STACKS & QUEUES

Chapter at a Glance

- **Stacks:** A stack is an abstract data type, which declares two methods PUSH and POP. Stacks are implemented either by an array or by a linked list.
- The **PUSH** operation allows us to insert data at the end of an array or linked list.
- The **POP** operation allows us to remove data from the end of an array or linked list.
- A **STACK** is called a last in first out (LIFO) system.
- A stack may be represented by a linked list. The first node of the list will be the topmost element of the stack and is pointed by a top pointer. This type of stack representation is called **linked stack**. The stack can be declared as follows:

```
typedef struct linked_list
{
  int data;
  struct linked_list *next;
} lstack;
lstack *top;
```

- **Various types of expression:** A mathematical expression involves constants (operands) and operations (operators).
  Infix notation: operand1 *operator* operand2, **A + B**
  Prefix notation: *operator* operand1 operand2, **+ A B**
  Postfix notation; operand1 operand2 *operator*, **A B +**
- **Conversion from INFIX to POSTFIX expression:** In order to convert the infix to its corresponding postfixes form; we need to do the following steps:
  - Fully parenthesize the expression according to the priority of different operators.
  - Move all operators so that they replace their corresponding right parentheses.
  - Delete all parentheses.

  The priorities of different operators are given below:

Operators	Priority
Unary - , unary + , not (!)	4
* , / , % , and (& / &&)	3
+ , - , or (| / ||)	2
<, <=, >, >=, = =, !=	1

- **Priority Queue:** A priority queue is essentially a list of items in which each item has associated with it **a *priority***. In general, different items may have different priorities and we speak of one item having a higher priority than another. Given such a list we can determine which is the highest (or the lowest) priority item in the list. Items are inserted into a priority queue in any, arbitrary order. However, items are withdrawn from a priority queue in order of their priorities starting with the highest priority item first. Two elements with the same priority are processed according to the order in which they were added to the queue.
- **De-queue:** De-queue is a linear data structure, where insertions and deletions are made to or from either end of the structure.

Multiple Choice Type Questions

1. A linear list in which elements can be added or removed at either end but not in the middle is known as [WBUT 2007, 2009, 2010]
a) queue b) deque c) stack d) tree
Answer: (b)

2. The prefix expression for the infix expression a * (b + c) / e – f is [WBUT 2008]
a) /*a + bc –ef b) –/* + abcef c) –/*a + bcef d) None of these
Answer: (a)

3. The number of stacks required to implement mutual recursion is [WBUT 201
a) 3 b) 2 c) 1 d) none of these
Answer: (c)

4. Priority queue can be implemented using [WBUT 201
a) array b) linked list c) heap d) all of these
Answer: (d)

5. Reserve Polish notation is often known as [WBUT 2012
a) Infix b) Prefix c) Postfix d) none of these
Answer: (c)

6. The evaluation of the postfix expression [WBUT 201
3, 5, 7, *, +, 12, % is
a) 2 b) 3 c) 0 d) 3.17
Answer: (a)

7. The operation for adding an entry to a stack is traditionally called [WBUT 201
a) Add b) Append c) Insert d) Push
Answer: (d)

8. The best data structure to evaluate an arithmetic expression in postfix form is [WBUT 20
a) Queue b) Stack c) Tree d) Linked Lis
Answer: (b)

9. The heap (represented by an array) constructed from the list of numbers 30, 80, 60, 15, 55, 17 is – [WBUT 20
a) 60, 80, 55, 30, 10, 17, 15 b) 80, 55, 60, 15, 10, 30, 17
c) 80, 60, 30, 17, 55, 15, 10 d) none of these
Answer: (b)

10. The postfix equivalent of the prefix $*+ab-cd$ is [WBUT 2014, 2015, 20
a) $ab+cd-*$ b) $abcd+-*$ c) $ab+cd*-$ d) $ab+-cd*$
Answer: (a)

11. The followi
(2), pop, push
out values are
a) 2, 2, 1, 1,
Answer: (a)

12. Stack cann
a) evaluate
b) impleme
c) allocate
d) convert i
Answer: (c)

13. The followin
push(1), push(2
the sequence o
a) 2, 2, 1, 2,
Answer: (b)

14. The deque c
a) as a stac
c) both as a
Answer: (b)

15. Inserting an
operation and d
..... operat
a) push, pop
c) insert, del
Answer: (a)

16. To make a qu
a) front = rea
c) front = rea
Answer: (a)

1. a) Convert th
using stack:
(A + B) * C – (D –
b) What is a Max

11. The following sequence of operations is performed on a stack: push (1), push (2), pop, push (1), push (2), pop, pop, pop, push (2), pop. The sequence of popped out values are [WBUT 2014]
a) 2, 2, 1, 1, 2 b) 2, 2, 1, 2, 2 c) 2, 1, 2, 2, 1 d) 2, 1, 2, 2, 2
Answer: (a)

