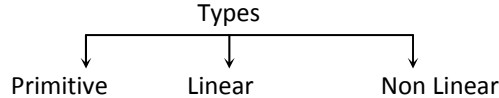


**Q.1(a) Explain different types of data structures with examples.**

[5]

**Ans.:** Types of Data Structure : Data Structures can be categories as follows :



- **Primitive** : They are in build datatypes provided by the programming language. For example : int, float, char etc.
- **Linear Data Structure** : In this all the elements would be at same level. Hence there is hierarchy between elements, for example : Stack, Queues, Array, linked list.
- **Non-Linear Data Structure** : In these the element would be at different levels. Hence there would be hierarchy between the elements. For example : Trees, Graph.

**Operation on Data Structure :**

- |           |           |           |                       |
|-----------|-----------|-----------|-----------------------|
| 1) Create | 2) Insert | 3) Delete | 4) Modify/Edit/update |
| 5) Sort   | 6) Search | 7) Merge  | 8) Traverse           |

**Q.1(b) Write a C program to implement Quick sort.**

[5]

**Ans.:** **Quick Sort (Recursive)**

```

Void quicksort (int x[], int lb, int ub)
{
    int p;
    if (lb < ub)
    {
        p = partition (x, lb, ub);
        quicksort (x, lb, p - 1);
        quicksort (x, p + 1, ub);
    }
}

int partition (int x[], int lb, int ub)
{
    int val = x [ lb ], down = lb+1, up = ub, t;
    while (down <= up)
    {
        while (down <= up && x [down] <= val)
            down ++;
        while (x [up] > val)
            up --;
        if (down < up)
        {
            t = x [down];
  
```

```

        x [down] = x [up];
        x [up] = t;
    }
}
x [b] = x [up];
x [up] = val;
return up;
}

```

**Q.1(c) Write a function for DFS traversal of graph.**

[5]

**Ans.:** Depth first search :

```

#include <stdio.h>
int adj [10] [10] = {0} visited [10] = { 0 }, n :
void dfs (int node)
{
    int i;
    visited[node] = 1
    printf ("%d", node +1);
    for (i = 0 ; i < n ; i ++ )
        if (adj [node] [i] == 1 visited [i] == 0)
            dfs(i);
}

dfs(0)
void main ( )
{
    int e, i, v1, v2, node;
    printf ("Enter no. of nodes \n");
    scanf ("%d", &n);
    printf ("Enter no. of edges \n");
    scanf ("%d", &e);
    for (i = 0; i < e ; i ++ )
    {
        printf ("Enter edge \n");
        scanf ("%d%d", &v1, v2);
        adj [v1 - 1] [v2 - 1] = adj[v2 - 1] [v1 - 1] = 1
    }
    printf ("Enter starting vertex \n")
    scanf ("%d", &nodes);
    dfs (node-1);
}

```

**Q.2(a) Write a C program to convert infix expression into postfix Expression.**

[10]

**Ans.: Implementation :**

```

#include<stdio.h>
#include<string.h>
#define MAX 50

```

```
type def struct stack
{
    char S[MAX];
    int tos;
} stack;
void push (stack * t, char ele)
{
    t → s [ + t → tos ] = ele;
}
int isempty (stack*t)
{
    return (t → tos == -1);
}
char pop (stack * t)
{
    char z = t → s [ t → tos ];
    t → tos -- ;
    return z;
}
char stacktop (stack * t)
{
    return (t → s [ t → tos ];
}
int isoperand (char ch)
{
    if (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z')
        return 1;
    else return 0;
}
int ipr (char ch)
{
    switch (ch)
    {
        case 'c' : return 3;
        case '*' :
        case '/' :
        case '%' : return 2;
        case '+' :
        case '-' : return 1;
    }
}
return -1;
}
int rpr (char ch)
{
    switch (ch)
    {
        case 'c' : return 0;
```

```

        case '*' :
        case '/' :
        case '%' : return 2;
        case '+' :
        case '-' : return 1;
    }
    return -1 ;
}
void convert (char in[ ] , char post [ ])
{
    stack x ;
    int l = strlen (in), i, k = 0;
    char ele;
    x -tos = -1;
    for (i=0 : i < l : i++)
    {
        if (in[i] == 'c')
            push (2x, in [i]);
        else if (in[i] == '(')
        {
            while (1)
            {
                ele = pop (&x);
                if (ele == 'C')
                    break;
                post [k++] = ele;
            }
        }
        else if (is operand (in [i]))
            post [k++] = in [i]
        else if (isempty (&x))
            push (&x, in [i]);
        else if (ipr (in[i]) > rpr [stacktop (&x)])
            push (&x, in[i]);
        else
        {
            while (isempty (&x) == 0 && ipr (in[i] < rpr (stacktop(&x)))
            {
                ele = pop (& x);
                post [k++] = ele;
            }
            push (&x, in [i]);
        }
    }
    while (isempty (&x) == 0)
    {
        ele = pop (&x);
    }
}

