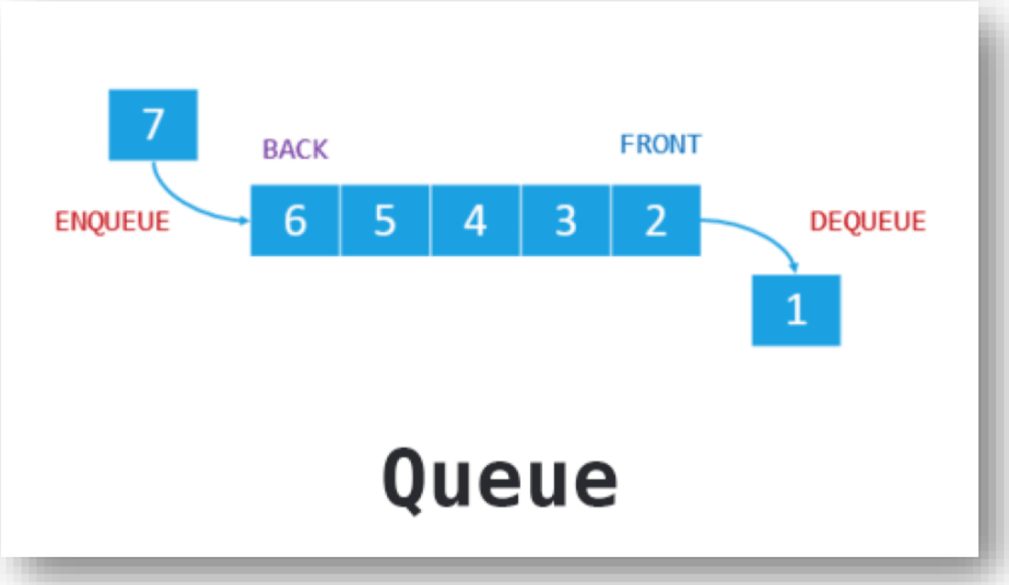
ADVANCED ALGORITHM

Queue data structure



Outline

❑ A Brief of Outline

- What is Queue?
- What are Queue operations?
- How to implement Queue in C++
- Examples

What is Queue?

❑ Definition

- A **queue** is a data structure that stores data in such a way that the element stored first will be retrieved first
- This method is also called **FIFO (First In First Out)**

Real life examples:

- ➢ A queue of customer waiting for payment at counter
- ➢ A queue of vehicles waiting at the petro pump
- ➢ People waiting at the bus store for the bus
- ➢ The first person to enter the queue is the first one to leave the queue
- ➢ Last person to join the queue is the last person to leave the queue

Applications of Queue

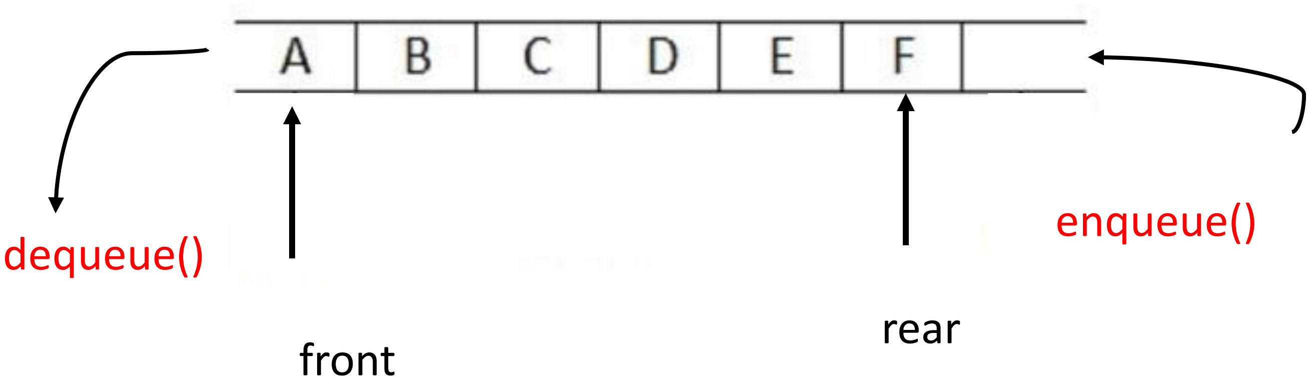
❑ Definition

- Queue finds their use in
  - CPU scheduling,
  - Message queuing,
  - Computer networks
  - etc.
- In time sharing system, queue helps in scheduling of jobs

