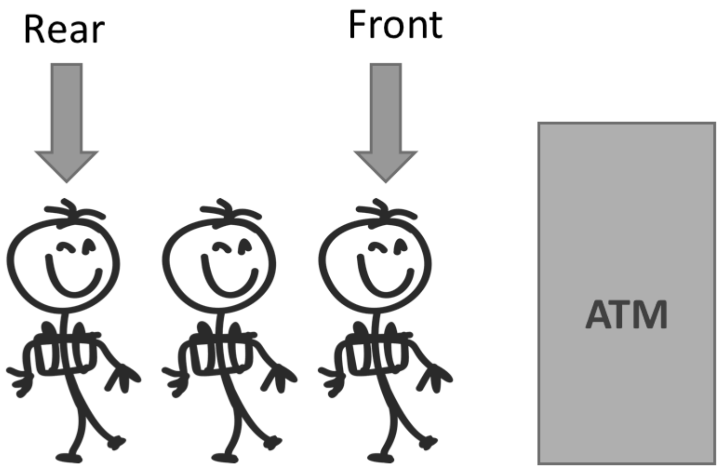
Lab Work 9 Part 2

Learning Objectives

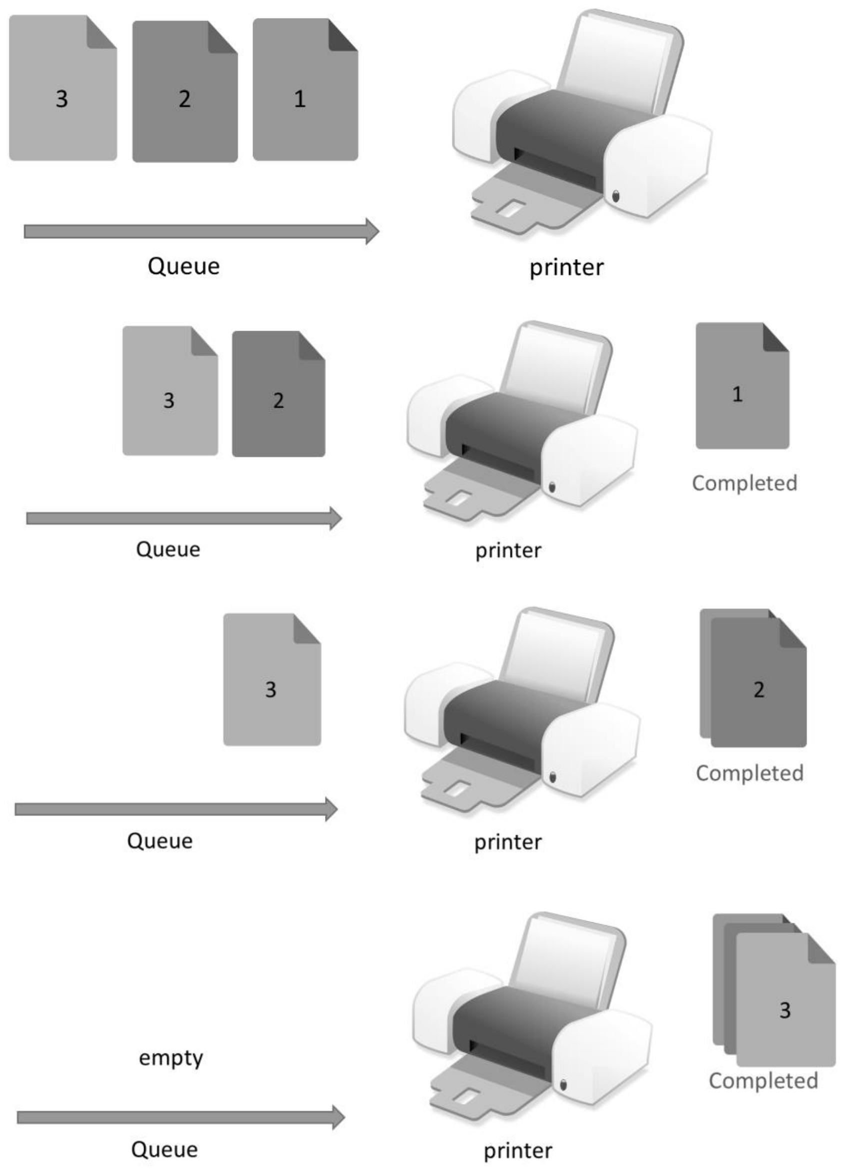
- Queue basics
- Queue using array
- Queue using linked list
- Built in queue

Queue Basics

- Queue is a linear data structure which follows FIFO i.e. First-In-First-Out method.
- That is the data/element which is stored first in the queue will be accessed first.
- The two ends of a queue are called Front and Rear.
- Insertion takes place at the Rear and the elements are accessed or removed from the Front.
- Suppose we take an example of people standing in a queue at an ATM, then the person at front will access the ATM first and then the second person and so on.
- Any new person coming will join the queue at the end i.e rear.



