**3.STACKS AND QUEUES**

**Stack data structure:**

* A stack is a linear **OR** non-primitive list in which insertion and deletion operations are performed at only one end of the list.
* A stack is **Last in First out (LIFO)** Structure.
* A stack is a linear data structure in which elements are added and remove from one end, called top of stack.
* **For example of stack** is, consider a **stack of plates on the counter in cafeteria**, during the time of dinner, customer take plates from top of the stack and waiter puts the washed plates on the top of the stack. So new plates are put on the top and old one yet bottom.
* Another example is **Railway Shunting System.**
* Application of stack is **Recursion, Stack Machine, Polish notation.**

450 **TOP**

100

300

230

124 **BOTTOM**

**Figure: Stack (vertical representation of stack)**

* The operations on the stack are represented by using vector consisting of number of elements.
* **The insertion operation is referred to as a PUSH** operation and **deletion operation is referred to as a POP operation**.
* The most accessible element in the stack is known as a **TOP** element and the least accessible element in the stack is known as a **BOTTOM** element.
* Since, the insertion and deletion operation are performed at only one end of the stack the element which is inserted last in the stack is first to delete. So stack is also known as ***Last in First out*** **(LIFO)**.

**Operations on Stack:**

1. **Push**: The operations that **add an element to the top of the stack** is called **PUSH** operation. It is used to **insert an element** into the stack.
2. **Pop**: The operation that **delete top element from the top of stack** is called **POP**. it is used to delete an element from stack
3. **Peep:** it is used to **retrieve ith element from the top of the stack.**
4. **Change**: it is used to **change value of the ith element from the top of the stack.**

**Algorithms for Stack Operation**

**(1) Algorithm for PUSH Operation.**

* In push operation, we can add elements to the top of the stack, so before push operation, user must check the stack, it should not be a full.
* If stack is already full and when we try to add the elements then error occurs. It is called **“Stack Over Flow”** error**.**

**PUSH(S, TOP, VAL)**

* This algorithm insert an element X to the top of the stack.
* The Stack is represented by vector S which contains N elements.
* TOP is a pointer which points to the top element of the Stack.
  1. **[Check for stack overflow]**

If TOP >= N then

Write (‘Stack Overflow’)

Return

* 1. **[Increment TOP]**

TOP = TOP + 1

* 1. **[Insert Element]**

S [TOP] = VAL

* 1. **[Finished]**

Return

**(2) Algorithm for POP Operation.**

* In POP operation, we can delete or remove an elements from top of the stack, so before pop operation, user must check the stack, stack should not be a empty.
* If the stack is empty, and we try to pop an element, then error occur. It is called **“Stack under Flow”** error.

**POP(S, TOP)**

* This algorithm removes an element from the Top of the Stack.
* The Stack is represented by vector S which contains N elements.
* TOP is a pointer which points to the top element of the Stack.
  1. **[Check for the Underflow on the Stack]**

If TOP = 0 then

Write (‘STACK UNDERFLOW’)

Exit

* 1. **[Decrement Pointer]**

TOP = TOP - 1

* 1. **[Return former top element of the stack]**

Return(S [TOP + 1])

**(3) Algorithm for PEEP Operation.**

**PEEP(S, TOP, I)**

* This algorithm returns the value of the ith element from the Top of the Stack.
* The Stack is represented by vector S which contains N elements.
* TOP is a pointer which points to the top element of the Stack.
  1. **[Check for the Underflow on the Stack]**

If TOP – i + 1 ≤ 0 then

Write (‘STACK UNDERFLOW’)

Exit

* 1. **[Assign the ith element from the TOP of the Stack]**

X🡨 S [TOP – i+ 1]

* 1. **[Return value]**

Return (X)

**(4) Algorithm for CHANGE Operation.**

**CHANGE(S, TOP, VAL, i)**

* This algorithm changes the value of the ith element from the Top of the Stack by X.
* The Stack is represented by vector S which contains N elements.
* TOP is a pointer which points to the top element of the Stack.
  1. **[Check for the Underflow on the Stack]**

If TOP – i+ 1 ≤ 0 then

Write (‘STACK UNDERFLOW’)

