UNIT - 1

→ Data structures.

Data are simply values or set of values. data values refer to a single unit of values an entity is something that has certain attributes or property which may be assigned values. These values may be numeric or non numeric.

Data structure is logical or mathematical ~~order~~ model of particular organization of data is called DS. Choice of particular data model depends on the particular application.

Types of data structures.

→ Array

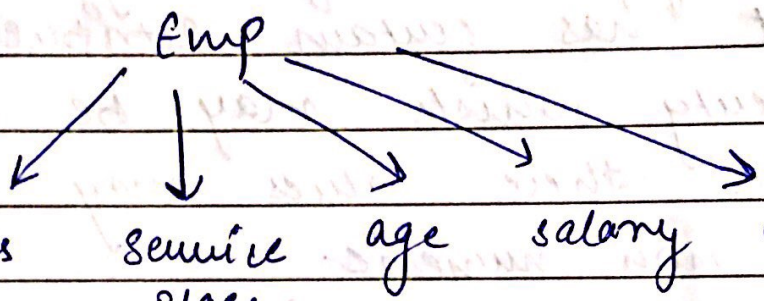
The simplest type of data structure is a linear array. List of finite no. of similar data elements reference by a set of consecutive no's.

→ Linked list

Linked list contains 2 parts in one node one part contain information and other contain linked information.

→ **Tree**

Tree is hierarchical relationship b/w various elements. The data structure which reflects this relationship is called rooted tree or simple tree.



→ **Stack (LIFO)**

It is also called last in first out (LIFO) system is a linear list which insertion and deletion take place at only one end called top.

→ **Queue (FIFO)**

It is a first in first out (FIFO) system.

It is a linear list which insertion can take place only at the one end of the list called rear. at the deletion can take place only at other end called front.

Data structure operations

4 operation play a major role for all data structure.

→ **Traversing**
Accessing each record exactly once so that certain item in the record may be proceed.

→ **Searching**
Finding the location of the record with the given key-value.

→ **Inserting**
Adding a new record to the data structure is called inserting.

→ **Deleting**
Removing a record from a data structure.

→ **Sorting**
Arranging the records in some logical order.

→ **Merging**
Combining the records of 2 different file into a single file.

ARRAY

UB → upper bound
LB → lower bound.

$n = UB - LB + 1$

ALGORITHM – 1

Traversing of Array.

Here LA is a linear array with lower bound LB and upper bound UB, apply operation PROCESS to each element.

Step 1 : [Initialize counter] set K := LB
Step 2 : Repeat step 3 & 4 while K ≤ UB
Step 3 : [Visit element] Apply PROCESS to LA[K]
Step 4 : [Increment counter] set K := K + 1
[End of step 2 loop]
Step 5 : Exit

ALGORITHM – 2

Inserting an element in linear array

Insert (LA, N, K, value)

Here LA is is linear array with UB & LB, apply PROCESS to each element.

Step 1 : [Initialize counter] set J := N
Step 2 : Repeat step 3 & 4 while J ≥ K

Step3: [Move Jth element downward] Set
LA[J+1] = LA[J]

Step4: [Decrease counter] Set J := J - 1
[End of loop 2]

Step5: [Insert an element] Set LA[K] := Valu

Step 6: [Reset N] Set: N: N+1

Step 7: Exit

★ **ALGORITHM-3**

Deletion an element in array at given location.

DELETE (LA, N, K, VALUE)
(K ↑ location)

Here LA is linear array K is location with UB & LB, apply process to each element.

Step1: Value = LA[K]

Step2: Repeat for J = K to N-1
Set LA[J] = LA[J+1]
[End of loop]

Step3: [Reset the number N element in LA]
Set: N = N-1

Step4: Exit

Single dimension

Formulae to find index value from given array.

$$LOC(LA[K]) = Base(LA) + W(K-1)$$

Two dimension

Column major

$$LOC(A[J,K]) = Base(LA) + W[M(K-1)+(J-1)]$$

Row major

$$LOC(A[J,K]) = Base(LA) + W[N(J-1)+(K-1)]$$

(array index is 0)

LA = Linear array
Base = Base address of array
K = Given location
W = Storage type

M, N → row, column
J, K → Given locations
W → Storage type
Base → Base address of LA.