Queue Operations

❑ Operation

- A **queue** is controlled by two main operations which implement the **FIFO method**
  - ***Insertion***
    - Add element to the queue.
    - This method is called ***enqueue***
  - ***Deletion***
    - Remove element from the queue.
    - This method is called ***dequeue***
- Two variables, FRONT and REAR are used
  - FRONT : used for keep track the first element of the queue
  - REAR : used for keep track the last element of the queue



Queue

Queue Operations

❑ More operations

- **enqueue**: Add element to end of queue
- **dequeue**: Remove element from front of queue
- **isEmpty**: Check if queue is empty
- **isFull**: Check if queue is full
- **peek**: Get the value of the front of queue without removing it

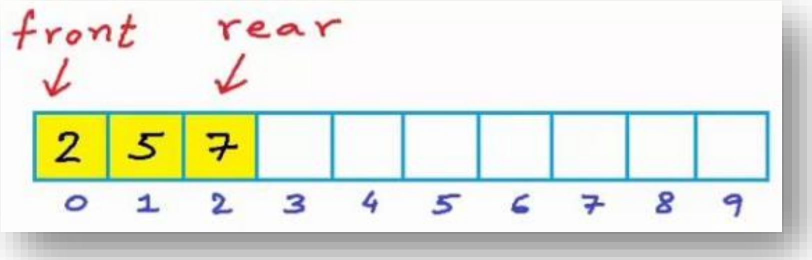
Queue Implementation

❑ Definition

- Queue can be implemented in two ways

1. As an Array

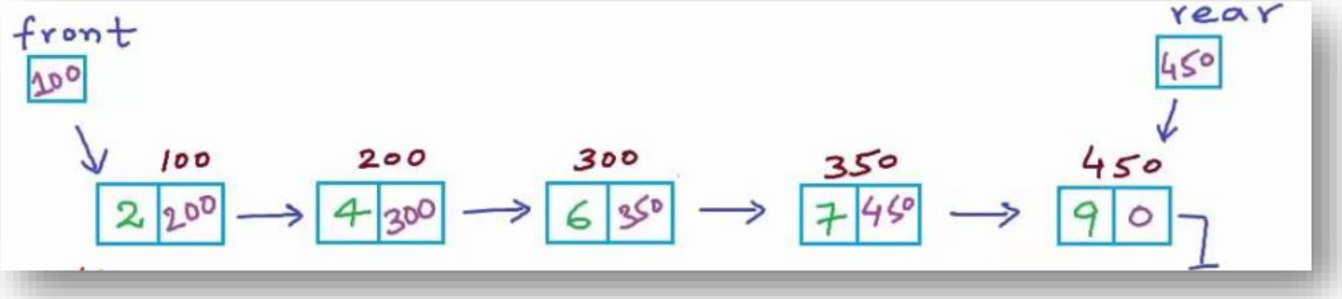
Queue as Array



front variable is used to store the index of the first element
rear variable is used to store the index of the last element