```

```

        post [k++] = ele;
    }
    post [k] = '\0';
}
void main ( )
{
    char infix [MAX], postfix [MAX] ;
    printf ("\n \n Enter infix expression:");
    gets (infix);
    convert (infix, postfix);
    printf ("Postfix = % \n", postfix);
}

```

**Q.2(b) Write a function to implement Binary Search on sorted set of Integer. [10]**

**Ans.:** Implementation: (Non Recursive)  
 int binarysearch (int x[], int n, int ele)

```

{
    int l = 0; h = n - 1, m;
    while (l <= h)
    {
        m = (l + h)/2;
        if (ele == x [m])
            return 1;
        if (ele < x [m])
            h = m - 1;
        else l = m + 1;
    }
    return 0;
}

```

**Q.3(a) Write a program in C to implement circular queue using always. [10]**

**Ans.:** Array implementation of circular Queue :

```

#include <stdio.h>
#define MAX 4
typedef struct queue
{
    int q [MAX];
    int f, r, count;
} queue;

void insert (queue *t, int ele);
{
    if (t->count == MAX)
    {
        printf ("\n queue overflow");
        return;
    }
}

```

```
t → count ++;
t → r(t → r+1) % MAX;
t → q [t→r] = ele ;
}
int isempty (queue *t)
{
    if (t→count == 0)
        return 1;
    else return 0;
}
int delete1(queue * t)
{
    int z;
    if (isempty (t))
    {
        printf ("Queue underflow");
        return -1;
    }
    t → count -- ;
    z = t → q [ t →f];
    t → f = (t → f + 1) % MAX;
    return z;
}
void display (queue *t)
{
    int i;
    if (isempty (t))
    {
        printf ("Queue empty \n");
        return;
    }
    i = t → f;
    while (1)
    {
        printf ("% d", t → q [i] );
        if (i == t → r )
            break;
        i = (i + 1) % MAX ;
    }
    printf("\n");
}
void main( )
{
    queue x;
    int ch, ele;
    x.f = 0;
```

