DATA STRUCTURES USING C

Assignment - Set I   PAPER- II

1. Write notes on various operators, Conditional and Looping Statements available in C With Suitable Syntax and example.

operators in c are symbols or tokens that perform operations on operands. operands can be variable, constants, expressions, or function calls. operators allow you to manipulate data and perform various tasks like arthmetic operations, Comparisons, Logical operations, bitwise operations, and more.

C language supports wide range of operators categorized into different types based on their functionality and usage. These operators play a crucial role in writing efficent and expressive c programs.

Arithmetic operators: These are used to perform mathematical operations.

+ Addition
- Subtraction
* Moltiplication
/ Division
% modulus Division

Realational operators: These are used to compare values.

==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Logical operators These are used to perform Logical operations.

&&	Logical AND
||	Logical OR
!	Logical NOT

Assignment operators These are used to assign value to variables.

=	Simple assignment
+=	Add and assign
-=	Subtract and assign
*=	multibly and assign
/=	Divide and assign
%=	modulus and assign

Increment and Decrement operators

++	increment by 1
--	Decrement by 1

Bitwise operators: These operate on bits and perform bit-level operations.

&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left Shift
>>	Right Shift

Conditional operator (Ternary operator) It's a shorthand for if-else statements.

```
Condition ? expression1 : expression2
```

Comma operator It evaluates multiple expressions and return the value of the last expression.

```
expr1, expr2
```

Size of operator It returns the size of a variable or data type.

```
Sizeof(type)
```

Conditional Statement (Syntax and example)

Syntax   condition? statement1: statement2;

ex:-

```
a>b ? printf("A is Big\n"): printf("B is Big\n");
```

Looping Statement (Syntax and example)

- While Loop

Syntax:

```
        While(condition)
    {
      Statement 1;
     Statement 2;
     ....
     increment /decrement;
  }
```

ex:- Print all integers from 1 to 40

```
#include<stdio.h>
#include<conio.h>
void main()
{
 Int i=1;
 While(i<=40)
 {
  printf("%d \n", i);
   i++
 }}
```

- do While Loop

Syntax :
```
do
{
 Statement body;
}
While(Condition);
```

example:- To read in a number from the keyboad untill a value in the range 1 to 10 is entered.

```
int i;
do
{
Scanf("%d\n", &i);
 -flushall();
}
While( i<1 && i>10);
```

- for Loop

Syntax:
```
for([initialisation]; [condition]; [increment])
        [Statement body];
```

example :- print out all numbers from 1 to 100

```
#include<stdio.h>
 void main()
 {
  int x;
 for (x=1; x<= 100; x++)
     Printf("%d \n", x);
 }
```

(2.) Discuss briefly about Functions, call-by-value, call-by-reference and Recursion.

In C arguments are passed to functions using the call-by-value (CBV) scheme. This means that compiler copies the value of the arguments passed by the calling function into the formal parameter list of the called functions. Thus if we change the values of the formal parameters within the called function we will have no effect on the calling arguments. The formal parameter of a function are thus local variable of the function are created upon entry and destroyed on exit.

for example program to add two numbers (CBV)

```
#include <stdio.h>
int add (int, int);
void main()
{
int x, y;
 puts("Enter two integers");
Scanf("%d %d", &x, &y);
printf("%d + %d = %d\n", x, y, add(x,y));
}
int add (int a, intb)
 {
  int result;
  result = a+b;
  return result;
 }
```

The add() function here has three Local variable, two formal parameters and the variable result. There is no connection between the calling arguments, x and y, and the formal parameters a and b other than that the formal parameter are initialized with the value in the calling arguments when the function is invoked. The situation is depicted below to emphasize the independence of the various variable

for example:- program that attempts to swap the value of two number (call-by-Reference)

```
#include <stdio.h>
```

```
void swap(int* ,int*);

void main()
{
 int a,b;

  printf("Enter two numbers");
  Scanf(" %d %d", &a, &b);
  Printf("a= %d", b=%d \n",a,b);
  Swap(&a, &b);
  print ("a= %d; b= %d \n", a,b);
 }
  void swap(int *x, int *y)
  {
   Int t;
   t = *x;
   *x= *y;
   *y= t;
 }
```

Since c uses call by value to pass parameters What we have actually done in this program is to swap the value of the formal parameters but we have no changed the value in main(). Also since we can only return one value the return statement we must find some other means to alter the value in the calling function.