12. Stack cannot be used to [WBUT 2015]
a) evaluate an arithmetic expression in postfix form
b) implement recursion
c) allocate resources (like CPU) by the operating system
d) convert infix expression to its equivalent postfix expression
Answer: (c)

13. The following sequence of operations is performed on a stack [WBUT 2015]
push(1), push(2), pop( ), push(1), push(2), pop( ), pop( ), pop( ), push(2), pop( ), the sequence of popped out values are
a) 2, 2, 1, 2, 1 b) 2, 2, 1, 1, 2 c) 2, 1, 2, 2, 2 d) 2, 1, 2, 2, 1
Answer: (b)

14. The deque can be used [WBUT 2016]
a) as a stack b) as a queue
c) both as a stack and as a queue d) none of these
Answer: (b)

15. Inserting an item into the stack when stack is not full is called ..... operation and deletion of item from the stack, when stack is not empty is called ..... operation. [WBUT 2017]
a) push, pop b) pop, push
c) insert, delete d) delete, insert
Answer: (a)

16. To make a queue empty, elements can be deleted till [WBUT 2018]
a) front = rear + 1 b) front = rear – 1
c) front = rear d) None of these
Answer: (a)

Short Answer Type Questions

1. a) Convert the following infix expression into equivalent postfix expression using stack: [WBUT 2009, 2010, 2015]
(A + B) * C – (D – E) / (F + G).
b) What is a Max Heap? [WBUT 2009]

$(A+B)*(C-(D-E))/(F+G)$

Answer:

a)

Symbol scanned	Stack	Output
(	(	-
A	(	A
+	(+	A
B	+	AB
)	-	AB+
*	*	AB+
C	*	AB+C
-	-	AB+C*
(	-(	AB+C*
D	-(	AB+C*D
-	-(-	AB+C*D
E	-(-	AB+C*DE
)	-	AB+C*DE-
/	-/	AB+C*DE-
(	-/(	AB+C*DE-
F	-/(	AB+C*DE-F
+	-/(+	AB+C*DE-F
G	-/(+	AB+C*DE-FG
)	-/	AB+C*DE-FG+
NONE	EMPTY	AB+C*DE-FG+/-

b) A heap is a tree-based data structure that satisfies the *heap property:* if B is a child node of A, then key(A) ≥ key(B). This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a *max-heap.*

2. What is a priority queue?
Mention the different design options for priority queue. **[WBUT 2009, 2012, 2013]** **[WBUT 2009]**

Answer:

A priority queue is essentially a list of items in which each item has associated with it a *priority*. In general, different items may have different priorities and we speak of one item having a higher priority than another. Given such a list we can determine which is the highest (or the lowest) priority item in the list. Items are inserted into a priority queue in any, arbitrary order. However, items are withdrawn from a priority queue in order of their priorities starting with the highest priority item first.
Two elements with the same priority are processed according to the order in which they were added to the queue.
Often a type of binary tree called a heap is used to store the items in a priority queue. A heap has a root node (usually draw at the top) and it has two children, a left child and a right child. Each node (parent, and left or right child) has a value associated with it, and the property that the parent's value is more than either of its children. When we add

item to the heap we check
then we swap the child wi
tree is "balanced," (i.e. tha
There are some other meth
They are
Sorted list implementation
Unsorted list implementa
append it to the end. To ge
the highest priority.

3. Define dequeue?

What is dequeue?

Answer:

Dequeue is a linear data s
either end of the structure.

4. Convert the following operator stack and outpu

$A*B+C*(D-E)-F*G$

Answer:

Symbol sc
A
*
B
+
C
*
(
D
-
E
)
-
F
*
G
NONE

item to the heap we check to see if the parent's value is more or less than it. If it is more, then we swap the child with the parent. We check again, and maybe swap again, until the tree is "balanced," (i.e. that it obeys the heap property).

There are some other methods of implementing priority queue which are not so efficient. They are

***Sorted list implementation*:**

***Unsorted list implementation*:** Keep a list of elements as the queue. To add an element, append it to the end. To get the next element, search through all elements for the one with the highest priority.