If M, N are not directly given then apply

$$A[L_1 \text{ ---- } U_1, L_2 \text{ ---- } U_2]$$

$$M = U_1 - L_1 + 1 \qquad N = U_2 - L_2 + 1$$

M = 4 N = 5

f A[4.... 7, -1 ..... 3]

W = 2 bytes

Base = 100

address calculation for A = 6 and 2
A[6, 2]

M = 54

N = 5

W = 2.

LOC (A[6,2]) = 100 (LA) + 2 (4 (2-1) + (6-1))

= 100 (LA) + 18

= 118

LOC (A[6,2]) = 100 + 2 [5 (6-1) + (2-1)]

= 100 + 1

=

Column major.

LOC (A[J,K]) = Base (LA) + W [M(K-L1) + (J-L2)]

Row major

LOC (A[J, K]) = Base (LA) + W [N(J-L1) + (K-L2)]

Recursion

```
#include<stdio.h>

main()
{
  int a, fact;
printf("Enter any number");
 scanf("%d", &a);
  fact = rec(a);
printf("fact value %d", fact);
}

  rec(x)
  int x;
{
   int f;
   if(x==1)
     return(1);
 else
    f = x * rec(x-1);
   return(f);
 }
```

Q Write a program in C for traversing an array.

```
#include<stdio.h>
void main()
{
    int A[] = {2,4,6,8,9};
    int i, n=5;
    printf("Enter 5 elements in an array");
    for(i=0; i<n; i++)
    {
    printf("A[%d]=%d", i, A[i]);
    }
}
```

Q Write a program in C to Insert an element in an array.

```
#include<stdio.h>
void main()
{
  int A[MAX] = {1,2,4,5};
  int N=4
  int i=0
  int index=2
  int value=3
printf("Enter array");
for(i=0; i<N; i++)
  {
  printf("array A[%d]=%d", i, A[i]);
  }
```

```
for (i=N; i>=index; i--)
{  LA[i+1] = LA[i]
}
LA[index] = value;
N++;
printf("Printing array after insertion")
    for (i=0; i<N; i++)
{
 printf("LA[%d] =%d", i, LA[i]);
}
return 0;
}.
```

Scanned with CamScanner

STACK : LIFO

Insertion: PUSH
Deletion: POP

→ Algorithm 1.4 [PUSH operation on stack]
PUSH [STACK, TOP, MAXSTK, VALUE]

Step1: [Stack already filled]
if TOP = MAXSTK then
write OVERFLOW and return
Step2: Set TOP := TOP+1 [Increase top by 1]
Step3: Set STACK [TOP] = VALUE
Step4: Return

→ Algorithm 1.5 [POP operation on stack]
POP [STACK, TOP, VALUE]

Step1: [Stack already empty]
if TOP = 0 then
write UNDERFLOW and return
Step2: set VALUE = STACK [TOP]
Step3: set TOP = TOP-1
Step4: Return

Evaluation of postfix expression

→ Algorithm 1.56
(This algorithm find the value for given arithmetic expression P written in postfix notation)