Exit

* 1. **[Change ith  element from the TOP of the Stack]**

S [TOP – i+ 1] 🡨 VAL

* 1. **[finished]**

Return

**Operation:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PUSH 100(top=1)**   |  | | --- | | 100 | | **TOP** | |  | |  | |  | | **PUSH 200(top=2)**   |  | | --- | | 200 | | 100 | |  | |  | |  | | **PUSH 300 (top=3)**   |  | | --- | | 300 | | 200 | | 100 | |  | |  | | **POP(top=2)**   |  | | --- | | 200 | | 100 | |  | |  | |  | | **POP(top=1)**   |  | | --- | | 100 | |  | |  | |  | |  | | **PUSH 1000(top=2)**   |  | | --- | | 1000 | | 100 | |  | |  | |  | |

**Application of Stack**

There are three main application of stack:

**(1) Recursion:**

* **Recursion** means function call itself.
* Itis the technique of defining a process in terms of itself.
* Stack is widely used in recursion because of its Last in First out Property.
* In terms of C language the function calling itself is known as recursion.
* There are two types of recursion:

(1) Primitive Recursive function: Example Factorial Function

(2) Non primitive Recursive function: Example Ackerman’s function **(not in your syllabus).**

* **Primitive Recursive function:**
* Primitive Recursive function is known as recursively defined function.
* For Example

The factorial function is defined as:

1 if N = 0

FACTORIAL (N) =

N \* FACTORIAL (N-1) otherwise

* **Non primitive Recursive function:**
* Non primitive Recursive function is known as recursive use of the function.
* For Example

The Ackerman’s function is defined as:

N + 1, if M = 0

A (M, N) = A (M-1, 1), if N=0

A (M-1, A (M, N-1)), otherwise

**(2)** **Evaluate polish notation:**

* There are basically three types of polish notation:

1. **Infix (B) Prefix (C) Postfix**

* When the operator**(+)** exists between two operands**(A & B)** then it is known as Infix notation.**(e.g. A+B)**
* When the operator**(+)** are written before their operands**(A & B)** then it is known as Prefix notation**(Polish notation)**. **(e.g. +AB)**
* When the operator**(+)** are written after their operands**(A & B)** then it is known as Postfix notation**(Reverse polish notation)**. **(e.g. AB+).**
* Stack is widely used to convert infix notation into prefix or postfix notation.
* One of the major uses of stack is a polish notation or polish expression.
* The process of writing the operators of an expression either before their operands or after operands are called the polish notation.

**Rules for converting infix notation to prefix/ postfix**

1. The operations with heights precedence are converted first and then after a portion of the expression have been converted to postfix.
2. It is to be treated as single operand.
3. Take only two operands at a time to convert in a postfix from like A+B🡪 AB+
4. Always, convert the parenthesis first
5. Convert postfix exponentiation second if there are more than one exponentiation in sequence then take order from right to left.
6. Convert in to postfix as multiplication and division operator, left to right.
7. Convert in postfix as addition and subtraction left to right.

**Example:**

* **Convert infix into POLISH notation (Prefix).**

|  |  |  |
| --- | --- | --- |
| 1. **(A + B) \* C**   = (+ A B) \* C  = \* (+ A B) C  = \* + A B C | 1. **(A + B ) \* (C + D)**   = (+ A B) \* (+ C D)  =\*(+ A B) (+ C D)  =\* + AB + CD | 1. **(A-B)+C\*A+B**   =-AB+C\*A+B  =-AB+\*CA+B  =+-AB\*CA+B  =++-AB\*CAB |
| 1. **(B\*C/D)/(D/C+E)**   =(\*BC/D)/(D/C+E)  =(/\*BCD)/(D/C+E)  =(/\*BCD)/(/DC+E)  =(/\*BCD)/(+/DCE)  =//\*BCD+/DCE | 1. **A\*B/(C+D-E)**   =A\*B/(+CD-E)  =A\*B/(-+CDE)  =\*AB/-+CDE  =/\*AB-+CDE |  |

