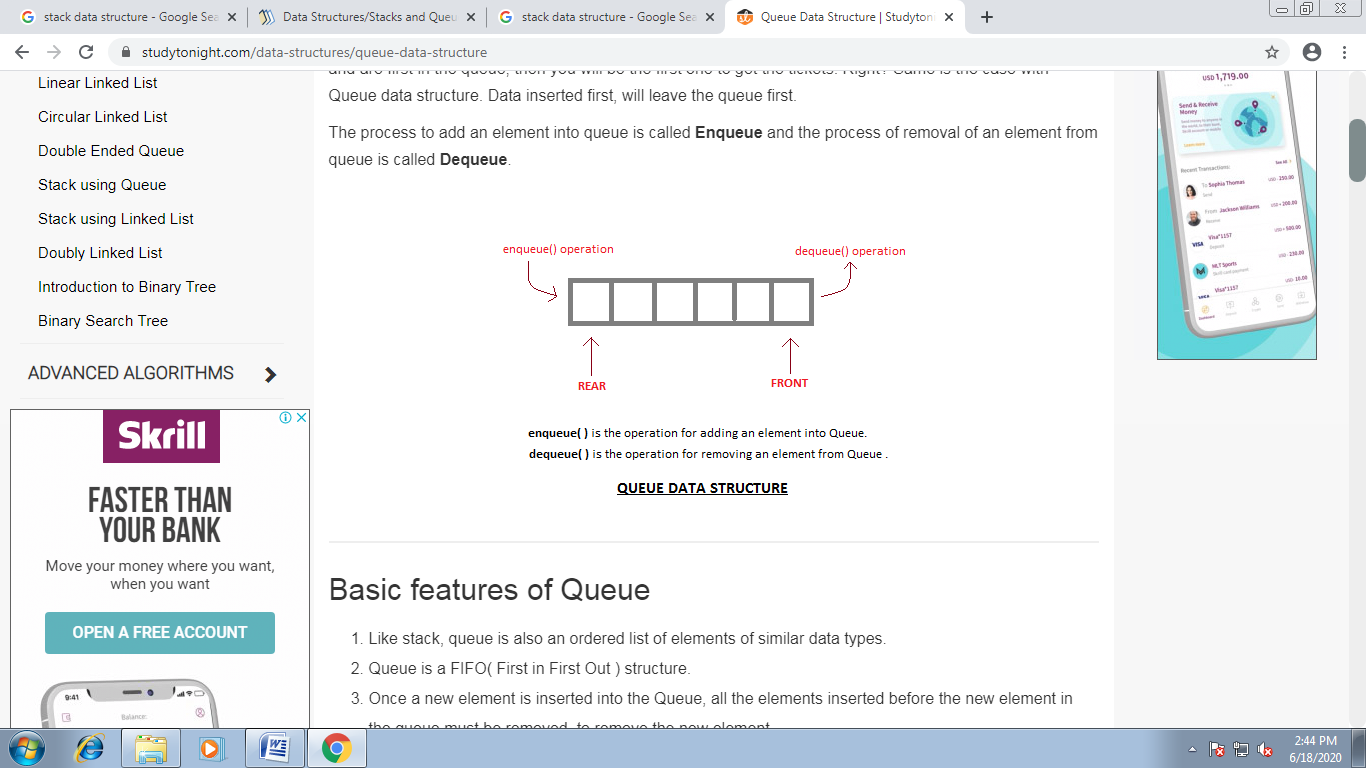
**Queues-** **Queue** is also an abstract data type or a linear data structure, just like stack data structure, in which the first element is inserted from one end called the **REAR**(also called **tail**), and the removal of existing element takes place from the other end called as **FRONT**(also called **head**).

This makes queue as **FIFO**(First in First Out) data structure, which means that element inserted first will be removed first.

Which is exactly how queue system works in real world. If you go to a ticket counter to buy movie tickets, and are first in the queue, then you will be the first one to get the tickets. Same is the case with Queue data structure. Data inserted first, will leave the queue first.

The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.



Basic Operations

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues −

* **enqueue()** − add (store) an item to the queue.
* **dequeue()** − remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are −

* **peek()** − Gets the element at the front of the queue without removing it.
* **isfull()** − Checks if the queue is full.
* **isempty()** − Checks if the queue is empty.

In queue, we always dequeue (or access) data, pointed by **front** pointer and while enqueing (or storing) data in the queue we take help of **rear** pointer.