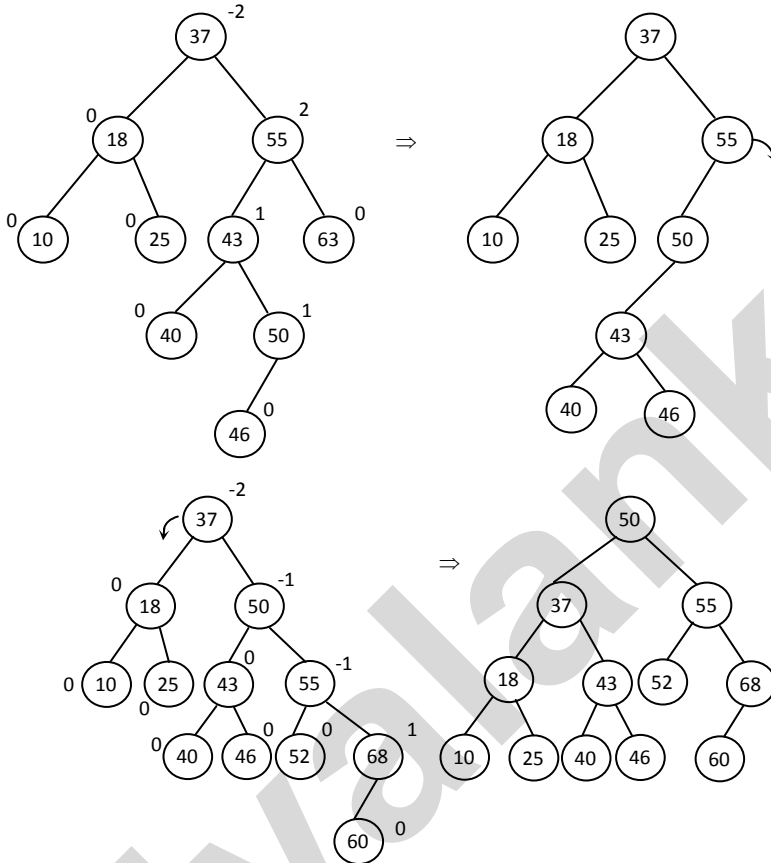
```
x.r = MAX -1;
x.count = 0;
while (1)
{
    printf ("Enter choice 1] Insert, 2] Delete");
    printf ("3] Queue front 4] Display, 5] Exit");
        \n");
    scanf ("%d", &ch);
    if (ch ==5)
        break;
    switch (ch)
    {
        case 1 : printf ("Enter element to be inserted");
                scanf ("% d", & ele);
                insert (&x ele);
                display (&x);
                break,
        case 2 : if (isempty (&x))
                printfff ("\n Queue underflow");
                else
                {
                    ele = delete 1(& x);
                    printf ("Deleted element = "%d \n", ele) ;
                    display (& x);
                }
                break;
        case 3 : if (isempty (&x))
                printfff ("\n Queue empty");
                else
                {
                    ele = queue front (& x);
                    printf ("Queue front =%d" \n, ele);
                    display ( & x);
                }
                break;
        case 4 : display (& x);
                break,
        default: printf ("\n Invalid choice");
    }
}
```

**Q.3(b) Construct AVL Trees using following.**

**[10]**

**37, 55, 18, 25, 68, 10, 43, 50, 40, 46, 52, 60**

**Ans.:** 37, 55, 18, 25, 68, 10, 43, 50, 40, 46, 52, 60



**Q.4(a) Construct huffman code for the symbols of message ACBADACA.**

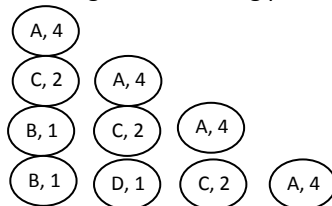
**[10]**

**Ans.:** Construct codes for the symbols of the message ACBADACA

Step 1: Constructing frequency table:

A	4
C	2
B	1
D	1

Step 2: Initialising it in ascending priority queue





**Q.4(b) Write a function to implement singly Linked list following operations :** [10]  
**(i) Insert a node at specific Location**      **(ii) Delete a node from end**  
**(iii) Display the list**

**Ans.:** General Linked List (Singly Linked List):

Implementation:

```
# include <stdio.h>
# include <stdlib.h>
typedef struct node
{
    int data;
    struct node * next;
}node;
```

```
typedef struct head
{
    node * start;
}head;
```

```
void insertpos (head*t, int ele, int pos)
{
    int i;
    node * p, *q;
    p = (node *) malloc (size of (node));
    p → data = ele;
    q = t → start;
    for (i = 0; i < pos - 2; i++)
        q = q → next;
    p → next = q → next;
    q → next = p;
}
```

pos	pos - 2
2	0
3	1
4	1

```
void display (head * t)
{
    node * q;
    if (t → start == NULL)
    {
        printf ("Linked List empty \n");
        return;
    }
    q = t → start;
    while (q != NULL)
    {
        printf ("%d", q → data);
    }
}
```

```

        q = q → next;
    }
    printf (“\n”);
}

int delete end (head * t)
{
    node * p, * q;
    if (t → start == NULL)
    {
        printf (“LL Empty \n”);
        return - 1;
    }

    if (t → start → next == NULL)
    {
        p = t → start;
        t → start = NULL;
    }
    else
    {
        q = t → start;
        while (q → next → next != NULL)
            q = q → next;

        p = q → next;
        q → next = NULL;
    }
    return (p → data);
}