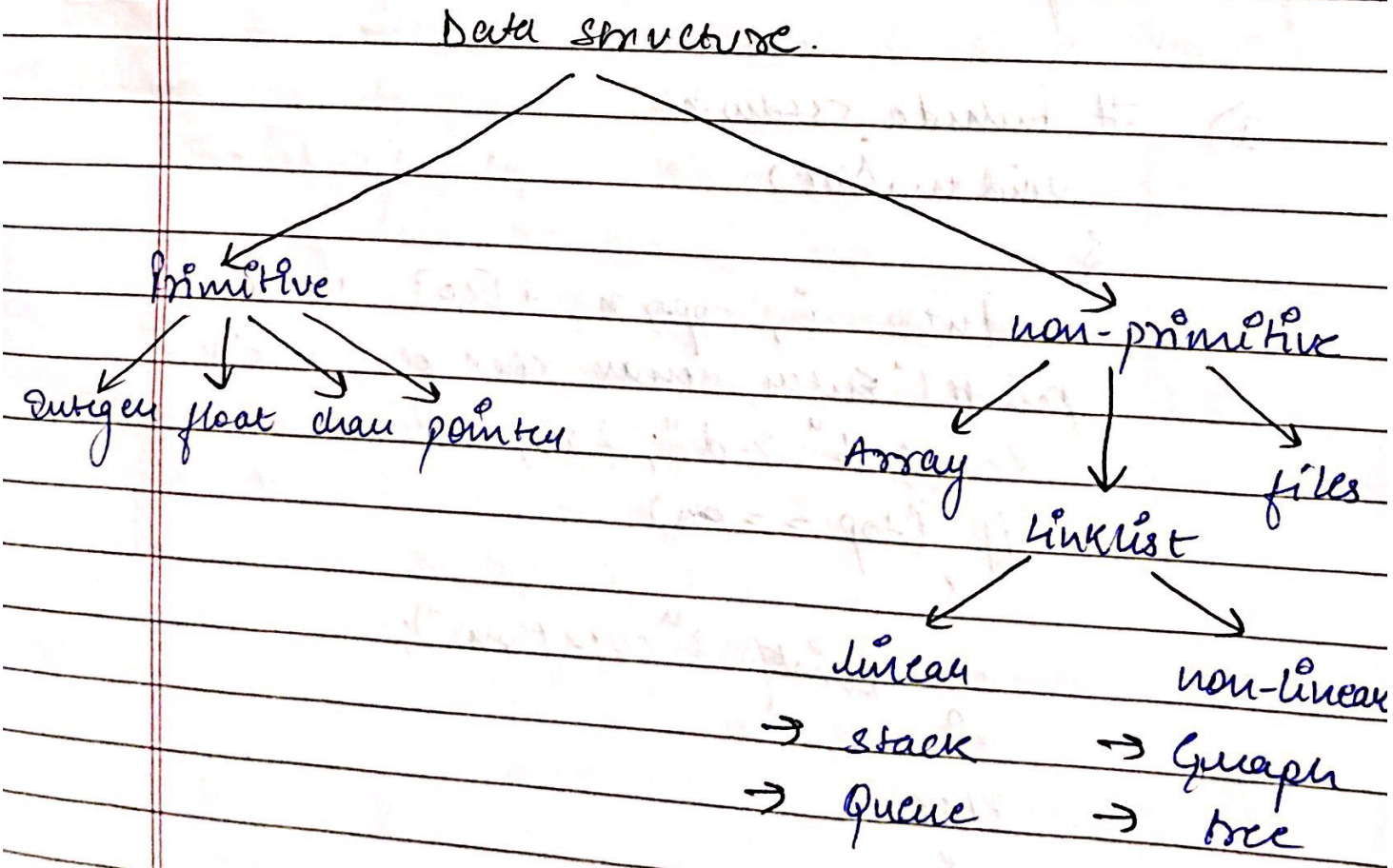
3	5	6	+

Step1: Add right parenthesis ")" at the end of P.
Step2: Scan P from left to right and repeat step3 & 4 for each element of P until the ")" is encountered.
Step3: If operand is encountered Push it on Stack.
Step4: If an operator ⊗ is encountered, then
a) remove the two top element of STACK, where A is top element & B is the next top element
b) Evaluate B ⊗ A
c) Replace the result (b) back on STACK
Step5: SET VALUE equal to the top of STACK
Step6: EXIT.

3, 5, 6, +, *, 7, /

P = 5, 6, 2, +, *, 12, 4, /, −,)

1	5	5			
2	6	5	6		
3	2	5	6	2	
4	+	5	8		
5	*	40			
6	12	40	12		
7	4	40	12	4	
8	/	40	3		
9	−	37			
10	)	Ans 37			



Q Evaluate the postfix expression

1.	9	9
2.	10	9 | 10
3.	3	9 | 10 | 3
4.	*	9 | 30
5.	2	9 | 30 | 2
6.	/	9 | 15
7.	+	24
8.	)	

Q WAP in c for PUSH and POP operation of stack.