3. Define dequeue? **[WBUT 2010]**

OR,

What is dequeue? **[WBUT 2015]**

Answer:

Dequeue is a linear data structure, where insertions and deletions are made to or from either end of the structure.

4. Convert the following infix expression to postfix notation by showing the operator stack and output string after reading each input token:

$A*B+C*(D-E)-F*G$ **[WBUT 2011]**

Answer:

Symbol scanned	Stack	Output
A		A
*	*	A
B	*	AB
+	+	AB*
C	+	AB*C
*	+*	AB*C
(	+* (	AB*C
D	+* (	AB*CD
-	+*(-	AB*CD
E	+* (-	AB*CDE
)	+*	AB*CDE-
-	+-	AB*CDE-*
F	+-	AB*CDE-*F
*	+-*	AB*CDE-*F-
G	+-*	AB*CDE-*FG
NONE	NONE	AB*CDE-*FG*-+

5. a) Consider the array int a [10] [10] and the base address 2000, then calculate the address of the array a [2] [3] in the row and column major ordering. [WBUT 20

Answer:

In row major ordering the linear offset from the beginning of the array to any given element A[row][column] can then be computed as:

offset = row*NUMCOLS + column

Thus a[2][3] will have address = 2000 + 2*10+3 = 2023

In column major ordering the memory offset could then be computed as:

offset = row + column*NUMROWS

Thus a[2][3] will have address = 2000 + 2+3*10 = 2032

b) Write the advantage of circular queue over linear queue. [WBUT 20

OR,

Why circular queue is better than simple queue? [WBUT 20

OR,

Why circular queue is used over simple queue [WBUT 20

Answer:

Refer to Question No. 5 of Long Answer Type Questions.

6. Evaluate the following postfix expression: [WBUT 20

4, 5, 4, 2, ∧, +, *, 2, 2, ∧, 9, 3, /, *, –

Write pseudo code for evaluating postfix expression.

Answer:

1st Part:

4, 5, 4, 2, ∧, +, *, 2, 2, ∧, 9, 3, /, *, –
≡ 4, 5, 16, +, *, 2, 2, ∧, 9, 3, /, *, –
≡ 4, 21, *, 2, 2, ∧, 9, 3, /, *, –
≡ 84, 2, 2, ∧, 9, 3, /, *, –
≡ 84, 4, 9, 3, /, *, –
≡ 84, 4, 3, *, –
≡ 84, 12 –
≡ 76

2nd Part: *Refer to Question No. 2 of Long Answer Type Questions.*

7. How many types of priority queues are there? Can you consider stack as priority queue? If yes, how? [WBUT 2

Answer:

Mainly there are two kinds of priority queue: 1) Static priority queue 2) Dynamic pri queue.

Stack can be considered as a priority queue where the priorities of each element inserted are considered higher than the previous one. This will let it behave e like a structure (i.e., stack).

8. Suppose a queue is imp element at the kth positio

Answer:

```
static int head, tai
of queue
static char theQueue
//-----------------
// Function: InitQue
// Purpose: Initiali
// Returns: void
//-----------------
void InitQueue()
{
        head = tail =
}
//-----------------
// Function: Enqueue
// Purpose: Enqueue
// Returns: TRUE if
//          or FALSE
//-----------------
int Enqueue(char ch,
{
        int c;
        // Check to
        if(isFull())

        // Increment
        tail++;
      for (c = MAX_S
            array[c+

        array[posit
        return TRUE;
}
```

1. Write the difference b priority queue.

Answer:

1st Part:

A Stack is a (ordered) coll sequence and all deletions stack is a container of da compared to the order the item that has been put last

8. Suppose a queue is implemented by an array. Write an algorithm to insert a new element at the kth position of the array. **[WBUT 2015]**

Answer:

```
static int head, tail; // Declare global indices to head and tail
of queue
static char theQueue[MAX_SIZE];          // The queue
//-----------------------------------------
// Function: InitQueue()
// Purpose: Initialize queue to empty.
// Returns: void
//-----------------------------------------
void InitQueue()
{
      head = tail = -1;
}
//-----------------------------------------
// Function: Enqueue()
// Purpose: Enqueue an item into the kth position of queue.
// Returns: TRUE if enqueue was successful
//          or FALSE if the enqueue failed.
//-----------------------------------------
int Enqueue(char ch, k)
{
      int c;
      // Check to see if the Queue is full
      if(isFull()) return FALSE;

      // Increment tail index
      tail++;
    for (c = MAX_SIZE - 1; c >= k - 1; c--)
         array[c+1] = array[c];

      array[position-1] = ch;
      return TRUE;
}
```