Recursion

A recursive function is a function that calls itself either directly or indirectly through another function. Recursive function calling is often the simplest method to eventually simplified into a series of more basic operations of the same type as the original complex operation.

This especially true of certain types of mathematical functions. for example to evalute the factorial of a number, n

n! = n* n-1 * n-2 * ..... *3*2*1.

```
int Fact (int n)
{
if (n<=1)    //terminating condition
    return 1;
   else
   return (n* Fact (n-1));
}
```

we can simplify this operation into

n! = n* (n-1)!

where the original problem has been reduced in complexity slightly. we continue this process

Untill we get the problem down to a task that may be solved directly. In this case as far as evaluating the factorial of 1 which is simply 1.

So a recusive function to evaluate the factorial of a number will simply keep calling it self until the argument is 1. All of the previous (n-1) recursive calls will still be active waiting until the simplest problem is solved before the more complex intermediate steps can be built back up giving the final solution.

3 Write notes on Stacks, Queues and programs to implement all operations on Stacks and Queues.

Stack is a Linear data structure (an ordered List) in which all insertions and deletions are performed through a single end called "top". Stack is also called as a LIFO (Last in First out) List or a FILO (First in Last out) List.

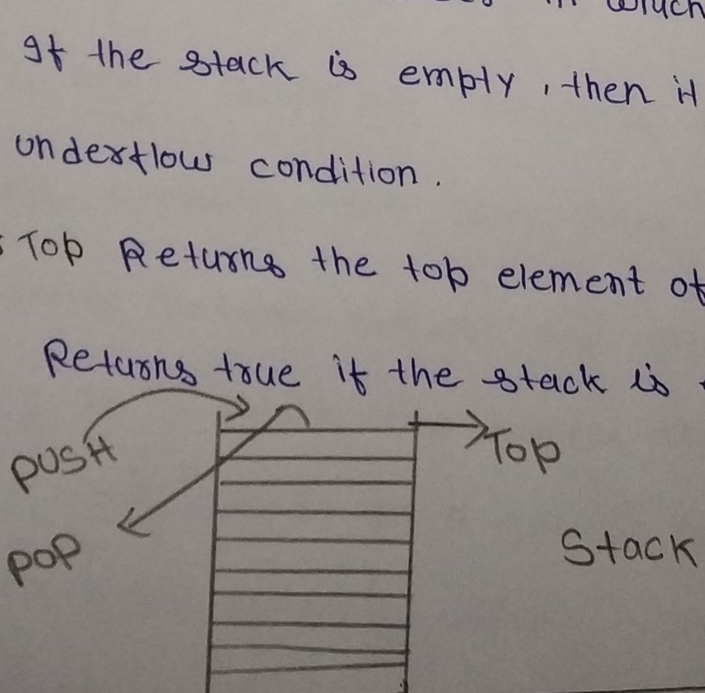
The following are the basic operations performed in the stack

Push Adds an Item in the Stack. If the stack is full, then it is said to be an overflow condition.

Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an underflow condition.

Peek or Top Returns the top element of the stack.

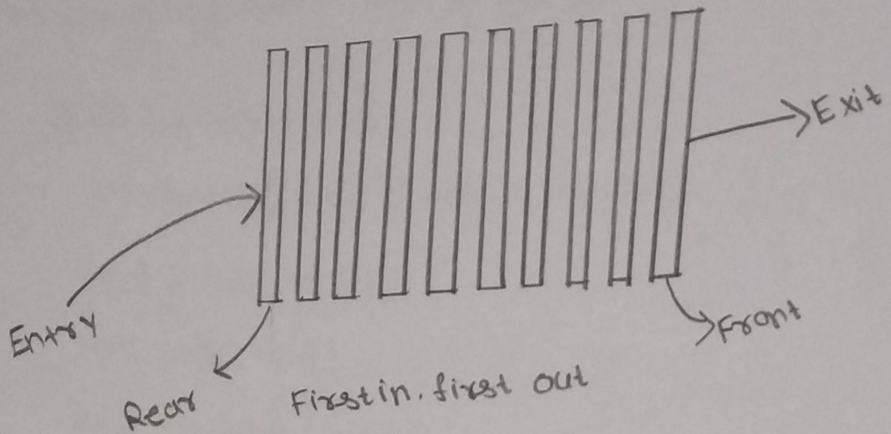
isEmpty Returns true if the stack is empty, else false



There are many real life examples of Stack. consider the simple example of plates stacked over one another in a canteen. The plate which is at the top is the first one to be removed i-e the plate which has been placed at the bottommost position remains in the stack for the longest period of time. so it can be simply seen to follow the LIFO/FILO order.