- Applications of queues in computer world :
  1. Scheduling : CPU Scheduling , Job Scheduling.
  2. Buffers : I/O Buffers.
- Example: Printer schedules all the jobs in a queue and prints them one by one.



Operations

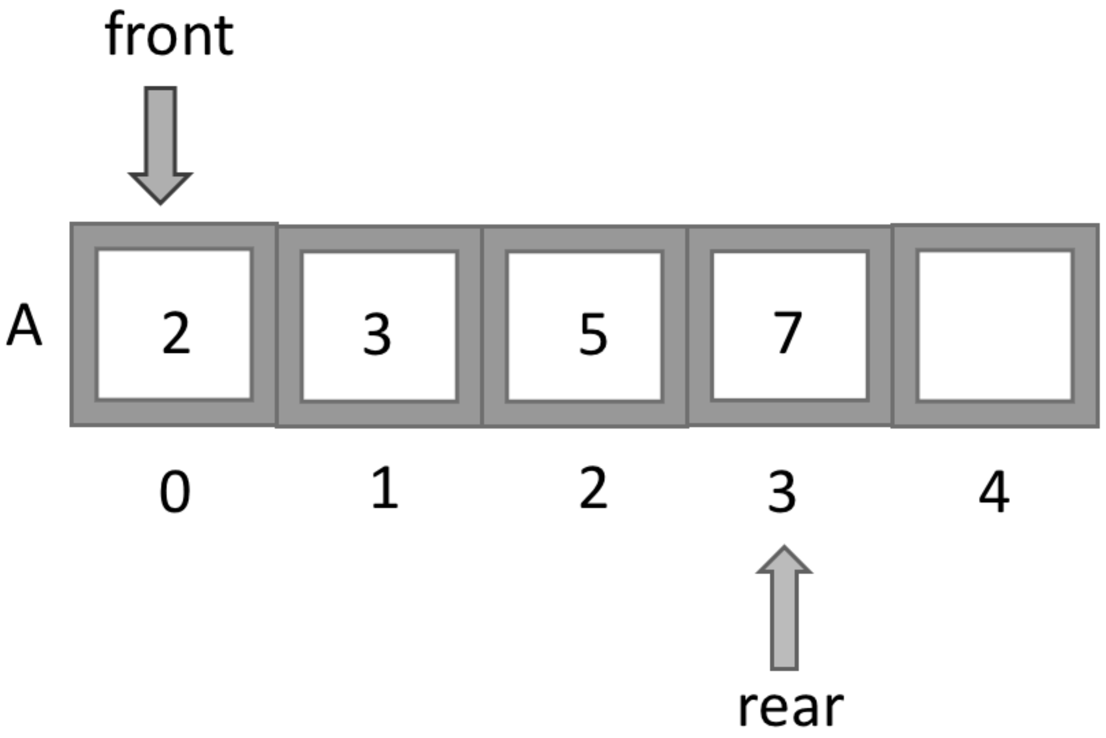
1. **enqueue()** : Insertion of new element in queue.
2. **dequeue()** : Removal of element at front from queue.
3. **showfront()** : To show the element at front.
4. **isempty()** : To check if queue is empty.

Characteristics

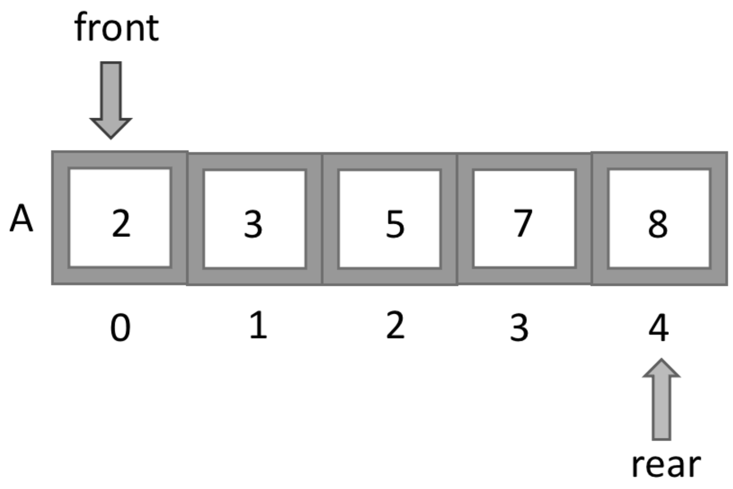
1. Queue is a linear data structure which follows FIFO i.e. First-In-First-Out method.
2. The two ends of a queue are called Front and Rear.
3. Initially when the queue is empty both front and rear are equal to -1.
4. Insertion takes place at the Rear and the elements are accessed or removed from the Front.
5. Applications of queues in computer world :