Long Answer Type Questions

1. Write the difference between stack and queue and implement the operations of priority queue. **[WBUT 2007]**

Answer:

1st Part:

A Stack is a (ordered) collection of items, where all insertions are made to the end of the sequence and all deletions always are made from the end of the sequence. In principle a stack is a container of data items, from which we get data items out in reverse order compared to the order they have been put into the container. We can also say that the item that has been put last in is coming first out. That's why a stack is also called LIFO

(Last In First Out list). We can as well say that the item, which is put first in the container is get last out (First In Last Out: FILO).
A Queue is a (ordered) collection of items, where all insertions are made to the end of the sequence and all deletions always are made from the beginning of the sequence. In principle a queue is container from which data items are retrieved out in the same order they are put in. This means that the queue is a container that preserves the order of items put there. We can also say that the item that is put last into the queue is taken last out from the queue. We can also say that the item which is put first into the queue is taken first out. (First In First Out: FIFO).

2nd Part:
A priority queue is an abstract data type that supports the following three operations:
insertWithPriority: add an element to the queue with an associated priority
getNext: remove the element from the queue that has the *highest priority*, and return it
/* C implementation using array of size MAX. Assume structure is as below*/

```
struct data
      {
       int val,p,o;
      }d[MAX];

    int front=rear=-1; /*initial values*/

/* insertWithPriority*/
    void insert(data d1)
    {
        if(rear==MAX-1)
           printf("Priority Queue is Full");
        else
        {
           rear++;
           d[rear]=d1;
           if(front==-1)
                front=0;
           data temp;
           for(int i=front;i<=rear;i++)
               for(int j=i+1;j<=rear;j++)
               {
                    if(d[i].p > d[j].p)
                    {
                         temp=d[i];
                         d[i]=d[j];
                         d[j]=temp;
                    }
                    else
                    {
                         if(d[i].p==d[j].p)
                         {
                              if(d[i].o > d[j].o)
```



```
                    }
               }
          }
data getNext()
{
          data d1;
          if(front=
               printf
          else
          {
               d1=d[f
               if(fro
                    f
               else
                    f
          }
          return d1
}
```

2. Write an algorithm to co

Write an algorithm to evalu
Answer:

- Scan the Infix string fro
- Initialise an empty stack
- If the scannned characte character is an operator a
- If the scanned characte precedence of the char topStack has higher pre the scanned character to topStack has precedence
- Repeat this step till all th (After all characters are have to the Postfix strin Pop the stack. Repeat thi

Return the Postfix string.

```
                                {
                                    temp=d[i];
                                    d[i]=d[j];
                                    d[j]=temp;
                                }
                            }
                        }
                    }
                }
            }
            data getNext()
            {
                data d1;
                if(front==-1)
                   printf("Priority Queue is Empty");
                else
                {
                   d1=d[front];
                   if(front==rear)
                        front=rear=-1;
                   else
                        front++;
                }
                return d1;
            }
```

2. Write an algorithm to convert an infix expression to postfix using stack. **[WBUT 2008, 2018]**

OR,

Write an algorithm to evaluate a postfix expression **[WBUT 2017]**

Answer:

- Scan the Infix string from left to right.
- Initialise an empty stack.
- If the scannned character is an operand, add it to the Postfix string. If the scanned character is an operator and if the stack is empty Push the character to stack.
- If the scanned character is an Operand and the stack is not empty, compare the precedence of the character with the element on top of the stack (topStack). If topStack has higher precedence over the scanned character Pop the stack else Push the scanned character to stack. Repeat this step as long as stack is not empty and topStack has precedence over the character.
- Repeat this step till all the characters are scanned.

(After all characters are scanned, we have to add any character that the stack may have to the Postfix string.) If stack is not empty add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.

Return the Postfix string.