* **Consider the following example**

**a + b / c –d \* e where a=10, b=6, c=2, d=8, e=13 It is evaluated as follow**

Step: 1 10 + 6 / 2 – 8 \* 13

Step: 2 10 + 3 – 8 \* 13

Step: 3 10 + 3 - 108

Step: 4 13 - 104

Step: 5 - 91

**Example: converts following arithmetic expression into REVERSE POLISH notation (Postfix).**

|  |  |  |
| --- | --- | --- |
| 1. **A \* B / C +D**   = [AB\*]/C+D  = [AB\*C/] +D  =AB\*C/D+ | 1. **C\*B\*A/D**   = [CB\*]\*A/D  = [CB\*A\*]/D  =CB\*A\*D/ | 1. **(A-B)+C\*A+B**   = [AB-] +C\*A+B  = [AB-] + [CA\*] +B  = [AB-CA\*+] +B  =AB-CA\*+B+ |
| 1. **(B\*C/D)/(D/C+E)**   = ([BC\*]/D)/(DC/+E)  = [BC\*D/]/[DC/E+]  =BC\*D/DC/E+/ | 1. **A\*B+C\*D+E\*F**   =[AB\*]+C\*D+E\*F  =[AB\*]+[CD\*]+E\*F  =[AB\*]+[CD\*]+[EF\*]  =[AB\*CD\*+]+[EF\*]  =AB\*CD\*+EF\*+ | 1. **A\*B+A\*(B\*D+E\*F)**   =A\*B+A\*([BD\*]+[EF\*]) =A\*B+A\*([BD\*EF\*+]) =[AB\*]+A\*[BD\*EF\*+]  =[AB\*]+[ABD\*EF\*+\*] =AB\*ABD\*EF\*+\*+ |
| 1. **A\*B/(C+D-E)**   =A\*B/([CD+]-E)  =A\*B/[CD+E-]  =[AB\*]/[CD+E-]  =AB\*CD+E-/ | 1. **A\*A-B\*B**   =[AA\*]-B\*B  =[AA\*]-[BB\*]  =AA\*BB\*- |  |

**(3) Stack Machine:**

* Stack Machines are also used to evaluate polish notations.
* Two example of the stack machine are PDP-11 and B5000. These machines are well suited for stacking local variables and parameter that arise in the function call.
* Stack machines provides faster execution of polish notation.

**IMPORTANT QUESTIONS(Asked question in the GTU Exam)**

1. Discuss Stack characteristics and Write PUSH and POP algorithms.
2. Write Infix, Prefix, Postfix notations with a suitable example. Also mention Operator precedence associated with it.
3. What is Stack? Explain Stack operations.
4. Convert following expression into the postfix notation:

(i) a+b\*(c/d)-e

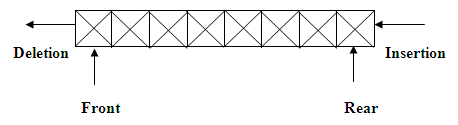
(ii) (a\*b) \* (c^(d+e)-f)

1. List application of stack? Explain any one in detail? Convert x\*(c+d)+(j+k)\*n+m\*p into postfix expression.
2. Give trace of conversion process of following infix string to reverse polish expression. a+b\*c-d/e\*f.
3. (i) Convert following algebraic expression into reverse polish notation:

(a) (a\*b+c)+(d\*e/b) (b) (a+(b\*c-(m/n$p)\*t)\*s

**Queue**

* A queue is a linear list in which insertion is performed at one end called rear end and deletion is performed at another end of the list called front end.
* The information in such a list is proceeds in the same order as it was received.
* Since insertion is performed at one end and deletion is performed at another end the element which is inserted first is first to delete. So it is also known as ***First in First out (FIFO)*** or ***First Come First Serve (FCFS)*** data structure.



**Examples of Queue**

* 1. A raw of students at registration counter
  2. The bullet in a machine gun.(you cannot fire 2 bullets at the same time)
  3. Line of cars waiting to proceeds in some direction at traffic signal

**Application of Queue**

* A queue is the natural data structure for a system to serve its incoming requests. Most of the process scheduling or disk scheduling algorithms in operating systems use queues.
* Computer hardware like a processor or a network card also maintain buffers in the form of queues for incoming resource requests. A stack-like data structure causes starvation of the first requests, and is not applicable in such cases.
* A mailbox or port to save messages to communicate between two users or processes in a system is essentially a queue-like structure.
* Queue is widely used in Simulation.