    1. Scheduling : CPU Scheduling , Job Scheduling.
    2. Buffers : I/O Buffers.

6. Operations
    1. enqueue() : Insertion of new element in queue.
    2. dequeue() : Removal of element at front from queue.
    3. showfront() : To show the element at front.
    4. isempty() : To check if queue is empty.
7. Suppose we have this queue of size 5.



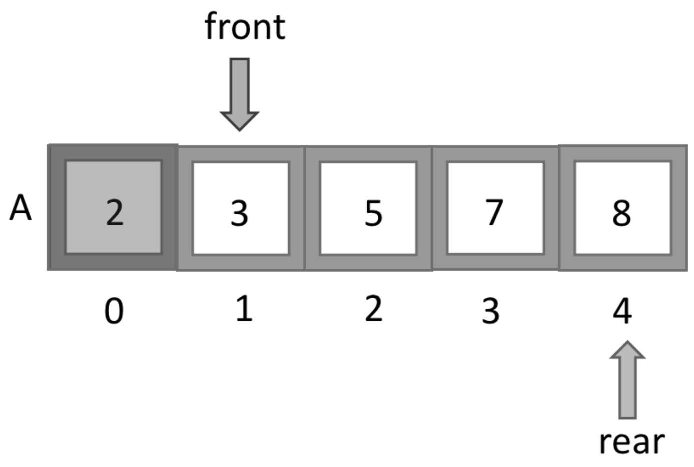
- And we have to insert the element 8 in the queue, so we will increment rear by one and insert 8 at rear.



- While inserting (enqueue) we have to consider two special cases:

1. Queue is empty - then initially front and rear will be -1 so we will increment rear and insert the element at rear and we will make front = 0 i.e. both front and rear will point to the first element.
2. Queue is full - then we cannot insert any more elements in the queue.

- Similarly if we have to remove the first element from the queue, then we will simply increment front by one.



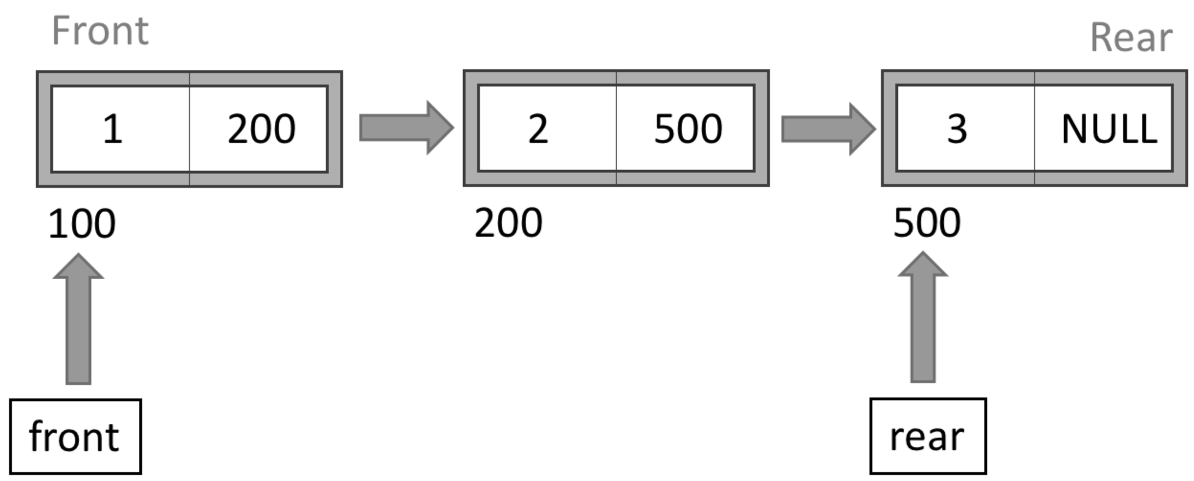
- While removing (dequeue) elements from the queue we also have two special cases:
  1. Queue is empty - in that case front and rear both will be equal to -1 and as there are no elements in the queue we cannot perform dequeue operation.
  2. Queue has only one element - in that case front will be equal to rear and we will make both equal to -1.