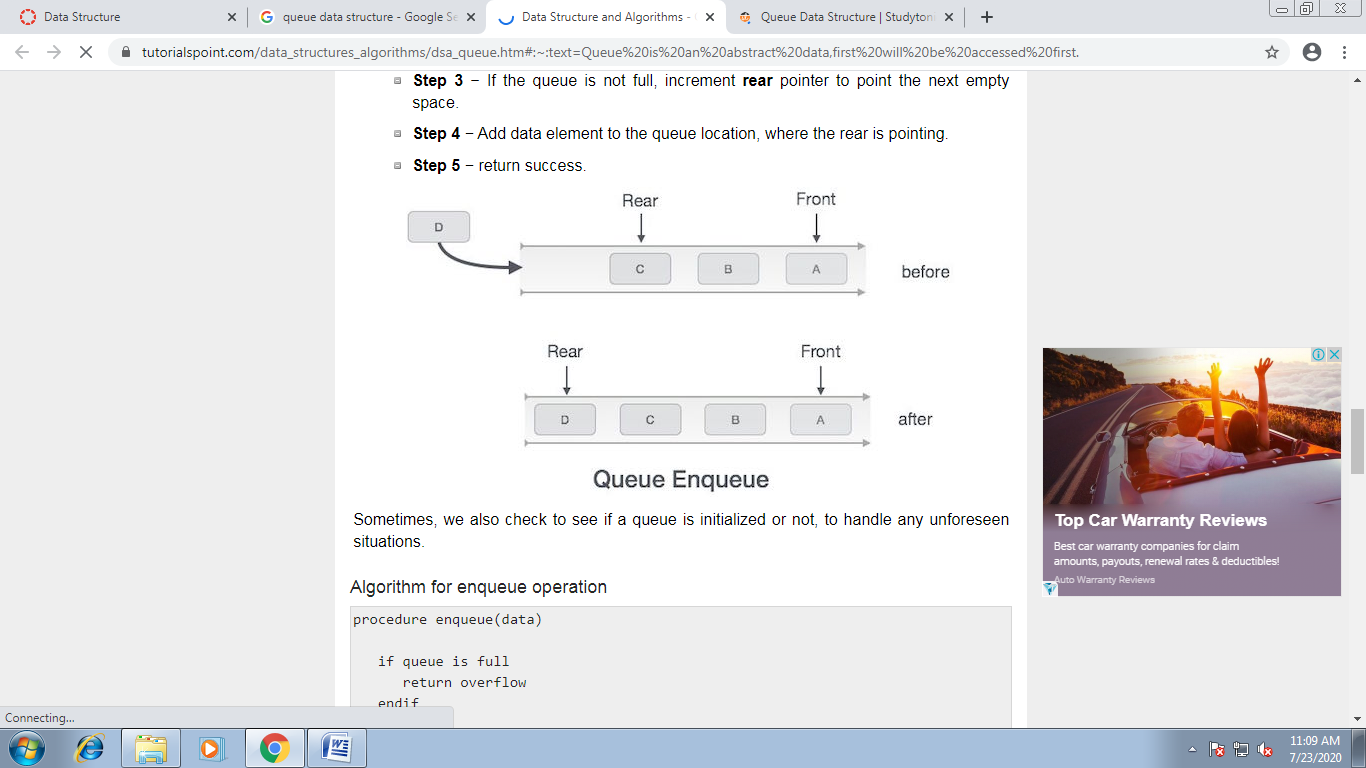
Let's first learn about supportive functions of a queue −

## Enqueue Operation

Queues maintain two data pointers, **front** and **rear**. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue −

* **Step 1** − Check if the queue is full.
* **Step 2** − If the queue is full, produce overflow error and exit.
* **Step 3** − If the queue is not full, increment **rear** pointer (if it is on -1) to point the next empty space.
* **Step 4** − Add data element to the queue location, where the rear is pointing.
* **Step 5** − return success.



Sometimes, we also check to see if a queue is initialized or not, to handle any unforeseen situations.