The C-function to convert an infix expression to it's equivalent postfix form is as given below:

```
void postfix(char x[])
{
  int i = 0;
  while (x[i]! = '\0')
  {
    if (isalpha(x[i]))
      printf("%c", x[i]);
    else if (x[i] == ')')
    {
      while (s[top] != '(')
        printf(" %c", pop());
      pop(); // delete '(' symbol
    }
    else
    {
      while (isp(s[top])>= icp(x[i]))
        printf(" %c", pop()); push(x[i]);
    }
      i++;
      while (top > 0)
        printf(" %c", pop());
  }
}
int isp(char a)
{
  if (a == ' *' || a == ' /' || a == '%')
    return (3);
  else if (a == ' + ' || a == ' - ')
    return (2);
  else if (a == ' (')
    return (1);
  else if (a == ' # ')
    return ( - 1);
  else if (a = ') ')
    return (0);
}
int icp(char a)
{
  if (a == '*' || a == '/' || a == '%')
    return (3);
  else if (a == ' + ' || a == ' - ')
    return (2);
  else if (a == ' (')
    return (4);
  else if (a == ') ')
    return (0);
}
```


3. a) Define circ

Answer:

Refer to Questi

b) Write an algo

Answer:

```
void QInsert(
{
  if (rear ==
  {
    printf("Q
    return;
  }
  CQ[++rear] =
  if (front ==
    front = 0;
}
```

c) What is input re

Answer:

An input-restricted
can only be made at

4. a) What is a Stac

b) Write a C functio
list.

c) Explain three use

Answer:

a) A Stack ADT is a
end of the sequence
principle a stack is a
order compared to th
the item that has bee
LIFO ((Last In First
container is get last ou

b) Assume that the list

```
typedef struct no
struct node
{
int data;
nptr next;
};
/* function pop *
int pop(nptr s) /*
{
nptr temp;
int y;
```

3. a) Define circular queue.

Answer: **[WBUT 2008, 2018]**

Refer to Question No. 5(1st & 2nd Part) of Long Answer Type Questions.

b) Write an algorithm to insert an item in circular queue. **[WBUT 2008, 2018]**

Answer:

```
void QInsert(int item)
{
  if (rear == N-1)
  {
    printf("Queue Is Full");
    return;
  }
  CQ[++rear] = item;
  if (front ==  - 1)
    front = 0;
}
```

c) What is input restricted Dequeue? **[WBUT 2008, 2018]**

Answer:

An input-restricted Dequeue is one where deletion can be made from both ends, but input can only be made at one end.

4. a) What is a Stack ADT?

b) Write a C function of popping an element from a stack implemented using linked list.

c) Explain three uses of stack data structure. **[WBUT 2009]**

Answer:

a) A Stack ADT is a (ordered) collection of items, where all insertions are made to the end of the sequence and all deletions always are made from the end of the sequence. In principle a stack is a container of data items, from which we get data items out in reverse order compared to the order they have been put into the container. We can also say that the item that has been put last in is coming first out. That's why a stack is also called the LIFO ((Last In First Out list). We can as well say that the item, which is put first in the container is get last out (First In Last Out: FILO).

b) Assume that the list is defined as below:

```
typedef struct node *nptr;
struct node
{
int data;
nptr next;
};
/* function pop */
int pop(nptr s) /*Function to pop the elements*/
{
nptr temp;
int y;
```

```
if(s->next==NULL)
        {
printf("Underflow on Pop");
return(-1);
}
Else
{
y=s->next->data;
temp=s->next;
s->next=temp->next;
free(temp);
return(y);
}
}
```

c) Use of stack

Reversing Data: We can use stacks to reverse data.

(example: files, strings). It is very useful for finding palindromes.

Consider the following pseudocode:

```
1)    read (data)
2)    loop (data not EOF and stack not full)
            1) push (data)
            2) read (data)
3)    Loop (while stack notEmpty)
            1) pop (data)
            2) print (data)
```

```
Converting Decimal to Binary:
Consider the following pseudocode
Read (number)
Loop (number > 0)
      1) digit = number modulo 2
      2) print (digit)
      3) number = number / 2
```

The problem with this code is that it will print the binary number backwards. (ex: 19 becomes 11001000 instead of 00010011.)

To remedy this problem, instead of printing the digit right away, we can push it onto the stack. Then after the number is done being converted, we pop the digit out of the stack and print it.

Evaluating arithmetic expressions.

In high level languages, infix notation cannot be used to evaluate expressions. We must analyze the expression to determine the order in which we evaluate it. A common technique is to convert a infix notation into postfix notation, then evaluating it. This is done using stack.

5. i) What is Circular queue?

ii) Write Q-insert algorithm for the circular queue.

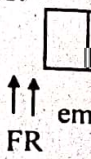
OR,

Write an algorithm to insert an element into circular queue.