Lab Exercise 1: Queue Implementation using array

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

#define SIZE 5
int A[SIZE];
int front = -1;
int rear = -1;
//function to check if queue is empty
bool isempty()
{
 if(front == -1 && rear == -1)
 return true;
 else
 return false;
}
//function to insert element in queue
void enqueue ( int value )
{
 if (rear == SIZE-1)
  cout<<"Queue is full \n";
 else
 {
  if( front == -1)
   front = 0;
  rear++;
  A[rear] = value;
 }
}
//function to remove element from queue
void dequeue ( )
{
 if( isempty() )
  cout<<"Queue is empty\n";
 else
 if( front == rear )
  front = rear = -1;
 else
  front++;
}
//function to display element at front
void showfront( )
{
 if( isempty())
  cout<<"Queue is empty\n";
 else
  cout<<"element at front is:"<<A[front]<<"\n";
}
//function to display elements of the queue
void displayQueue()
{
 if(isempty())
  cout<<"Queue is empty\n";
 else
 {
  for( int i=front ; i<= rear ; i++)
   cout<<A[i]<<" ";
  cout<<"\n";
 }
}

int main()
{
 //inserting elements in queue
 cout<<"Inserting elements in queue\n";
 enqueue(2);
 displayQueue();
 enqueue(3);
 displayQueue();
 enqueue(5);
 displayQueue();
 enqueue(7);
 displayQueue();
 enqueue(8);

 displayQueue();
 // queue is full
 enqueue(9);
 //show element at front
 showfront();

 cout<<"Removing elements from queue\n";
 //removing elements from the queue
 dequeue();
 displayQueue();
 dequeue();
 displayQueue();
 dequeue();
 displayQueue();
 dequeue();
 displayQueue();
 dequeue();

 return 0;
}
```

Queue implementation using linked list

- Queue is a linear data structure which follows FIFO i.e. First-In-First-Out method.
- The two ends of a queue are called Front and Rear, where Insertion always takes place at the Rear and the elements are accessed or removed from the Front.
- While implementing queues using arrays:
    1. We cannot increase the size of array, if we have more elements to insert than the capacity of array.
    2. If we create a very large array, we will be wasting a lot of memory space.
- Therefore if we implement Queue using Linked list we can solve these problems, as in Linked list Nodes are created dynamically as an when required.
- So using a linked list we can create a Queue of variable size and depending on our need we can increase or decrease the size of the queue.
- Thus instead of using the head pointer with Linked List, we will use two pointers, front and rear to keep track of both ends of the list, so that we can perfrom all the operations in O(1).

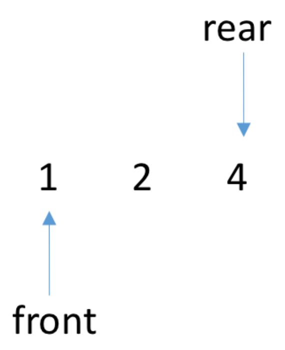


Lab Exercise 2- Queue implementation using linked list

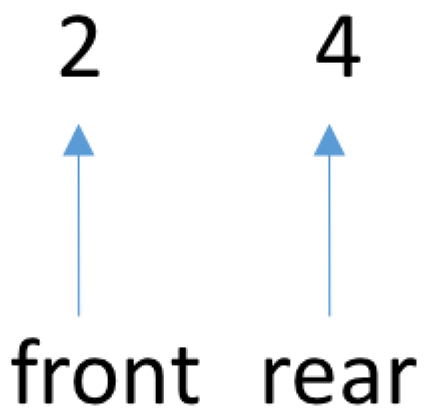
```
#include <iostream>
using namespace std;

// Structure of Node.
struct Node
{
int data;

Node *link;
};

Node *front = NULL;
Node *rear = NULL;

//Function to check if queue is empty or not
bool isempty()
{
 if(front == NULL && rear == NULL)
 return true;
 else
 return false;
}

//function to enter elements in queue
void enqueue ( int value )
{
 Node *ptr = new Node();
 ptr->data= value;
 ptr->link = NULL;

 //If inserting the first element/node
 if( front == NULL )
 {
  front = ptr;
  rear = ptr;
 }
 else
 {
  rear ->link = ptr;
  rear = ptr;
 }
}