2. As a Linked List

Queue as Linked List



front variable is head of the list
rear variable is tail of the list

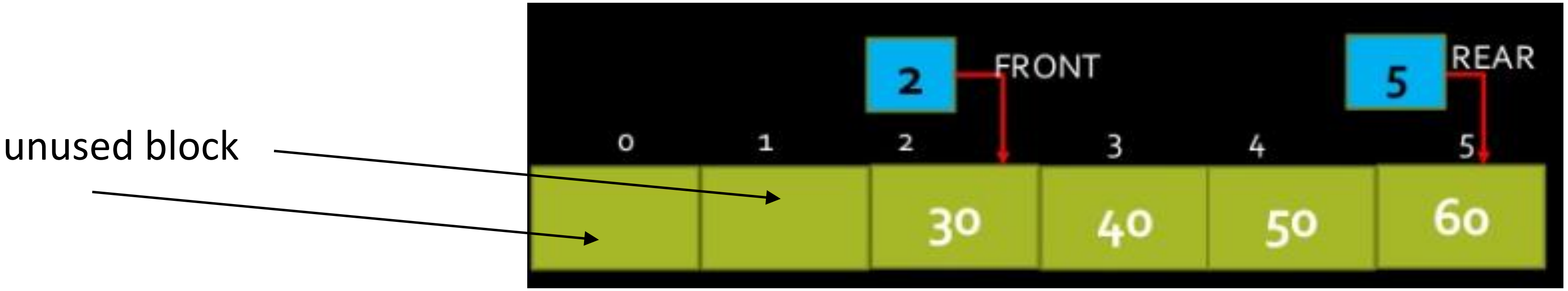
Disadvantage of Queue as Array

❑ Definition

- Implementing queue as an array has one major drawback
  - Since arrays are fixed in size, elements can not be inserted beyond the max size of the array

For example:

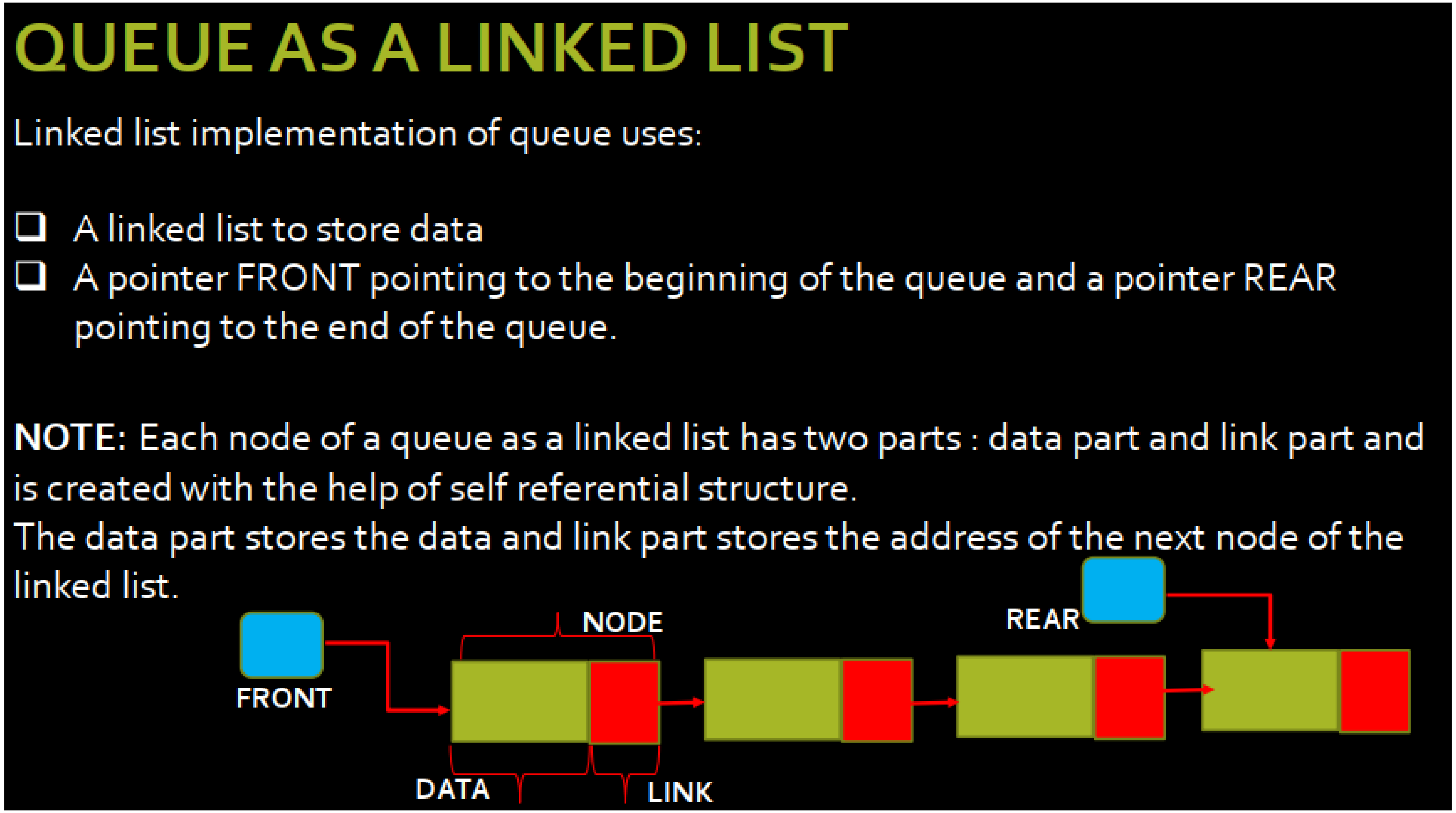
➢ This queue is considered as full although there are two empty spaces in the beginning of the queue