```

**Q.5(a) Write a C function for following operations on Double Linked list**

**[10]**

**(i) Insert Beginning**

**(ii) Delete Beginning**

**Ans.:**

```

typedef struct node
{
    int data;
    struct node * next, * prev;
} node;
typedef struct head
{
    node * start * last;
} head;
void insertbeg (head*t, int ele)
{
    node * p;
    p = (node *) malloc (size of (node));

```

```

p → data = ele;
p → next = p → prev = NULL;
if (t → start == NULL)
    t → start = t → last = p;
else
{
    p → next = t → start;
    t → start → prev = p;
    t → start = p;
}
}

```

```

int deletebeg (head*t)
{
    node * p;
    if (t → start == NULL)
    {
        printf ("DLL Empty \n");
        return - 1;
    }
    p = t → start;
    if(t → start == t → last)
    if (t → start = t → last = NULL;
    else
    {
        t → start = t → start → next;
        t → start → prev = NULL;
    }
    return (p → data);
}

```

**Q.5(b) Write a C function to perform Deletion operation in Binary Search Tree.**

**[10]**

**Ans.:**

```

void delete1 (head * t, int ele)
{
    node * q, * f;
    if (t → root == NULL)
    {
        printf ("BST Empty \n");
        return;
    }
    q = t → root;
    while (q != NULL)
    {
        if (ele == q → data)
            break;
        if (ele < q → data)

```

```
        q = q → left;
    else    q = q → right;
}
if (q == NULL)
{
    printf ("% d is not found \n", ele);
    return;
}
if (q → left == NULL && q → right == NULL)
{
    if (q == t → root)
    {
        t → root = NULL;
        return;
    }
    f = father (t, q);
    if (f → left == q)
        f → left = NULL;
    else f → right = NULL;
    return;
}

if (q → left != NULL)
{
    int val;
    node * max = findmax (q → left);
    val = max → data;
    delete 1(t, max → data);
    q → data = val;
    return;
}
if (q == t → root)
{
    t → root = t → root → right;
    return;
}
f = father (t, q);
if (f → left == q)
    f → left = q → right;
else f → right = q → right;
}
```

**Q.6 Attempt following (Any 2) [20]**

**Q.6(a) Explain B Tree & B+ Tree. [10]**

- Ans.:**
- B tree stands for Balanced Tree
  - In B tree of order n(odd) :
    - each node can contain
    - max data = n-1
    - maximum no. of children = n
    - No. of children = no. of data + 1
    - min data = n/2 (except root)

- B + Tree :
  - In B Tree the element are stored on all the levels.
  - Hence the record pointers must be kept at all the levels and linear search in the leaf nodes is also not possible.
  - In achieve this we must have all the elements in leaf nodes only.
  - This can be done by performing following modification in B Tree.
    - (i) When leaf node is split, the median is sent up and its copy is also retained in left child.
    - (ii) All the leaf nodes must be linearly connect.
  - A B Tree with this modification is called B+ Tree

**Q.6(b) Explain : [10]**

- (i) Various methods to represent graph in computer memory
- (ii) Explain different Applications of Linked List

**Ans.:** (i) Graph Representation / Representing graph in memory

Graph can be represented in memory using any of the following method.

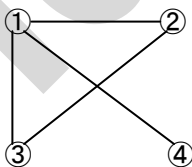
- 1) Adjacency matrix
- 2) Adjacency List

**Adjacency Matrix :**

- In this, a graph with n vertices can be represented using a square matrix of order n.
- The adjacency between the nodes can be stored using zeros & is as follows :

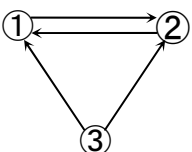
$$\text{adj}(i, j) = \begin{cases} 1 & \text{if } (i, j) \in E(G) \\ 0 & \text{if } (i, j) \notin E(G) \end{cases}$$

Undirected :



$$\begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

Directed :