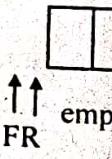
[WBUT 2010]
[WBUT 2010]
[WBUT 2017]

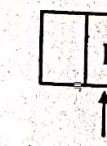
Answer:

i) The two algori
front pointer F n
amount of memo
performing opera
certain intervals.
Let us consider th
rear pointers.



To avoid this pro
prevents an excess
circular fashion w
circular queues.
So the above sequ
queue).





de
As we can see that
by redirecting the re

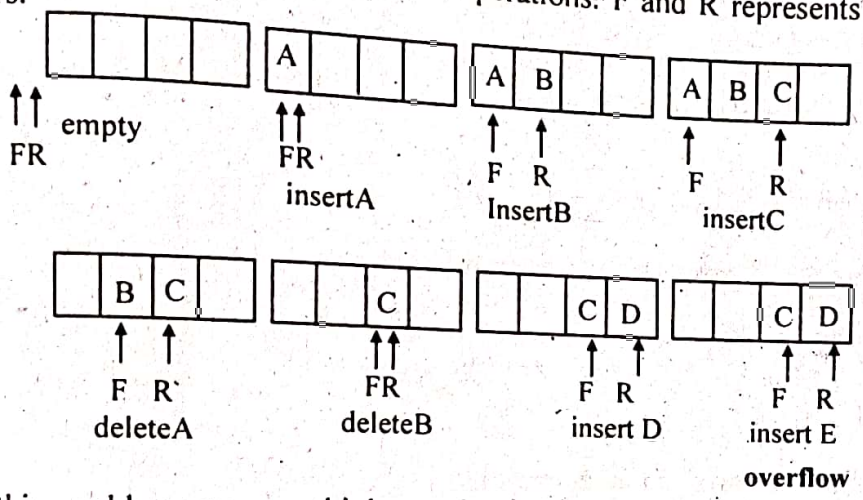
ii) Let us now write
Let us also assume t

```
void CQInsert(i
{
  if (rear == N
  {
    printf("Que
```

Answer:

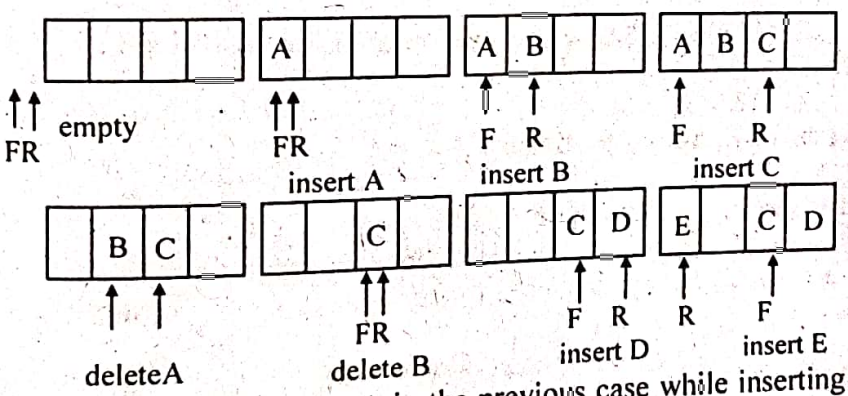
i) The two algorithms QINSERT & QDELETE can be very wasteful of storage if the front pointer F never manages to catch up the rear pointer. Actually an arbitrary large amount of memory would be required to accommodate the elements. The method of performing operations on a Queue should only be used when the queue is emptied at certain intervals.

Let us consider the following sequence of operations. F and R represents the front and rear pointers.



To avoid this problem we can think an alternative representation of a queue, which prevents an excessive use of memory. In this representation elements are arranged as in a circular fashion where the rear again points to the front. This type of queue is known ***circular queues***.

So the above sequence of operations can be represented are shown below (in circular queue).