Implementing Queue as Linked List

Queue Implementation

❑ Queue as a Linked List



QUEUE AS A LINKED LIST

Linked list implementation of queue uses:

- ❑ A linked list to store data
- ❑ A pointer FRONT pointing to the beginning of the queue and a pointer REAR pointing to the end of the queue.

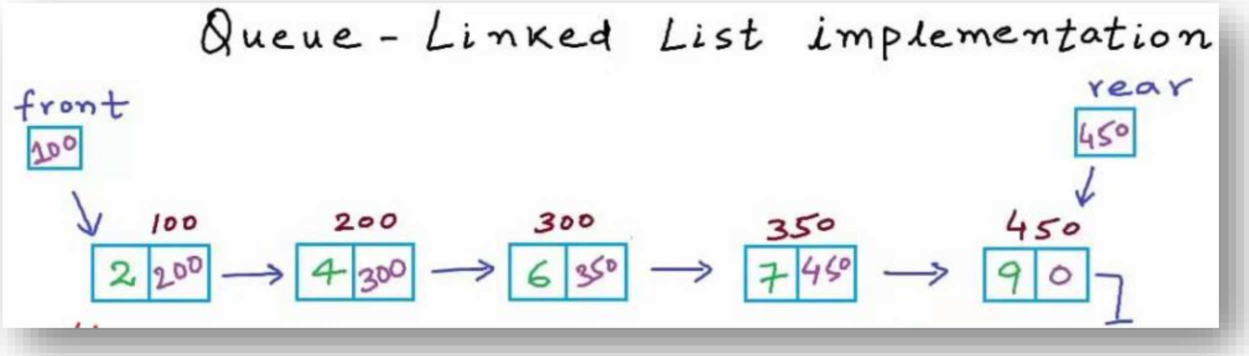
NOTE: Each node of a queue as a linked list has two parts : data part and link part and is created with the help of self referential structure.
The data part stores the data and link part stores the address of the next node of the linked list.

Queue Implementation: Examples

❑ Queue as a Linked List

How to implement this queue?

Demo coding in class



Queue Implementation

❑ Queue as a Linked List

- Implementing queue as a linked list is just like implementing a linked list with some choices