Queue Data Structure like Stack

Queue is also a linear structure (an ordered list), in which all insertions are performed from the "rear" end and all deletions are performed from the "front" end. A Queue is also called as a First In First out (FIFO) list. A good example of queue is any queue of consumers for a resourse where the consumer that came first is served first The difference between stacks and queues is in removing In a stack we remove the item the most recently added; In a queue, we remove the item the least recently added.



Operations on Queue

The following are the basic operation performed on queue

Inset/Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an overflow condition

Delete/Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. if the queue is empty, then it is said to be an underflow condition.

Front: Get the front item from queue.

Rear Get the last Item from queue.

Implementation

There are two ways to implement a Stack

- using Array
- using linked List

Implementing Stack using Arrays

```
#include<stdio.h>
  int Stack[100], choice,n, top, x, i;
 void push(void);
 void pop(void);
 void display (void);
 int main()
 {
  top =-1;
  printf("\n Enter the size of STACK [max =100]:");
  Scanf("%d", &n);
 printf("\n\t STACK OPERATIONS USING ARRAY");
 printf("\n\t-------------------");
 printf("\n\t 1. PUSH\n\t 2. POP\n\t 3. DISPLAY\n\t
                                    4. EXIT");
do
{
  printf("\nEnter the choice:");
  Scanf("%d, &choice);
  Switch(Choice)
  {
    case 1: push();
            break;
    Case 2: pop();
            break;
   Case 3: display();
           break;
  Case 4; printf("\n\t EXIT POINT");
          break;
 default: printf("\n\t plese Enter a valid choice");

    }
  }
  While (choice != 4);
     return 0;
 }
 void push()
 {
  if (top >= n-1)
  {
   printf("\n\tstAck is FULL, Push operation not
                         possible \n");
  }
  else
  {
   printf("Enter a value to be pushed: ");
   Scanf("% d", &x);
   Stack [++top]=x;
```

```
}
}
void pop()
{
 if (top<=-1)
  {
  printf("\n\t Stack is empty\npop operation not
                              possible \n");
 }
 else
 {
  printf("\n\t The popped elements is %d, stack[top--]);
 }
}
void display()
{
 if (top>=0)
 {
  printf("\n The elements in STACK \n");
  for(i=top; i>0; i--)
      printf("\n%d", Stack[i]);
    printf("\n Press Next choice ~~The STACK is empty~~");
  }
  else
  {
    printf("\n The STACK is empty");
  }
  }
```

OUTPUT

```
Enter the size of STACK[MAX =100]:10

    STACK OPERATIONS USING ARRAY
    ---- ---- ------- ----- ----

 1. PUSH
 2. POP
 3. DISPLAY
 4. EXIT

   Enter the choice:1
   Enter a value to be pushed:2
   Enter the choice:1
   Enter a value to be pushed:24
   Enter the choice:1
   Enter the value tobe pushed:98
   Enter the choice:3
 The elements in STACK
 98
 24
 12
 Press Next choice
 Enter the choice:2
        The popped elements is 98
   Enter the choice :3
                                   Press Next choice
 The elements in STACK
  24                               Enter the choice:4
  12
```

Implementing Stack using Linked List

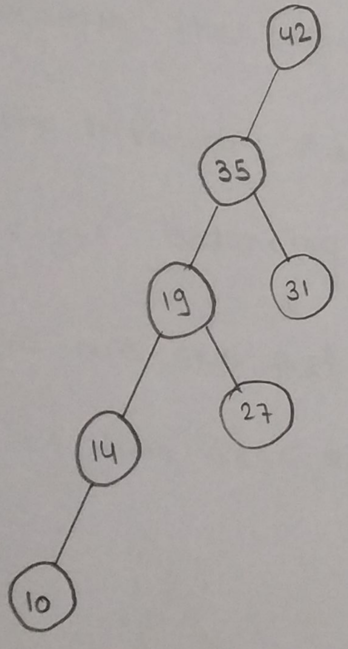
```
# include <stdio.h>
# include <Stdlib.h>

// Define a Structure for each element of th stack

Struct StackNode * next;
};

// Function to create a new node struct StackNode* newNode
                                              (int data)
{
Struct StackNode* StackNode=(Struct StackNode*)
   malloc(Sizeof(Struct StackNode));
   StackNode->data = data;
   StackNode->next = NULL;
   return StackNode;
}
int isEmpty(Struct StackNode *root)
{
 return !root;
}
void push(Struct StackNode = newNode(data);
   StackNode->next=*root;
   *root=StackNode;
Printf ("%d pushed to stack\n", data);
 }
int pop (Struct StackNode** root)
```

```
{
if (isEmpty(*root))
  {
  printf("Stack underflow\n");
  return-1;
 }
struct StackNode * temp = * root
 * root = (*root)->next;
 int popped = temp->data;
 free(temp);
 return popped;
}

 void display(struct Stack Node*root)
{
 if (isEmpty (root)){
  printf("Stack is Empty\n");
  return;
 }
 struct stackNode * temp=root;
 printf(" Stack elements are:\n");
 While (temp != NULL){
  printf("%d\n ", temp->data);
   temp= temp->next
  }
 }
 Int main()
 {
```