As we can see that the overflow issue in the previous case while inserting E is resolved by redirecting the rear to the front. This is the essence of circular queue.

ii) Let us now write the insert and delete functions of the circular queue implementation. Let us also assume that F and R represents front and rear pointers respectively

```
void CQInsert(int item)
{
  if (rear == N-1)
  {
    printf("Queue Is Full");
```

```
    return;
  }
  CQ[++rear] = item;
  if (front ==  - 1)
    front = 0;
}
int CQDelete()
{
  int x;
  if (front ==  - 1)
  {
    printf("Queue Is Empty");
    exit(0);
  }
  x = CQ[front];
  CQ[front] =  - 1;
  if (front == rear)
    front = rear =  - 1;
  else
    front++;
  return x;
}
```

In linear queue the condition for queue full is
QREAR==MAXLIMIT
Suppose maxlimit is ten and queue is full now if we delete 9 element from queue then inspite of queue is empty we cannot insert any element in the queue. This wastage of memory is solved through circular queue where queue full condition is
QREAR==Qfront+1

6. Define the ADT for stack. Show the implementation of the stack data structure using linked list. [WBUT 2011]

Answer:

The Stack ADT has two main functions:

Push

input: a stack data object; an element of the base type
output: a boolean indicating whether the operation was successful
effect: the element is added to the sequence in the stack, in the top position

Pop

input: a stack data object
output: a boolean indicating whether the operation was successful.
effect: the element in the top position in the sequence in the stack is removed

Stack Implementation

First, let us define the linked list data structure.

```
typedef struct linked_list
{
   int data;
```



```
      struct l
} Node;
```

For us to be ab
following code u

```
// recursive
// of the st
void Display
{
    // recur
    if (curre
    {
        retu
    }
    // the no
    // displa
    printf("
    // recurs
    // displa
    DisplaySt
}
```

The code for push

```
// push item
// this is sa
// at the top
void Push(int
{
    // assumpt
program
    // 1. crea
    Node *temp
    temp->data

    // 2. inse
    temp->next

    // 3. upda
    head = temp
}
```

The code for poppi

```
// pop an eleme
// this is same
// from the lis
Node* Pop()
{
    // check fo
    if (head ==
    {
        printf(
        return
```

```
    struct linked_list *next;
} Node;
```

For us to be able to test our code, we need to define a way to display our stack. The following code uses recursion to display stack.

```
// recursively display the contents
// of the stack
void DisplayStack(Node* currentNode)
{
    // recursive termination condition
    if (currentNode == NULL)
    {
        return;
    }
    // the node is not null
    // display the data
    printf(" -> %d", currentNode->data);
    // recursively call the display to
    // display the next element in the stack
    DisplayStack(currentNode->next);
}
```

The code for pushing an element onto the stack is as given below.

```
// push item on the stack
// this is same as adding a node
// at the top of the list
void Push(int dataToAdd)
{
    // assumption: head is already defined elsewhere in the
program
    // 1. create the new node
    Node *temp = new Node;
    temp->data = dataToAdd;

    // 2. insert it at the first position
    temp->next = head;

    // 3. update the head to point to this new node
    head = temp;
}
```

The code for popping an element from stack is given below.

```
// pop an element from the stack
// this is same as removing the first element
// from the list
Node* Pop()
{

    // check for empty list
    if (head == NULL)
    {
        printf("Stack is empty\n");
        return NULL;
```

```
    }
    // get the top node
    Node *firstNode = head;
    // move the head
    head = head->next;
    // disconnect the node
    // from the list
    firstNode->next = NULL;

    // return the top node
    return firstNode;
}
```

7. What are the differences between stack and Queue? Write the algorithm for insertion and deletion in a circular Queue. [WBUT 2013]

Answer:

1st Part: ***Refer to Question No. 1(1st Part) of Long Answer Type Questions.***

2nd Part: ***Refer to Question No. 5 of Long Answer Type Questions.***

8. a) Evaluate the postfix expression using stack: [WBUT 2016]

3, 16, 2, +, *, 12, 6, /, –

b) Convert the infix expression into its equivalent prefix expression using stack:

$a+b*c+(d*e+f)*g.$

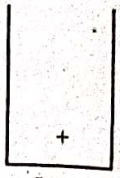
Answer:

a) The steps are as shown below:

3, 16, 2, +, *, 12, 6, /, –

Input token	Operation	Stack contents (top on the right)	Details
3	Push on the stack	3	
16	Push on the stack	3, 16	
2	Push on the stack	3, 16, 2	
+	Add	3, 18	Pop two values: 16 and 2 and push the result 18 on the stack
*	Multiply	54	Pop two values: 3 and 18 and push the result 54 on the stack
12	Push on the stack	54, 12	
6	Push on the stack	54, 12, 6	
/	Divide	54, 2	Pop two values: 12 and 6 and push the result 2 on the stack
-	Divide	54, 2	Pop two values: 54 and 2 and push the result 52 on the stack
(End of tokens)	(Return the result)	52	Pop the only value 52 and return it