**Algorithms for Queue operations**

**(1) Algorithm to insert an element in queue**

**QINSERT (Q, Front, Rear, N, Val)**

* This function insert an element into the queue
* The Queue is represented by vector Q which contains N elements.
* Front is a pointer which points to the front end
* Rear is a pointer which points to the rear end
* Y is the element to be inserted.

**1. [Overflow?]**

If Rear≥N then

Write (‘Queue Overflow’)

Exit

**2. [Increment rear pointer]**

Rear🡨Rear+1

**3. [Insert element]**

Q[Rear]🡨 Val

**4. [Is front pointer properly set?]**

If Front=0 then

Front🡨1

Return

**[Finished]**

Exit

**(2) Algorithm to delete an element from the queue**

**QDELETE (Q, Front, Rear)**

* The Queue is represented by vector Q which contains N elements.
* Front is a pointer which points to the front end
* Rear is a pointer which points to the rear end

**1. [Underflow?]**

If Front = 0 then

Write (‘Queue Underflow’)

Exit

1. **[Delete element]**

X🡨Q [F]

1. **[Queue empty?]**

If Front =Rear

Then Front🡨Rear🡨0

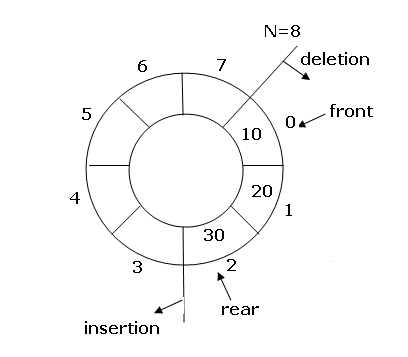
Else Front🡨Front+1

1. **[Return element]**

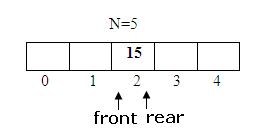
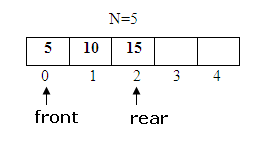
Return (X)

**Circular Queue**

* A circular queue is a queue in which elements are arranged such that the first element in the queue follows the last element in the queue.
* The arrangement of circular queue is shown in figure below:



* Here in the circular queue the first element q[0] follows the last element q[n-1].
* **Disadvantage of simple queue** is that even if we have a free memory space in a queue we cannot use that free memory space to insert element.
* For example consider following operations:



* As shown in figure we insert three elements 5, 10, 15 in simple queue. After that we delete 5 and 10 as shown in figure. Now even we have a free memory space we cannot use that memory space. So simple queue results in wastage of memory space. This problem can be solved by using circular queue.
* For example consider the following operations:

|  |  |
| --- | --- |
|  |  |

* As shown in 1st figure, we insert eight elements 10, 20,30,40,50,60,70,80 in simple queue. After that we delete 10, 20 and 30 as shown in 2nd figure. Now we have a free memory space in circular queue and we can use that memory space by incrementing rear pointer by 1(rear=0).

**Algorithms for Circular queue Operations**

* 1. **Algorithm to insert element in circular queue.**

**CQINSERT (Q, Front, Rear, N, Val)**

* This function inserts an element in to circular queue.
* The Queue is represented by vector Q which contains N elements.
* Front is a pointer which points to the front end
* Rear is a pointer which points to the rear end
* Val is the element to be inserted.