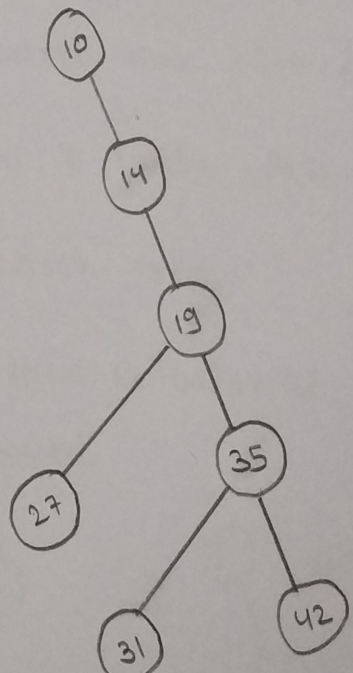
```
Struct StackNode * root = NULL;

 push (&root, 10);
 push (&root, 20);
 push (&root, 30);

 display (root);

 printf ("%d popped from stack\n", pop (&root));

 printf ("%d popped from stack\n", pop (& root));

 display (root);
 return0;
}
```

4. Briefly discuss about AVL Tree and the insertion operation on AVL Tree with suitable Rotations and examples.

AVL Tree if the input to binary search tree comes in a sorted (ascending or descending) manner? it will then look like this



it input 'appears' non-increasing manner



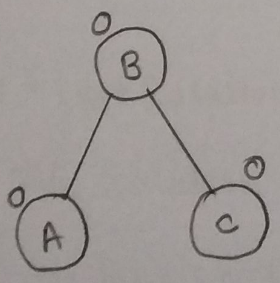
it input 'appears' in non-decreasing

It is observed that BST's worst-case performance is closest to Linear Search algorithms, that is O(n) In real-time data, we cannot predict data pattern and their frequencies So, a need arises to balance out the existing BST.

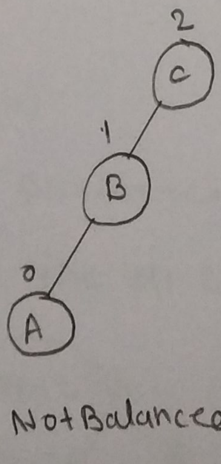
AVL trees are binary search trees in which the difference between the hight of the left and right Sub tree is either -1, 0, or +1. This difference is called the Balance Factor.

AVL trees are also called a self-bal-ancing binary search tree. These trees help to maintain the Logarithmic search time. Named after their inventor Adelson. velski & Landis. AVL tree are height balancing binary search tree.

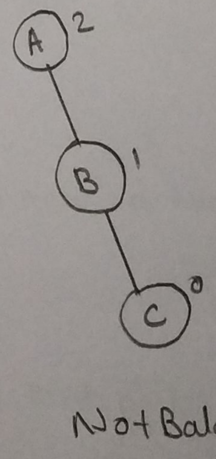
Here we see that the first tree is balanced and the next two tree are not balanced.



Balanced



NotBalanced



NotBalanced

In the second tree, the left subtree of C has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of A has height 2 and the left is missing, so it is 0, and the difference is 2 again

AVL tree permits difference (balance factor) to be only 1.

