printf("ENTER YOUR CHOICE (P/p/Q): ");
      scanf(" %c",&c);
      if(c == 'P')
      {
         if(is_full(&q) == 1)
         {
            printf("QUEUE IS FULL\n");
         }
         else
         {
```

```
            scanf("%d",&a);
            push(&q,a);
        }
    }
    else if(c == 'p')
        if(is_empty(&q) == 1)
        {
            printf("QUEUE IS EMPTY\n");
        }
        else
        {
            printf("%d\n" , pop(&q));
        }
    else if(c == 'Q')
    {
        break;
    }
    else
    {
        printf("INVALID INPUT\n");
    }
  }
  return 0;
}

void push(struct queue *ptr , int a)
{
  (ptr->rear)++;
  ptr->a[ptr->rear] = a;
}

int pop(struct queue *ptr)
{
  int t = ptr->a[0];
  for(int i=0; i<ptr->rear; i++)
  {
    ptr->a[i] = ptr->a[i+1];
  }
  ptr->a[ptr->rear] = 0;
  (ptr->rear)--;
  return t;
}

int is_empty(struct queue *ptr)
{
  if(ptr->rear == -1)
    return 1;
  else
    return 0;
}

int is_full(struct queue *ptr)
{
  if(ptr->rear == SIZE - 1)
```

```
        return 1;
    else
        return 0;
}
```