a)
```
# include <stdio.h>
void main()
{
    int i, top=0, n, a[20], value;
printf("Enter max size of stack");
  scanf("%d", &n);
  if (top == n)
  {
      printf("overflow");
  }
  else
  {
      top = top+1;
      printf("Enter the value");
      scanf("%d", &value);
```

```
    a[top] = value;
}
for (i=0; i<top; i++)
{
    printf("%d", a[i]);
}
```

Pop

b)
```
# include <stdio.h>
void main()
{
    int i, top=0, n, a[10], value;
    printf("Enter size of stack");
    scanf("%d", &n);
    if (top == n)
    {
        printf("underflow");
    }
else
{
    top = top-1;
    printf("Enter the value");
    scanf("%d", &value);
    value = a[top];
}
for (i=0; i<top; i++)
{
    printf("%d", a[i]);
}
```

+AB → Prefix
A+B → Infix
AB+ → Postfix.

Conversion of infix to postfix notation

Infix = (E) Q = (A+B).

Step 1: Push "(" onto stack and ")" to the end of expression (Q).

Step 2: Scan Q from left to right and repeat Step 3 to 6 for each element of Q until the stack is empty.

Step 4: If left parenthesis is encountered push it onto stack.

Step 3: If operand is encounted, add it to P

Step 5: If an operator ⊗ is encountered then
a) Repeatedly pop from stack and add to P each operator on the top of stack which has the same precedence or higher precedence the ⊗.
b) Add ⊗ to stack.

Step 6: If right parenthesis is encountered then a) Repeatedly pop from stack & add to P each operator until left parenthesis is encountered
b) Remove the left parenthesis

Step 7: Exit.

S.No	stack	Expression (postfix)
C	(C	—
A	(C	A
+	(C+	A
B	(C+	AB
)	(+	AB+
*	(*	AB+
C	(*	AB+C
)		AB+C*

(A+B)/(C-D)

S.No	stack	Expression (postfix)
(	(C	—
A	(C	A
+	(C+	A
B	(C+	AB
)	C	AB+
/	C/	AB+
(	C/(	AB+
C	C/(C	AB+C
-	C/(C-	AB+C
D	C/(C-	AB+CD
)	C/C-	AB+CD-
)		AB+CD-/

A * B + C/D

↑ Power
* Right

SYMBOL	STACK	EXPRESSION (POSTFIX)
A	(C	A
*	(* C*	A
B	(* C*	A B
+	(C+	A B *
C	C+	A B * C
/	C+/	A B * C
D	C+/	A B * C D
)		A B * C D / +

5 * (6+2) − (12/4)

SYMBOL	STACK	EXP.
5	C	5
*	C*	5
(	C C*	5
(	C (*	5,6
+	C C+	5,6 *
2	C C +	5,6 * 2
)	C	5,6 * 2 +
−	C−	5,6 * 2 +
(	C C−	5,6 * 2 +
12	C C−	5,6 * 2 + 12
/	C C−/	5,6 * 2 + 12 , 4
4	C C−/	5,6 * 2 + 12 4 /
)	C C−/	5,6 * 2 + 12 4 / −
)		

Again

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

SNO	Stack	Exp
(	CC	
A	CCA	A
+	CC+	A
C	CC+C	A
B	CC+C	AB
*	CC+C*	AB
C	CC+C*	ABC
)	CC+	A,B,C*
-	CC-	ABC*+
C	CC-C	ABC*+
D	CC-C	ABC*+D
/	CC-C/	ABC*+D
E	CCC-C/	ABC*+DE
↑	CC-C/↑	ABC*+DE
F	CC-C/↑	ABC*+DEF
)	CC-	ABC*+DEF↑/
*	CC*	ABC*+DEF↑/G
G	CC*	ABC*+DEF↑/G
)	C	ABC*+DEF↑/G*
*	C*	ABC*+DEF↑/G*-
H	C*	ABC*+DEF↑/G*-H
)		ABC*+DEF↑/G*-H*