  Choice 1

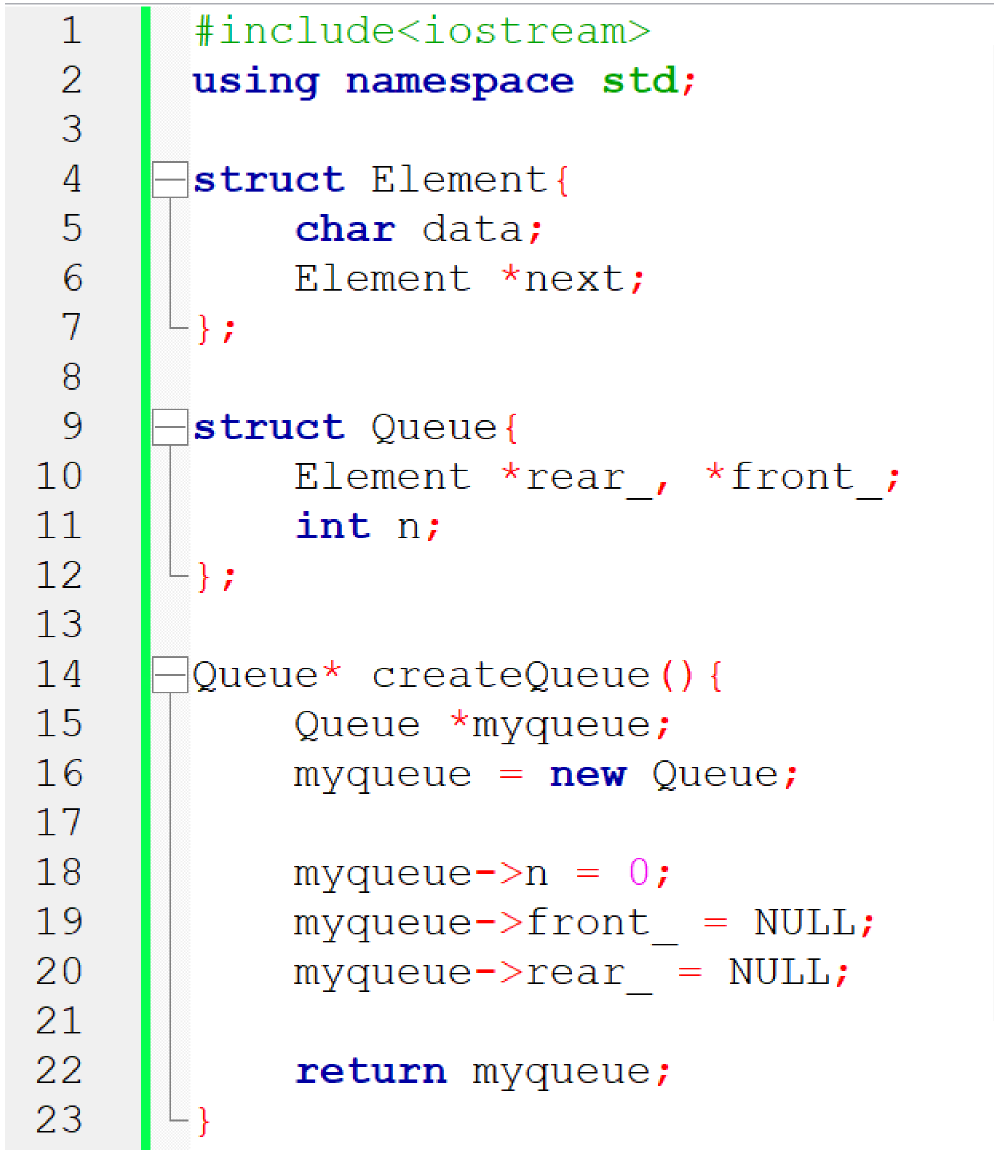
  - Element is added to the end of the list (*enqueue* operation)
  - Element can be only removed from the beginning of the list (*dequeue* operation)