b) First, the symbol "
pushed onto the stack
affairs at this juncture

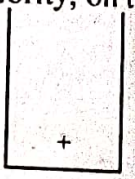


Stack

Next a '*' is read. Th
nothing is output and
stack. Next, "c" is rea

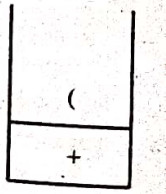
Stack

The next symbol is a
the output, pop the oth
but equal priority, on t



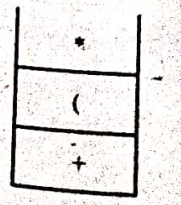
Stack

The next symbol rea
the stack. Then "d" is



Stack

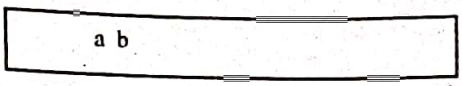
We continue by readi
when a closed parenth
output.



Stack

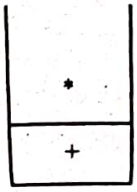
b) First, the symbol "a" is read, so it is passed through to the output. Then '+' is read and pushed onto the stack. Next "b" is read and passed through to the output. The state of affairs at this juncture is as follows:

Stack



Output

Next a '*' is read. The top entry on the operator stack has lower precedence than '*', so nothing is output and '*' is put on the
stack. Next, "c" is read and output. Thus far, we have



Stack

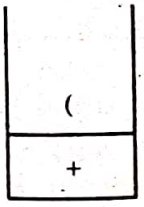
Output

The next symbol is a '+'. Checking the stack, we find that we will pop a '*' and place it on the output, pop the other '+', which is not of lower
but equal priority, on the stack, and then push the '+'.

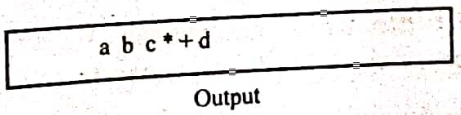
Stack

Output

The next symbol read is an '(', which, being of highest precedence , is placed on the stack. Then "d" is read and output.

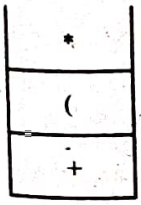


Stack

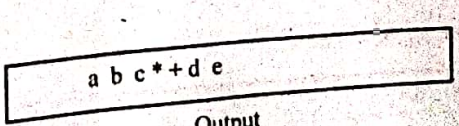


Output

We continue by reading a '*'. Since open parentheses do not get removed except when a closed parenthesis is being processed, there is no output. Next, "e" is read and output.

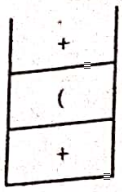


Stack

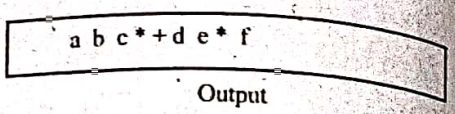


Output

The next symbol read is a '+'. We pop and output '*' and then push '+'. Then we read and output.

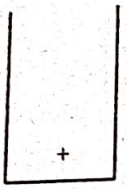


Stack

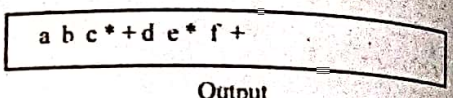


Output

Now we read a ')', so the stack is emptied back to the '('. We output a '+'.

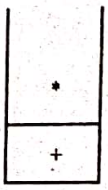


Stack

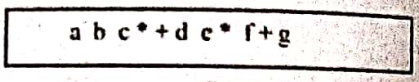


Output

We read a '*' next; it is pushed onto the stack. Then "g" is read and output.



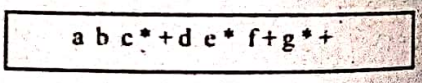
Stack



Output

The input is now empty, so we pop and output symbols from the stack until it is empty.

Stack



Output

9. Convert the infix expression 9+5*7−6^2+15/3 into its equivalent postfix expression and evaluate that postfix expression, clearly showing the state of the stack. [WBUT 2017]

Answer:

Symbol scanned	Stack	Output
9		9
+	+	9
5	+	95
*	+*	95
7	+*	957

Symbol scanned
-
6
^
2
+
15
/
3
NONE

Symbol scanned	Stack	Output
-	-	957*+
6	-	957*+6
^	-^	957*+6
2	-^	957*+62
+	+	957*+62^-
15	+	957*+62^15
/	+/	957*+62^15
3	+/	957*+62^153
NONE	NONE	957*+62^153/+