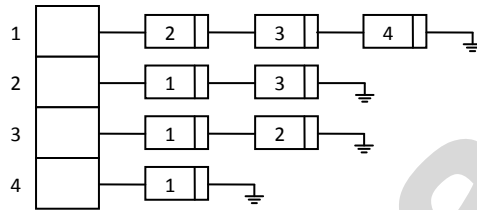
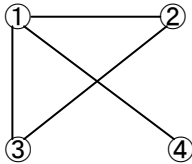


$$\begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

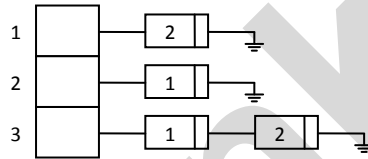
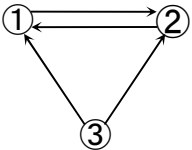
**Adjacency List :**

- In these the graph with n vertices is stored using an array of n linked list.
- Hence 1 node is mapped with n linked list which can be used to store all the nodes adjacent to that node.

Undirected :



Directed :

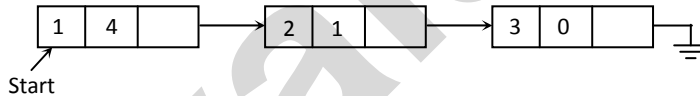


**(ii) Application of Linked List :**

Polynomial representation & addition.

- A Polynomial can be represented using linked list.
- Each node can store co-efficient, exponent and pointer to the next node.

e.g. :  $x^4 + 2x + 3$



**Q.6(c) Explain a c program to implement Double Ended Queue.**

[10]

```

Ans.: # include <stdio.h>
# define MAX 7
typedef struct queue
{
    int q [MAX]
    int f, r;
} queue;

void insert right (queue *t, in tele)
{
    if (t->r == MAX-1)
    {
        printf ("\n Queue overflow");
        return;
    }
    t->r++;
    t->q [t->r] = ele;
}
    
```

```
void insert left (queue *t, int ele)
{
    if (t → f == -1)
    {
        printf (“\n Queue overflow”);
        return;
    }
    t → q [t → f] = ele;
    t → f ++;
}
```

```
int isempty (queue * t)
{
    if (t → f == t → r)
    {
        return 1;
        else
        return 0;
    }
}
```

```
int deleteright (queue *t)
{
    int z;
    if (isempty (t))
    {
        printf (“\n Queue underflow”);
        return -1;
    }
    z = t → q [t → r];
    t → r --;
    return z;
}
```

```
int deleteleft (queue * t)
{
    if (isempty (t))
    {
        printf (“Queue underflow”);
        return -1;
    }
    t → f ++;
    return (t → q [t → f]);
}
```

```
void display (queue * t)
{
```

```

int i
if (isempty (t))
{
    printf ("Queue Empty");
    return;
}
for (i = t → f + 1 ; i < t → r ; i ++ )
printf (" % d" t → q [ i ] );
printf("\n");
}

void main ( )
{
    queue x;
    int ch, ele;
    x.f = x.r = MAX 12;
    while (1)
    {
        printf ("Enter choice : 1] Insert right, 2] Insert left");
        printf ("3] Delete right, 4] Delete left, 5] Display");
        printf ("6] exit");
        scanf ("%d", &ch)
        if (ch == 6)
            breaks;
        switch (ch)
        {
            Case 1 : printf ("Enter element");
                    scanf ("%d", & ele);
                    insertright (& x, ele);
                    display ( & x);
                    break,
            Case 2 : printf ("Enter element");
                    scanf ("%d", & ele);
                    insertleft (&x, ele);
                    display (& x);
                    break;
            Case 3 : if (isempty ( &x))
                    printf ("\n Queue underflow")
                    else
                    {
                        ele = deleteright(&x);
                        printf "Deleted element = %d", ele");
                        display(&x);
                    }
                    break;
            Case 4 : if (isempty (&x))
                    printf ("Queue underflow")

```



```
else
{
    ele = deleteleft (&x);
    printf ("Deleted element = % d", ele);
    display (&x);
}
break ;
Case 5 : display (&x);
break;
default: printf ("Invalid choice")
}
}
}
```

□□□□□