  Choice 2

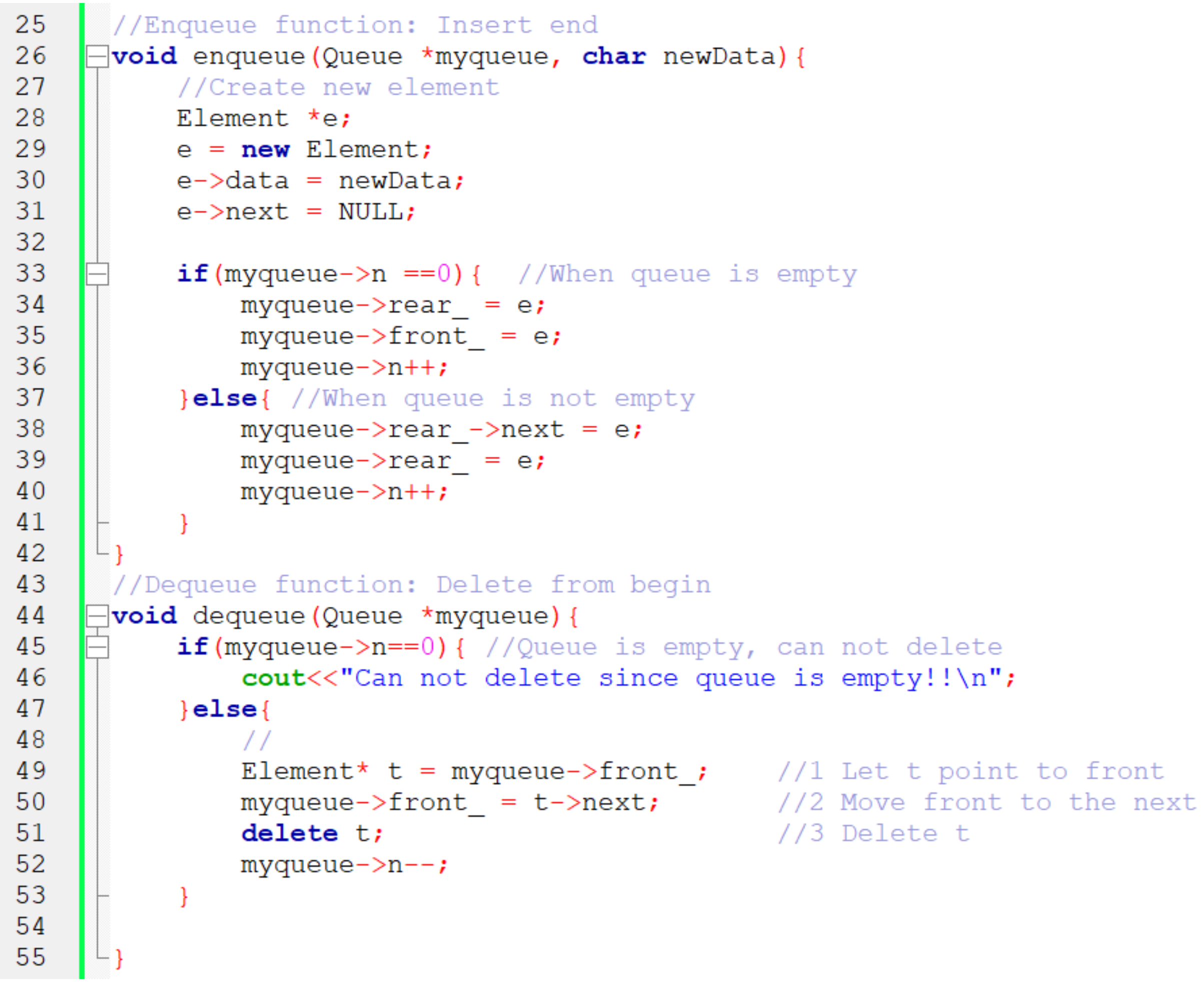
  - Element is added to the beginning of the list (*enqueue* operation)
  - Element can be only removed from the end of the list (*dequeue* operation)

Remark: Choice 1 is recommended.

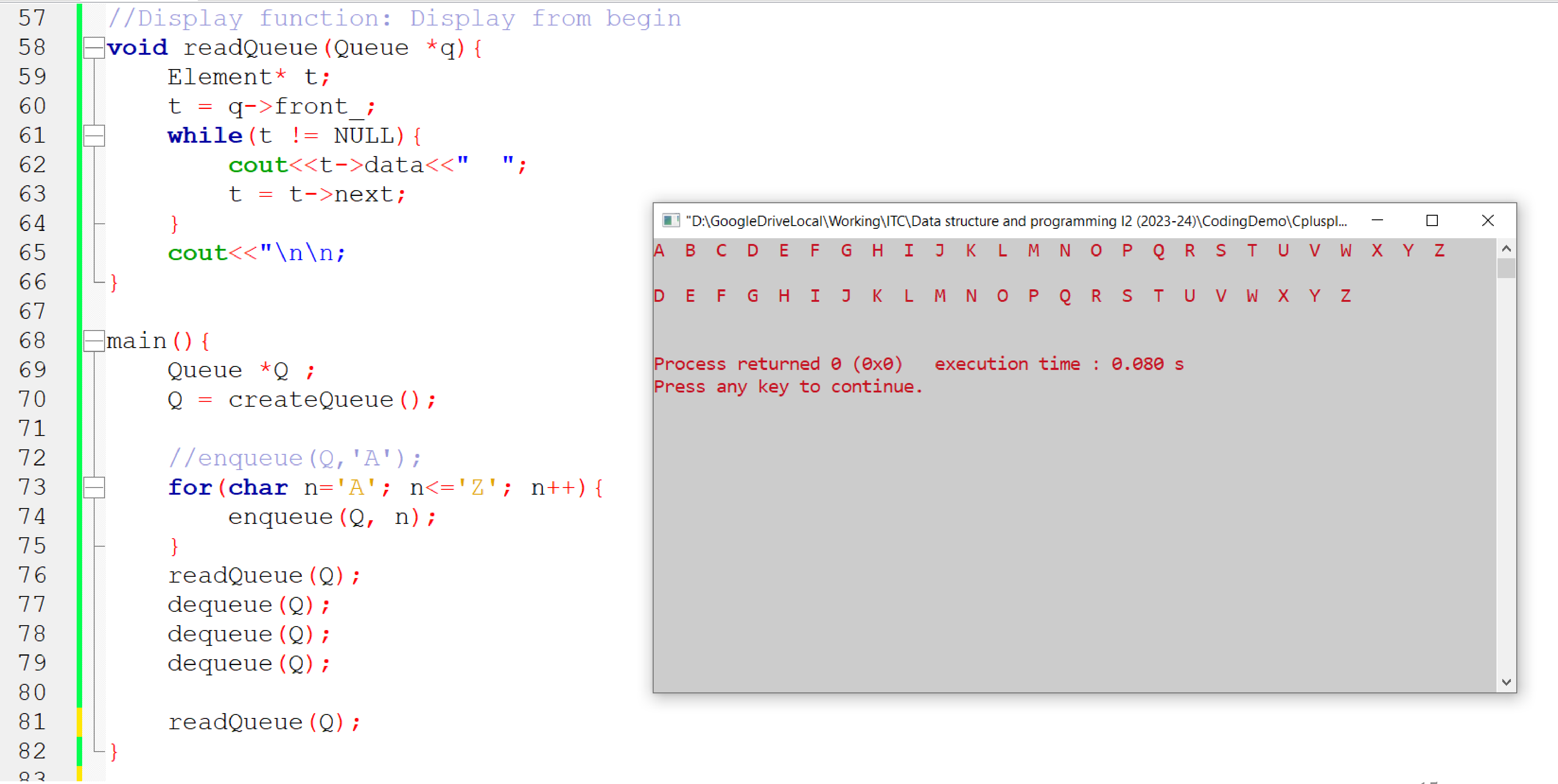
Let’s code it using a structure to implement queue data structure as linked list.