12,4-1

Page

Push Pop

```
#include <stdio.h>
#include <conio.h>
#define maxsize 30
void push();
void pop();
void display();
int stack[Maxsize], top;

void main()
{
clrscr();
int ch=1, top = -1;
while(ch)
{
printf("Push");
printf("pop");
printf("Enter ch your choice");
Scanf("%d", &ch);

Switch(ch)
{
  Case 1: Push();
          break;
  Case 2: pop();
          break;
  Default: exit(0);
}
}

  getch();
```

```
{
void push()
{
    int no;
    if (top == maxsize-1)
{ {
    printf("overflow");
}
    else
{
     top = top+1
     printf("Enter the number");
     scanf("%d", &NO);
      stack[top] = no;
  }
printf("Element in the stack");
       display();
}
  void pop()
{
     int no;
    if (top == -1)
    printf("Underflow");
else
{
    top = top-1
    printf("Element in Stack");
      display();
}
```

```
void display()
{
  int i;
  for (i=0; i<=top; i++)
  {
    printf ("%d", stack[i]);
  }
}
```

Queue

Front → Deletion f=0

Rear → Insertion R=0

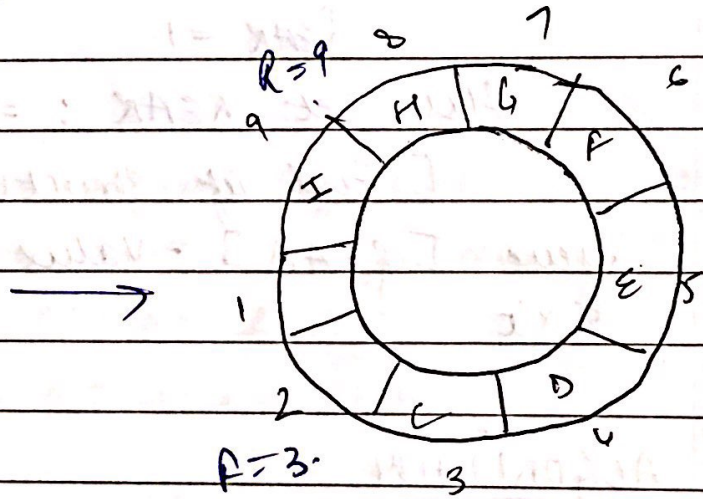
1	2	3	4	5	6	7
10	13	14	15	16	17	18

f=1 f=2 f=3 R=7

R=1 R=2 R=3

f = R = Valid position

Single element.



10 Six array (Linear array)

Insert → 5, 6, 7, 8, 9

delete → 5, 6

Insert → 10, 11, 12, 13, 14

delete → 7, 8, 9

After circle

Insert 15, 16, 17

delete 10, 11

f=8

R=3

5	6	7	8	9

f=6

R=10

ALGORITHM

QUEUE : QINSERT (QUEUE, N, FRONT, REAR, VALUE).

Step 1 : [Queue already filled]
if FRONT = 1 and Rear = N.
print "overflow".

Step 2 : if FRONT = NULL
[Queue is empty]
Set FRONT := 1, REAR := 1
else if REAR = N and FRONT ≠ 1
then
REAR = 1
else set REAR := REAR + 1
[End if structure]

Step 3 : Queue [REAR] = value

Step 4 : Exit

ALGORITHM

QDELETE [QUEUE, N, FRONT, REAR, VALUE

Step 1 : [Queue already empty]
if FRONT = NULL and REAR = NULL
Print "underflow".

Step 2 : Value = Queue [FRONT]

Step 3 : if FRONT = REAR
Set FRONT = NULL, REAR = NULL
else if FRONT = N then

Set FRONT = 1

else

Set FRONT = FRONT + 1

[End if structure]

Step 4: Exit.

Program

QINSERT

```
# include [stdio.h>
 int main ()
{
  int N, QUEUE, FRONT, REAR, VALUE, NULL=
 printf ("Enter the size of queue");
      Scanf ("%d", &N);
      if (FRONT = 1 && REAR = N)
       {
          printf ("overflow");
       }
      else
       {
       if (FRONT = = 0)
         {
            FRONT = 1;
          REAR = 1;
         }
      else if (REAR = N && FRONT ≠ 1)
            {
              REAR = 1;
            REAR = REAR + 1.
```

printf("Enter the value");
scanf("%d", &value);
else Queue[REAR] = value;
printf("%d", Queue[N]);
}

⇒ Program for Queue (circular)