### Algorithm for enqueue operation

procedure enqueue(data)

if queue is full

return overflow

endif

rear ← rear + 1 //Initially rear == -1

queue[rear] ← data

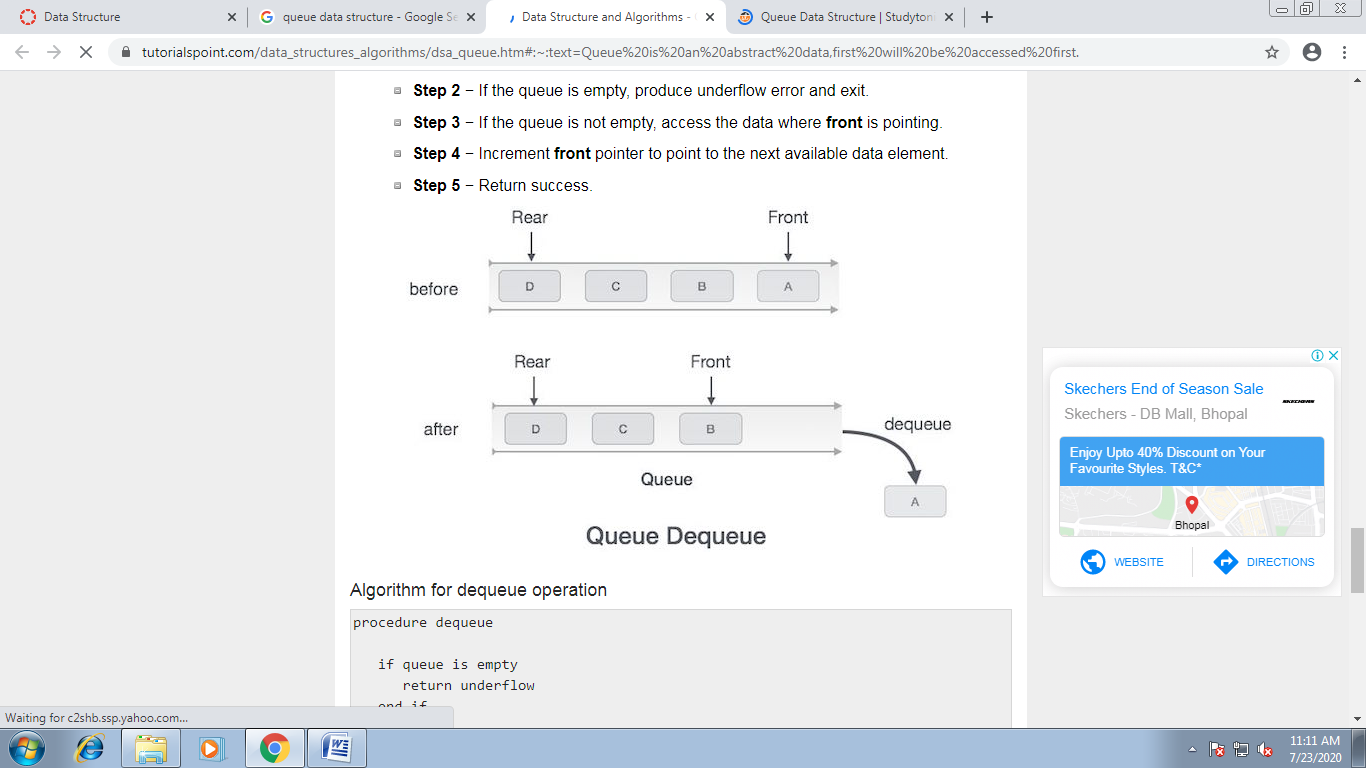
return true

end procedure

## Dequeue Operation

Accessing data from the queue is a process of two tasks − access the data where **front** is pointing and remove the data after access. The following steps are taken to perform **dequeue** operation −

* **Step 1** − Check if the queue is empty.
* **Step 2** − If the queue is empty, produce underflow error and exit.
* **Step 3** − If the queue is not empty, access the data where **front** is pointing.
* **Step 4** − Increment **front** pointer (if it is on -1) to point to the next available data element.
* **Step 5** − Return success.



### Algorithm for dequeue operation

procedure dequeue

if queue is empty

return underflow

end if

data = queue[front]

front ← front + 1

return true

end procedure

#include<stdio.h>

#include<conio.h>

#define size 100

int front = 0;

int rear = 0;

int queue[size];

void insertion();

void deletion();

void display();

void main()

{

clrscr();

while(1)

{

clrscr();

int choice;

printf("\n\n\t\t The program of linear queue through ARRAY");

printf("\n\t\t(MAX Size = 100)\n\n\n\n");

printf("\n\n\t\t1. Insert the element OR");

printf("\n\n\t\t2. Withdraw the element OR");

printf("\n\n\t\t3. Display the contents OR");

printf("\n\n\t\t4. Exit.");

printf("\n\n\n\n\t\tWhich operation you want to perform =");

fflush(stdin);

scanf("%d",&choice);

switch(choice)