BalanceFactor = height (left - subtree) - height (right - subtree)

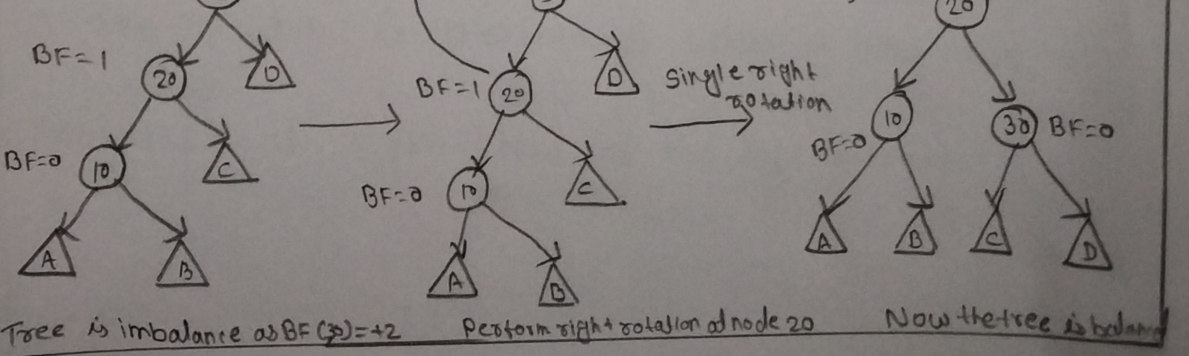
AVL Roatations

To make the AVL Tree balance itself, when inserting or deleting a node from the tree, rotations are performed.

we performed the following LL rotation, RR rotation. LR rotation, and RL rotation

Left - Left Rotation (LL)

This rotation is performed when a new is inserted at the left child of the left subtree.

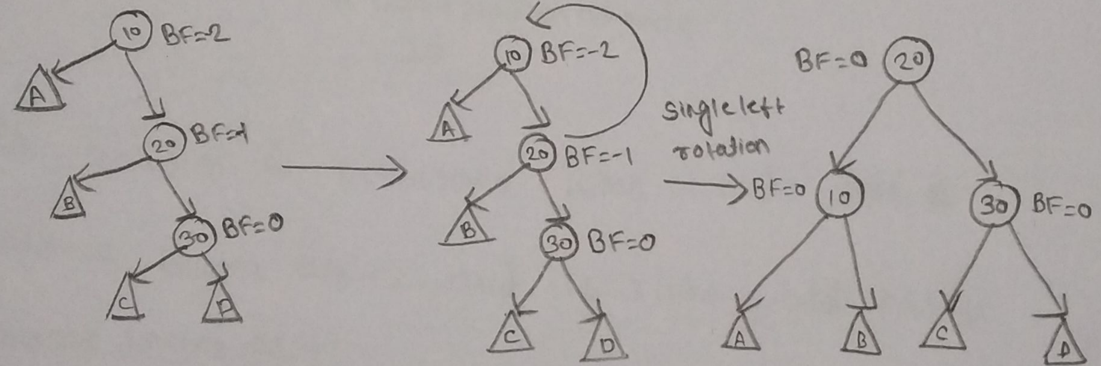


Tree is imbalance as BF (30) = +2   Perform right rotation at node 20   Now the tree is balanced

A single right rotation is performed. This type of rotatin is identified when a node has a balanced factor as +2. and its left-child has a balance factor as +1

Right - Right Rotation (RR)

This rotation is performed when a new node is inserted at the right child of the right subtree.