1. **[Reset rear pointer?]**

If Rear = N-1 then

Rear🡨1

Else

Rear🡨Rear + 1

1. **[Check Overflow]**

If Front=Rear then

Write (“Queue Overflow”)

Exit

1. **[Insert element]**

Q[Rear]🡨Val

1. **[Is front pointer properly set?]**

If Front = -1 then

Front🡨0

Return

* 1. **Algorithm to delete element from circular queue.**

**CQDELETE (Q, F, R)**

* This function deletes an element from circular queue
* The Queue is represented by vector Q which contains N elements.
* F is a pointer which points to the front end
* R is a pointer which points to the rear end

1. **[Check Underflow error]**

If Front = -1 then

Write (“Queue Underflow“)

Exit

1. **[Delete element]**

X=Q[Front]

1. **[Queue empty?]**

If Front = Rear or Front=N-1 then

Front🡨Rear🡨-1

Else

Front🡨Front+1

Return (X)

1. **[Finished]**

Exit

**Priority Queue**

* A queue in which we are able to insert items or remove items from any position based on some priority is known as priority queue.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **R1** | **R2** | **……** | **Ri-1** | **O1** | **O2** | **…….** | **Oj-1** | **B1** | **B2** | **……** | **Bk-1** |
| **1** | **2** | **……** | **1** | **2** | **2** | **……..** | **2** | **3** | **3** | **……** | **3** |

* Figure represents a priority queue of jobs waiting to use a computer
* Priorities of 1, 2, 3 have been attached to jobs of real time, on-line and batch respectively.
* Figure shows how the single priority queue can be visualized as three separate queues.

Priority 1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| R1 | R2 | ……… | Ri-1 | ….. |

Ri  
Priority 2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| O1 | O2 | ……… | Oj-1 | ….. |

Oj  
Priority 3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| B1 | B2 | ……… | Bk-1 | ….. |

Bk

* When elements are inserted, they are always added at the end of one of the queues as determined by the priorities.
* Elements in the second queue are deleted only when the first queue is empty.
* Elements in the third queue are deleted only when the first and second queue are empty.
* There are two types of priority queue(1)Ascending priority queue (2)Descending priority queue.

**Comparison of LIFO and FIFO**

|  |  |
| --- | --- |
| **LIFO** | **FIFO** |
| (1) In LIFO the insertion and deletion operation are performed at only one end. | (1) In FIFO the insertion and deletion operation are performed at two different ends. |
| (2) In LIFO the element which is inserted last is first to delete. | (2) In FIFO the element which is inserted first is first to delete. |
| (3) LIFO require only one pointer called TOP | (3) FIFO requires two pointers called front and rear. |
| (4) Example: piles of trays in cafeteria | (4) Example: students at registration counter |
| (5) In LIFO there is no wastage of memory space. | (5) In FIFO even if we have free memory space sometimes we cannot use that space to store elements. |

* **Write a program to perform stack operation.**

#include<stdio.h>

#include<conio.h>

#define N 5

void main()

{

int s[N],val,op,i,top=-1;

while(1)

{

clrscr();

printf("\nSTACK OPERATION \n");

printf("\n1-PUSH \n2-POP \n3-PEEP \n4-CHANGE \n5-DISPLAY \n6-EXIT \n");

printf("\nSELECT OPERATION :- ");

scanf("%d",&op);

switch(op)

{

case 1:

if(top>=N-1)

{

printf("\nSTACK OVERFLOW");

}

else

{

printf("ENTER VALUE TO PERFORM PUSH OPERATION :- ");

scanf("%d",&val);

top++;

s[top]=val;

printf("VALUE INSERTED...");

}

break;

case 2:

if(top<0)

{

printf("\nSTACK UNDERFLOW");

}

else

{

val=s[top];

top--;

printf("\nPOPED ELEMENT :-%d",val);

}

break;

case 3:

printf("\nENTER POSITION TO SHOW DATA :- ");

scanf("%d",&i);

if(i>0)

{

if((top-i+1)<0)

{

printf("\nSTACK UNDERFLOW");

}

else

{

val=s[top-i+1];

printf("\nDATA AT POSITION %d FROM TOP= %d",i,val);

}

}

else

{

printf("\nINVALID POSITION");

}

break;

case 4:

printf("\nENTER POSTION TO CHANGE THE VALUE :- ");

scanf("%d",&i);

if((top-i+1)<0)

{

printf("\nSTACK UNDERFLOW");

}

else

{

printf("\nENTER NEW VALUE FOR %d POSITION :- ",i);

scanf("%d",&val);

s[top-i+1]=val;

printf("VALUE CHANGED AT THE POSITION %d...",i);

}

break;

case 5:

if(top==-1)