{

case 1: insertion();

break;

case 2: deletion();

break;

case 3: display();

break;

case 4: printf("\n\n\n\n\t\tYou have successfully terminated from the program.");

getch();

exit(0);

default:printf("\n\n\n\n\t\tYour choice is not valid.");

printf("\n\n\t\tPress any key to rechoice.......");

getch();

continue;

}

}

}

void insertion()

{

int num;

clrscr();

if(rear==size)

{

printf("\n\n\t\t Array/Queue is FULL .......");

printf("\n\n\t\t Please first delete some element then try again.");

getch();

}

else

{

printf("\n\n\t\tEnter the number to insert = ");

scanf("%d", &num);

queue[rear]=num;

rear++;

printf("\n\n\t\t Your element has been successfully inserted.");

getch();

}

}

void deletion()

{

int num;

clrscr();

if(front==rear)

{

printf("\n\n\t\t There is no element in the queue");

printf("\n\n\t\t Please insert some element and then try again.");

getch();

return;

}

else

{

num =queue[front];

printf("\n\n\t\t Your element %d has been successfully deleted.", num);

front++;

getch();

return;

}

}

void display()

{

int i;

for(i=front;i<rear;i++)

printf("\n\t\t\t%d",queue[i]);

getch();

}

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

struct que

{

int info;

struct que \*next;

};

typedef struct que queue;

queue\* insertion(queue \*front);

queue\* deletion(queue \*front);

void display(queue \*front);

void main()

{

queue \*front;

int choice;

front=NULL;

while(1)

{

clrscr();

printf("\n\n\t\tThats the program of linear queue through LINKED LIST");

printf("\n\n\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("\n\n\t\t\t------------------------------\n");

printf("\n\n\t\t\t1. Insert the element OR");

printf("\n\n\t\t\t2. Withdraw the element OR");

printf("\n\n\t\t\t3. Display the contents OR");

printf("\n\n\t\t\t4. Exit.");

printf("\n\n\t\t\t------------------------------\n");

printf("\n\n\n\n\t\tWhich operation you wanna to perform =");

fflush(stdin);

scanf("%d",&choice);

switch(choice)

{

case 1: front=insertion(front);

break;

case 2: front=deletion(front);

break;

case 3: display(front);

break;

case 4: printf("\n\n\n\n\t\t\*\*You have successfully terminated from the program!!!\*\*");

getch();

exit(0);

default:printf("\n\n\n\n\t\tYour choice is not valid.");

printf("\n\n\t\tPress any key to rechoice.......");

getch();

continue;

}

}

}

queue\* insertion(queue \*front)

{

queue \*tmp,\*tmp2;

clrscr();

tmp=(queue\*)malloc(sizeof(queue));

printf("\n\n\t\tEnter the number to insert = ");

scanf("%d",&tmp->info);

tmp->next=NULL;

if(front==NULL)

{

printf("\n\n\t\t\*\*Your element has been successfully inserted.\*\*");

getch();

return tmp;

}

else

{

for(tmp2=front;tmp2->next!=NULL;tmp2=tmp2->next);

tmp2->next=tmp;

printf("\n\n\t\t\*\*Your element has been successfully inserted.\*\*");

getch();

return front;

}

}

queue\* deletion(queue \*front)

{

queue \*tmp;

int num;

clrscr();

if(front==NULL)

{

printf("\n\n\t\t\*\*There is no element in the queue for deletion.\*\*");

getch();

return front;

}

else

if(front->next==NULL)

{

printf("\n\n\t\t\*\*Your element has been successfully deleted.\*\*");

num=front->info;

printf(“\n\n\t deleted element is %d”,num);

getch();

return NULL;

}

else

{

num=front->info;

printf("\n\n\t\t\*\*Your element %d has been successfully deleted.",num);

getch();

return front->next;

}

}

void display(queue \*front)

{

queue \*tmp;

clrscr();

if(front==NULL)

{

printf("\n\n\t\t\*\*There is no element in the list for display\*\*");

getch();

return;

}

printf("\n\n\t\tYour element are as follows -- \n\n\t\t\t");

for(tmp=front;tmp!=NULL;tmp=tmp->next)

printf("\n\n\t\t\t%d",tmp->info);

getch();

}