//function to delete/remove element from queue
void dequeue ( )
{
 if( isempty() )
 cout<<"Queue is empty\n";
 else
 //only one element/node in queue.
 if( front == rear)
 {
  free(front);
  front = rear = NULL;
 }
 else
 {
  Node *ptr = front;
  front = front->link;
  free(ptr);
 }
}

//function to show the element at front
void showfront( )
{
 if( isempty())
 cout<<"Queue is empty\n";
 else
 cout<<"element at front is:"<<front->data;
}

//function to display queue
void displayQueue()
{
 if (isempty())
  cout<<"Queue is empty\n";
 else
 {
  Node *ptr = front;
  while( ptr !=NULL)
  {
   cout<<ptr->data<<" ";
   ptr= ptr->link;
  }
 }
}

//Main Function
int main()
{
 int choice, flag=1, value;
 while( flag == 1)
 {
  cout<<"\n1.enqueue 2.dequeue 3.showfront 4.displayQueue 5.exit\n";
  cin>>choice;
  switch (choice)
  {
  case 1: cout<<"Enter Value:\n";
          cin>>value;
          enqueue(value);
          break;
  case 2: dequeue();
          break;
  case 3: showfront();
          break;
  case 4: displayQueue();
          break;
  case 5: flag = 0;
          break;
  }
 }

 return 0;
}
```

Built in Queue

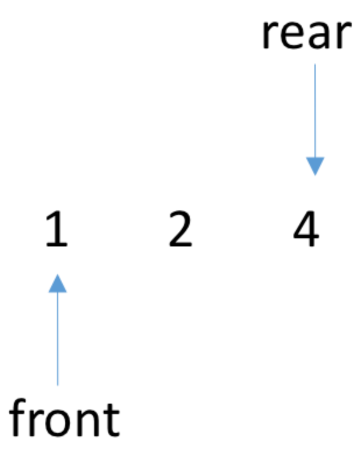
- Queue is a linear data structure which follows FIFO i.e. First-In-First-Out method.
- The two ends of a queue are called Front and Rear, where Insertion always takes place at the Rear and the elements are accessed or removed from the Front.
- To use queue we have to include header file #include <queue>and then we can define a queue of any type using the general format queue <TYPE> name; for example:
    1. queue <int> q1; will create an integer queue called q1
    2. queue <char> q2; will create a character queue called q2
    3. queue <float> q3; will create a float number queue called q3
- Suppose we create a queue q1 of type integer : queue <int> q1;
- **Enqueue** : to perfrom enqueue operation we have the function void push(value) which takes the value to be inserted as argument, example :
    1. q1.push(1); will push the value 1 in the queue q1.
    2. q1.push(2); will push the value 2 in the queue q1.
    3. q1.push(4); will push the value 4 in the queue q1.



- **Show element at front and rear** : The front() function which returns a reference to the element at front and The back() function which returns a reference to the element at rear. example :
    1. cout<< q1.front()<<" "<< q1.back(); will produce the output : 1 4
- **Dequeue** : We will use the void pop() function to remove values from queue. For example :
    1. q1.pop(); will remove the element 1 from the queue.



- **To check if queue is empty or not** : We will use the bool empty() function to check if the queue is empty or not which will return 1 if the queue is empty, otherwise it will return 0.
- **Size** : We will use the size() function which returns the number of elements in the queue. For example :



cout<< q1.size(); For this queue the output will be 3.

Lab Exercise 3: Built in queue

```
#include <iostream>

#include <queue>

using namespace std;


void displayQueue(queue<int> q)

{

 int n = q.size();


 for(int i=0; i<n;i++)

 {


  cout<<q.front()<<" ";

  q.pop();

 }

 cout<<"\n";
```

```
}

int main(){

 queue<int> q1;

 //Enqueue - Pushing elements to queue
 q1.push(1);
 q1.push(2);
 q1.push(4);

 cout<<"Elements of Queue are : ";
 //display Queue
 displayQueue(q1);

 //Show front and rear of queue
 cout<<"Element at front is : "<<q1.front()<<"\n";
 cout<<"Element at rear is : "<<q1.back()<<"\n";

 //Dequeue - removing element from queue
 q1.pop();

 cout<<"Queue after pop operation : ";
 displayQueue(q1);
 //To check if queue is empty or not
 cout<<"Queue is empty or not : "<<q1.empty()<<"\n";

 //size of the queue
 cout<<"Size of the Queue is : "<<q1.size();

 return 0;

}
```