Insertion

```
#include<stdio.h>
#define max 5
int cqueue[MAX];
int front = -1
int rear = -1
void main()
{
void insert();
void del();
void display();
int choice;
while(1)
{
printf("1. Insert");
printf("2. Delete");
printf("3. display");
printf("4. Exit");
printf("Enter the choice");
scanf("%d", &choice);
Switch(choice)
{
case 1: insert();
        break;
case 2: del();
        break;
case 3: display();
        break;
```

```
case 4: exit ();
        break;
default : Printf (" invalid choice");
}}
void Insert ()
{
   Int value;
if (front == 0 && Rear = max -1)
   printf ("overflow");
   return;
   if (front == -1)
{
   front = 0;
   Rear = 0;
}
else if (rear == max-1 && front != 0)
        rear = 0;
else
    rear = rear+1
    printf ("Enter the value");
    Scanf ("%d", &value);
     Queue [rear] = value;
  }
void display ()
{
```

Infix to prefix :

2 * 3 / (2-1) + 5 * (4-1)

SNO	STACK	Exp.
)	)	
)	))	
1	))	1
-	)) -	1,
4	)) -	1, 4
(	)	1 4 -
*	) *	1 4 -
5	) *	1 4 - 5
+	) +	1 4 - 5 *
)	) +)	1 4 - 5 *
1	) +)	1 4 - 5 * 1
-	) +) -	1 4 - 5 * 1
2	) +) -	1 4 - 5 * 1 2
(	) +	1 4 - 5 * 1 2 -
/	) + /	1 4 - 5 * 1 2 -
3	) + /	1 4 - 5 * 1 2 - 3
*	) + / *	1 4 - 5 * 1 2 - 3
2	) + / *	1 4 - 5 * 1 2 - 3 2
(		1 4 - 5 * 1 2 - 3 2 * /

+, /, *, 2, 3, -, 2, 1, *, 5, -, 4, 1

(A + (B * C) - (D / E ↑ F) * G) * H

SNO	Stack	Expression
)	)	
H	)	H
*	)*	H
)	)*)	H
G	)*)	HG
*	)*)*	HG
)	)*)*)	HG
F	)*)*)	HGF
↑	)*)*)↑	HGF
E	)*)*)↑	HGFE
/	)*)*)/	HGFE↑
D	)*)*)/	HGFE↑D
(	)*)*	HGFE↑D/
-	)*)-	HGFE↑D/*
)	)*)-)	HGFE↑D/*
C	)*)-)	HGFE↑D/*C
*	)*)-)*	HGFE↑D/*C
B	)*)-)*	HGFE↑D/*CB
(	)*)-	HGFE↑D/*CB*
+	)*)-+	HGFE↑D/*CB*
A	)*)-+	HGFE↑D/*CB*A
(	)*	HGFE↑D/*CB*A+-
(		HGFE↑D/*CB*A+-*

*-+A*BC*/D↑EFGH

Deletion

```
void delete ()
{
  int value;
if [ front ==0 && rear ==0)
  printf ("underflow");
  return;
if [front == rear]
{
  front =0;
  rear =0;
}
else if [ front == max -1]
{
  front =0;
}
else
{
  front = front +1;
}
Queue [front] = value;
```

Sequence → Selection.

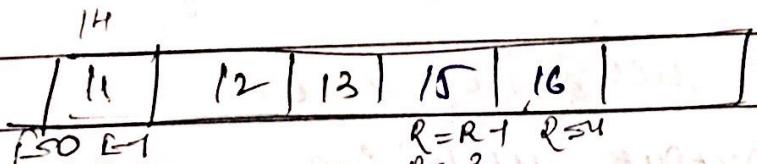
Priority Queue

Priority queue is a collection of element such that each element has been assigned a priority and the order in which elements are deleted and process comes from the following rules :-

(1) An element of higher priority is processed before element of lower priority

(2) 2 elements with same priority are process acc. to the order in which they were added in the queue.

Double ended Queue



11 → delete 11th Position → 14

F=1 R=0 F=0 R=0

After 13th 15, 16

F=0 R=4

16 Delete

R=3 F=0.