{

printf("\nSTACK IS EMPTY");

}

else

{

printf("\nNOW STACK IS :-\n");

printf("\n top->");

for(i=top;i>=0;i--)

{

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

printf("\t");

}

}

break;

case 6:

exit(0);

break;

default:

printf("\nINVALID CHOICE FOR OPERATION");

break;

}

getch();

}

}

* **Write a program to perform Queue Operation.**

#include<stdio.h>

#include<conio.h>

#define N 5

void main()

{

int q[N],val,op,i,front=-1,rear=-1;

while(1)

{

clrscr();

printf("\nSTACK OPERATION \n");

printf("\n1-PUSH \n2-POP \n3-DISPLAY \n4-EXIT \n");

printf("\nSELECT OPERATION :- ");

scanf("%d",&op);

switch(op)

{

case 1:

if(rear==N-1)

{

printf("\nQUEUE OVERFLOW");

}

else

{

printf("ENTER VALUE TO PERFORM PUSH OPERATION :- ");

scanf("%d",&val);

rear++;

q[rear]=val;

printf("VALUE INSERTED...");

if(front==-1)

front=0;

}

break;

case 2:

if(front<0)

{

printf("\nQUEUE UNDERFLOW");

}

else

{

val=q[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else

{

front++;

}

printf("\nPOPED ELEMENT :-%d",val);

}

break;

case 3:

if(front==-1)

{

printf("\nQUEUE IS EMPTY");

}

else

{

printf("\nNOW QUEUE IS :-\n");

printf("\n front->");

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

{

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

printf("\t");

}

}

break;

case 4:

exit(0);

break;

default:

printf("\nINVALID CHOICE FOR OPERATION");

break;

}

getch();

}

}

* **Write a program to perform Circular queue operation.**

#include<stdio.h>

#include<conio.h>

#define N 5

void main()

{

int q[N],val,op,i,j,k,front=-1,rear=-1;

while(1)

{

clrscr();

printf("\nSTACK OPERATION \n");

printf("\n1-PUSH \n2-POP \n3-DISPLAY \n4-EXIT \n");

printf("\nSELECT OPERATION :- ");

scanf("%d",&op);

switch(op)

{

case 1:

printf("ENTER VALUE TO PERFORM PUSH OPERATION :- ");

scanf("%d",&val);

if(rear>=N-1)

{

rear=0;

}

else

{

rear++;

}

if(rear==front)

{

if(rear==0)

rear=N-1;

else

{

rear--;

}

printf("\nQUEUE OVERFLOW");

break;

}

q[rear]=val;

printf("VALUE INSERTED...");

if(front==-1)

front=0;

break;

case 2:

if(front==-1)

{

printf("\nQUEUE UNDERFLOW");

}

else

{

val=q[front];

if(front==rear||front==N-1)

{

front=-1;

rear=-1;

}

else

{

front++;

}

printf("\nPOPED ELEMENT :-%d",val);

}

break;

case 3:

printf("front=%d rear=%d",front,rear);

if(front==-1)

{

printf("\nQUEUE IS EMPTY");

}

else

{

printf("\nNOW QUEUE IS :-\n");

printf("\n front->");

if(front>rear)

{

for(j=0;j<=rear;j++)

{

printf("%d\n",q[j]);

printf("\t");

}

for(k=front;k<=N-1;k++)

{

printf("%d\n",q[k]);

printf("\t");

}

}

else

{

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

{

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

printf("\t");

}

}

}break;

case 4:

exit(0);

break;

default:

printf("\nINVALID CHOICE FOR OPERATION");

break;

}

getch();

} }

**IMPORTANT QUESTIONS(Asked question in the GTU Exam)**

1. What is Queue? Explain disadvantages of simple queue.
2. Explain Queue fundamentals with Queue Insertion & deletion algorithms.
3. What is Circular queue? Compare circular queue with normal queue.
4. Write and explain algorithm to insert(PUSH) & delete(POP) element in circular queue.

**OR**

Using array data structure, write an algorithm to insert & delete a data item from the circular queue.

**Inspirational Quote:**

**The secret of success is to know something nobody else knows.**