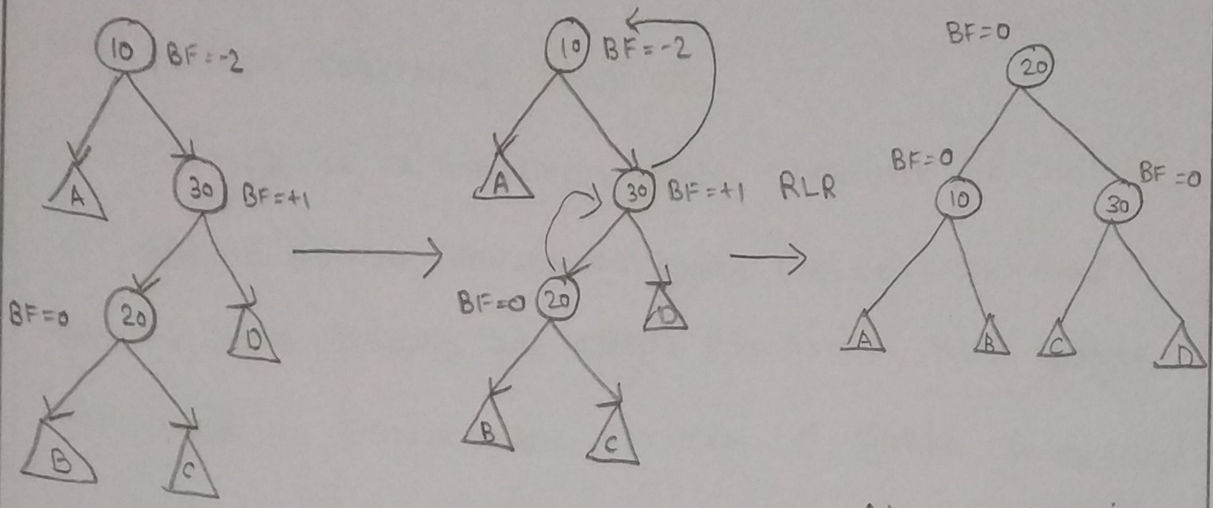
Tree is imbalance as BF(10)=-2

Perform a left rotation at node 20

Now the tree is balanced

Right -'Left Rotation (RL)

This rotation is performed when a new node is inserted at right child of the left subtree.



Tree is imbalance as BF(10)=-2

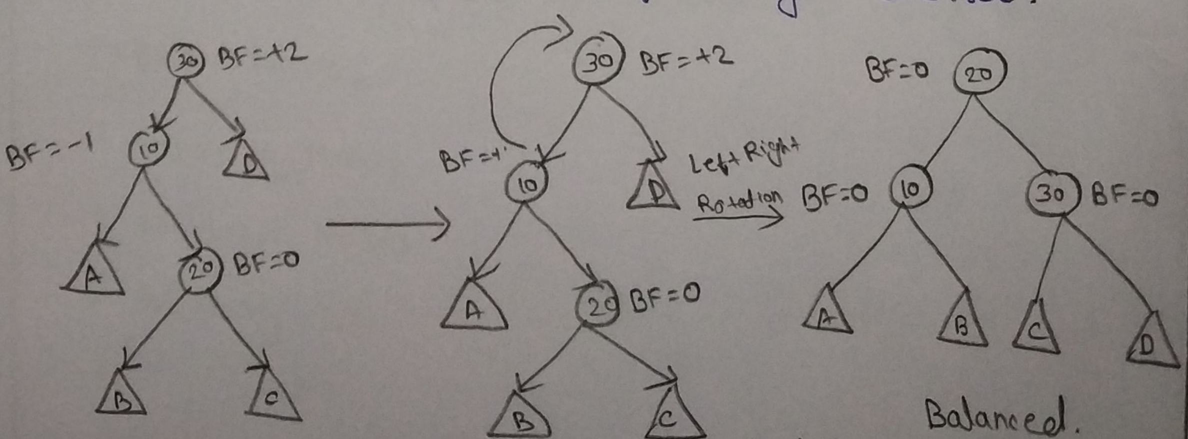
perform a right rotation at node 20 & a left rotation at node 30

Now the tree is balanced.

This rotation is performed when a node has a balance factor as -2, and its right child has a balance factor as +1

Left - Right Rotation (LR)

This rotation is performed when a new node is inserted at the left child of the right subtree.



Tree is imbalance as BF(30)=-2 node 20 & a right at node 10

Balanced.

5 Discuss in detail about Graph, Graph Representations and Graph Traversal Technique.

Graph is a non-Linear data structure. It contains a set of points knows as nodes (or vertices) and a set of links knows as edges (or Arcs). Here edges are used to connect the vertices. A graph is defined as follow

Graph is a collection of vertics and arcs in which vertices are connected with arcs.
Graph is a collection of nodes and edges in which nodes are connected with edges

Generally, a graph G is represented as G = (V, E) where V is set of vartices and E is set of edges.