Double ended Queue

1. Input restrictive queue
2. Output restrictive queue

Dequeue or Double ended queue

Dequeue is a linear list in which elements can be added or removed at either end but not in the middle.
There are 2 variation in queue.

1. Input restrictive queue
This queue allows insertion at only one end of the list but deletion at both end of the list.

2. Output restrictive queue.
An output restrictive queue is a deque which allow deletion only at one end of the list but insertion at both end of the list.

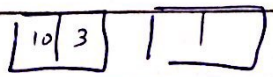
Page

Q

	Info	Link.	Del
1	~~50~~ 50	~~6~~ 2	
2	10	~~4~~ 4	
3	40	~~0~~ X	
4	20	6	
5	50	2	
6	60	~~0~~ 7	
7	30	~~X~~ 3	

start → 5

60' → Link[6]



	Info	Link.
1		
2	~~11~~	~~3~~
3		
4	22	~~8~~ 7
5	~~77~~	*
6	~~44~~	~~7~~
7		
8	66	4 X

start → 5

DYNAMIC MEMORY ALLOCATION

Malloc (Memory allocation)

Dynamic memory allocation can be defined as a procedure in which the size of data structure is changed during the runtime.

Malloc or memory allocation method is used to dynamically allocate the single large block of memory with specified size it returns a pointer of type void which can be cast into a pointer of any form.

for dynamic:- #include <stdlib.h>

ptr = (cast-type *) malloc (byte-size)

example:

ptr = (int *) malloc (100 * sizeof(int));

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int *ptr;
    int n, i;
    n = 5;
    printf("Enter number of elements %d", n);
    ptr = (int *) malloc (n * sizeof(int));
    if (ptr == NULL)
    {
        printf("memory not allocated");
        exit(0);
    }
    else
    {
    for (i=0; i<n; i++)
    {
        ptr[i] = i+1;
    }
    printf("The elements of the array");
    for (i=0; i<n; i++)
    {
        printf("%d", ptr[i]);
    }
    }
    return 0;
}
```

Calloc or contiguous Memory allocation.

Calloc or contiguous allocation is used to dynamically allocate the specified no of blocks of memory of specified type. & initialise each block with a default value 0.

Syntax:

ptr = (cast-type *) calloc(n, element-

example:

ptr = (float *) calloc(25, size of (float)

program - same as Malloc.

Malloc → calloc.

ptr = (int *) calloc(5, size of (in

Free

free method is used to dynamically deallocate the memory the memory allocated using functions malloc & Calloc are not deallocated on their own. hence the free method is used whenever the dynamic memory allocation takes place. It helps to reduce the wastage of memory by freeing it.

```
free (ptr);
```

Realloc

```
ptr = realloc (Ptr, newsize)
```

for exam:

```
ptr = realloc (ptr, n * size of (int))
```

Realloc means reallocation method is used to dynamically change the memory allocation of previously assigned memory. In other words if memory previously allocated with the help of malloc or Calloc is insufficient can be used to dynamically reallocate memory.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *ptr;
    int n, i;
    n = 5;
    printf("Enter number of element %d", n);
    ptr = (int *) calloc(n, sizeof(int));
    if (ptr == NULL)
    {
        printf("Memory not allocated");
        exit(0);
    }
    else
    {
        for (i = 0; i < n; i++)
        {
            ptr(i) = i+1;
        }
        n = 10;
        printf("Enter the number of elements %d", n);
        ptr = realloc(ptr, n * sizeof(int));
        for (i = 5; i < n; i++)
            ptr[i] = i+1;
        for (i = 0; i < n; i++)
        {
            printf("%d", ptr[i]);
        }
    }
```

free (Ptr);
return 0;
}.

output - 12345
1 2 3 4 5 6 7 8 9 10

++
--
+ with integer
- with integer
ptr1 + ptr2 ← not applicable
ptr1 - ptr2 ✓

2-D array
pointer to pointer.

int arr[3][03]; → 0, 1

arr →	2	3	4
arr+1 →	5	6	7
arr+2 →	8	9	1

((arr+i)+j)

7 = *(*(arr+1)+2)

9 = *(*(arr+2)+1).

5 = *(*(arr+1)+0)

*(*arr) = 2.