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

struct Element{
    char data;
    Element *next;
};

struct Queue{
    Element *rear_, *front_;
    int n;
};

Queue* createQueue(){
    Queue *myqueue;
    myqueue = new Queue;

    myqueue->n = 0;
    myqueue->front_ = NULL;
    myqueue->rear_ = NULL;

    return myqueue;
}
```



```cpp
//Enqueue function: Insert end
void enqueue(Queue *myqueue, char newData){
    //Create new element
    Element *e;
    e = new Element;
    e->data = newData;
    e->next = NULL;

    if(myqueue->n ==0){   //When queue is empty
        myqueue->rear_ = e;
        myqueue->front_ = e;
        myqueue->n++;
    }else{ //When queue is not empty
        myqueue->rear_->next = e;
        myqueue->rear_ = e;
        myqueue->n++;
    }
}
//Dequeue function: Delete from begin
void dequeue(Queue *myqueue){
    if(myqueue->n==0){ //Queue is empty, can not delete
        cout<<"Can not delete since queue is empty!!\n";
    }else{
        //
        Element* t = myqueue->front_;     //1 Let t point to front
        myqueue->front_ = t->next;        //2 Move front to the next
        delete t;                         //3 Delete t
        myqueue->n--;
    }

}
```



```
//Display function: Display from begin
void readQueue(Queue *q){
    Element* t;
    t = q->front_;
    while(t != NULL){
        cout<<t->data<<"  ";
        t = t->next;
    }
    cout<<"\n\n;
}

main(){
    Queue *Q ;
    Q = createQueue();

    //enqueue(Q,'A');
    for(char n='A'; n<='Z'; n++){
        enqueue(Q, n);
    }
    readQueue(Q);
    dequeue(Q);
    dequeue(Q);
    dequeue(Q);

    readQueue(Q);
}
```

"D:\GoogleDriveLocal\Working\ITC\Data structure and programming I2 (2023-24)\CodingDemo\Cpluspl...

```
A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z

D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z


Process returned 0 (0x0)   execution time : 0.080 s
Press any key to continue.
```

Q and A

Practice

Using queue data structure

Create a queue that stores each letter for an English word input by a user. Then add each letter of this word to this queue. Add to end of the queue and remove from begin.

- Ask another user to input a word then test whether a word stored in this queue is the same.