Graph Representations

Graph data structure is represented using following representation

1 Adjacency Matrix

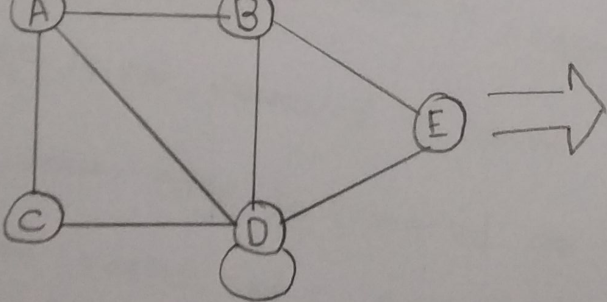
2. Incidence Matrix

3. Adjacency List

Adjacency Matrix

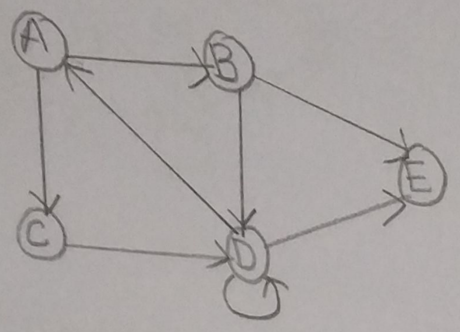
In this representation, the graph is represented using a matrix of size total number of vertices by a total number of vertices. That mean a graph with 4 vertices is represented using a matrix of size 4x4 In this matrix, both rows and coloms represent vertices. This matrix is filled with either 1 or 0. Here 1 represents that there is an edge from row vertex to colomm vertex and 0 represents that there is no edge from row vertex to colomn vertex.

for example, consider the following undirected graph representation



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 1 & 0 \\ B & 1 & 0 & 0 & 1 & 1 \\ C & 1 & 0 & 0 & 1 & 0 \\ D & 1 & 1 & 1 & 1 & 1 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$

Directed graph representation



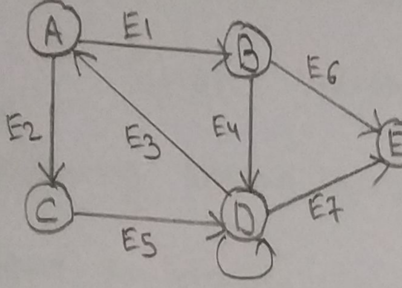
$\Rightarrow$

$$\begin{array}{c|ccccc}
 & A & B & C & D & E \\
\hline
A & 0 & 1 & 1 & 0 & 0 \\
B & 0 & 0 & 0 & 1 & 1 \\
C & 0 & 0 & 0 & 1 & 0 \\
D & 1 & 0 & 0 & 1 & 1 \\
E & 0 & 0 & 0 & 0 & 0
\end{array}$$

Incidence Matrix

In this representation, the graph is represented using matrix of size total number of vertices by total number of edges. That means graph with 4 vertices and 6 edges is represented using a matrix of size 4X6. In this matrix, rows represent vertices and columns represents edges. This matrix is filled with 0 or 1 or -1 Here, 0 represents that the row edge is not connected to column vertex. 1 represents that the row edge is connected as the outgoing edge to column vertex and -1 represents that the row edge is connected as the incoming edge to column vertex

For example, consider the following directed graph

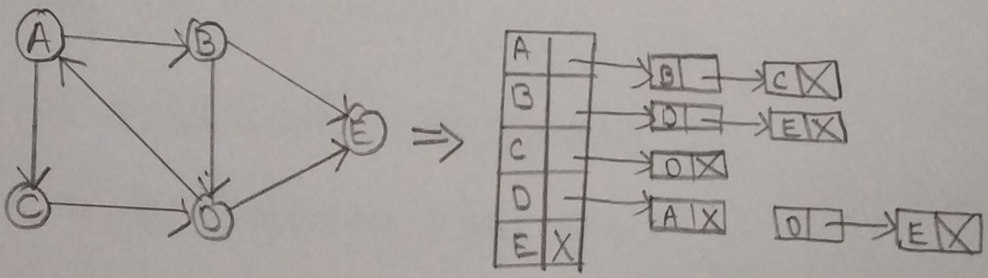


$$\Rightarrow \begin{array}{c|cccccccc} & E1 & E2 & E3 & E4 & E5 & E6 & E7 & E8 \\ \hline A & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ B & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ C & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ D & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 1 \\ E & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \end{array}$$