*(*arr+1) => *((*arr+0)+1) = 3
=> *((*arr)+1) = 3

Link list

Traversing of link list

Traverse [PTR, INFO, LINK]

Step 1: PTR = START
Step 2: Repeat step 3 & 4 while PTR ≠ NULL.
Step 3: Apply Process to INFO [PTR]
Step 4: Set PTR = LINK [PTR]
[End of while loop]
Step 5: Exit.

Insertion at the beginning.

Insert [INFO, LINK, START, VALUE, AVAIL]

Step 1: if AVAIL = NULL, print overflow.
Step 2: Set New = Avail
Avail = Link [Avail]
Step 3: INFO [new] = VALUE.
Step 4: Set LINK [New] = Start.
Step 5: Start = New
Step 6: Exit

creation:

```
#include <stdio.h>
#include <stdlib.h>                                    Y/N
struct node
{
    int info;
    struct node * link
};
    struct node * first;
    void main()
    {
        void create();
        create();  ---> void printf("data entered");
    }                   display();
    void creat()
{
    struct node * ptr, * cpt;
    char ch;
    ptr = (struct node *) malloc (size of(struct node)
printf("Input fixed first node");
    scanf("%d", & ptr -> info);
        first = ptr;
    do
    {
        cpt = (struct node *)malloc(sizeof(struct node));
        printf("Input next node in link list");
        scanf("%d", & cpt -> info);
            ptr -> link = cpt;
            ptr = cpt;
        printf("press 'Y/N for more node");
```

```
        ch = getch();
    }
    while (ch == 'y');
        ptr -> link = NULL;
}
void display()
{
    struct node * ptr;
    printf("Element of Link list");
        ptr = first;
    while (ptr != NULL)
{
    printf("%d", ptr -> info);
        ptr = ptr -> link;
    }
}
```

→ Insert is a link is given location;

Insert LOC (INFO, LINK, START, AVAIL, LOC, VALUE)

Step 1: If AVAIL = NULL Print "Overflow".
Step 2: Set New = Avail and Avail = Link [Avail]
Step 3: Set INFO [NEW] = Value
Step 4: If LOC = NULL then Insert as a first node Set LINK [NEW] = Start and Start = NEW
else
Set LINK [NEW] = LINK [LOC]
LINK [LOC] = NEW
[End if structure]
Step 5: EXIT

Program

```
void Insert_beg()
{
  struct node *ptr;
  ptr = (struct node *) malloc (sizeof(struct node));
  if (ptr == NULL)
  {
    printf ("overflow");
    return;
  }
  printf ("Input new node");
  scanf ("%d", &ptr -> info);
  ptr -> Link = first;
}
```

X Write an algorithm for insert a node in a linklist at the end.

Step 1: If AVAIL = NEW print "Overflow".
Step 2: Set NEW = Avail and Avail = Link[Avail]
Step 3: Set INFO[NEW] = value;
Step 4: If LOC = NULL
Set LINK[NEW] = Start and Start = NEW
else Set LINK[NEW] = NULL
LINK[LOC] = NEW
(End if structure)
EXIT

XX Deletion in a link list :-

DEL LINKList(INFO, START, LINK, AVAIL, LOCP, LOC

Step 1: If LOCP = NULL (i = i+1)
Set Start = LINK[Start]
else
Set LINK[LOCP] = LINK[LOC]
[End if struct]
Step 2: [Return deleted node to Avail link]
Set LINK[LOC] = Avail & Avail = LOC

Step 3: Exit

This algorithm deletes the node & with location LOC. &
LOCP is the location of the node which precedes N.

Program

Insertion at given LOC:

```
void Insert_given_node()
{
  struct node * ptr, * cpt;
  int data;
  ptr = (struct node *) malloc(sizeof(struct node));
  if (ptr == NULL)
  {
    printf("overflow");
    return;
  }
  printf("Input Newnode Information");
  scanf("%d", &ptr->info);
  printf("Input Information of node after
      which Insertion will be made");
  Scanf("%d", &data)
    cpt = First;
  while (cpt->info != data)
    {
      cpt = cpt->link;
```

```
        ptr -> link = cpt -> link;
        cpt -> link = ptr;
    }
}
```