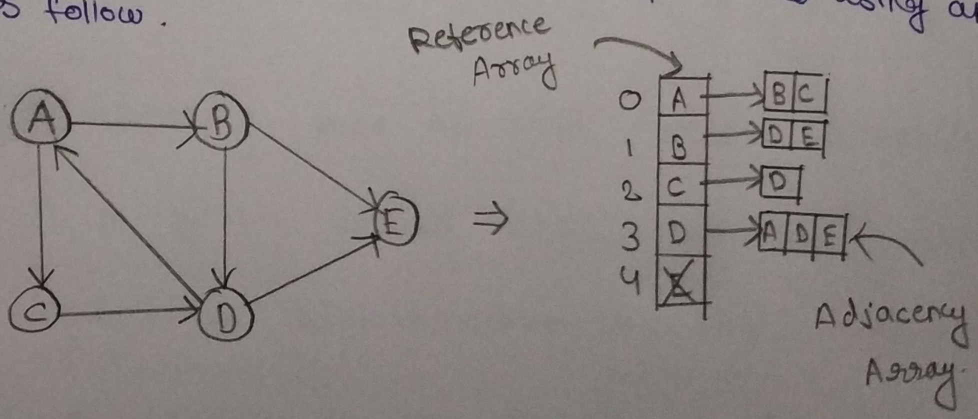
Adjacency List

In this repreentation, every vertex of a graph contains List of its adjacent vertices.

For example. Consider the following directed graph representation implemented using linked list.



This repreentation can also be implemented using an array as follow.



Graph Traversal Techniques

Graph traversal is a technique used for a searching vertex in a graph. The graph traversal is also used to decide the order of vertices is visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

There are two graph traversal technique and they are as follows

1) DFS (Depth First Search)
2) BFS (Breadth First Search)

DFS (Depth First Search)

DFS traversal of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops we use stack data structure with maximum size of total number of vertices in the graph to implement DFS

We use the following Steps to implement DFS traversal

Step 1- Define a stack of size total number of vertices in the graph.

Step 2- Select any vertex as starting point for traversal. visit that vertex and push it on the stack

Step 3. visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on the stack.

Step 4. Repeat Step 3 untill there is no new vertex to be visited from the vertex which is at the top of the stack.

Step 5. When there is no new vertex to visit then use back tracking and pop one vertex from the Stack.

Step 6. Repeat Steps, 3, 4 and 5 until stack becomes Empty

Step 7: When stack becomes Empty then produce final spanning tree by removing unused edges from the graph.

BFS (Breadth First Search)

BFS traversal of a graph produces a spanning tree as final result Spanning Tree is a graph without loops.. We use Queue data structure with maximum size of total number of vertices in the graph to implement BFS traversal

we use the following Steps to implement BFS traversal

Step 1:. Define a Queue of size total number of vertices in the graph.

Step 2. Select any vertex as starting point for traversal visit that vertex and insert ~~it~~ into the Queue.

Step 3. Visit all the non-visited adjacent vertices of the vertex. Which is at front of the Queue and insert them into the Queue.

Step 4. When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.

Step 5. Repeat step 3 and 4 until queue becomes empty

Step 6. When queue becomes empty, then produce final spanning tree by removing unused edge from the graph.

DATA STRUCTURES USING C

Assignment Set - II PAPER - II

1. Write notes on various String Manipulation Function, Pointers and structures of C with suitable examples.

In C programming a string is a seallence of characters terminated will a null character \o.

for example

```
Char c[]= "cString";
```

When the compiler encounters a sequence of characters enclosed in the double quation marks it appends a null character \o at the end by default.

c		S	t	r	i	n	g	\0

memory diagram

Declaring a String

```
Char S[5];
```

S[0]	S[1]	S[2]	S[3]	S[4]

String Declaration in C, Here we have declared a String of 5 characters.

Initializing a String

```
Char c[] = "abcd";
Char c[50] = "abcd";
Char c[] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

c[0]	c[1]	c[2]	c[3]	c[4]
A	b	c	d	\0

String Initialization in C

Assigning value to strings

Arrays and strings are second-class citizens in c; they do not support the assignment operator once it is declared. for excample

```
char c[100];
c = "c programming";    // Error! array type is not
                            assignable
```

Note:- Use the strcpy() function to copy the string insted.

Read a string from the user

You can use the scanf() function to read a string.

The scanf() function reads the sequence of characters until it encounters whitespace (space newline, tab etc)

example

```
 # include < Stdio.h>
  int main()
 {
 Char ~~mame~~ name[20];
 Printf("Enter name:");
 Scanf("%S", name);
 Printf("your name is %S.", name);
 return 0;
}
```

outpot

```
Enter name: Faisal
your name is Faisal.
```

FUNCTIONS

Function are essentially just group of statements that are to be executed as a unit in a given order and that can be referenced by a unique name. The only way to execute these statements is by invoking them or calling them using the function's name.

Traditional program design methodology typically involves a top-down or structured approach to developing software solution. The main task is first divided into a number simpler sub-tasks if these sub-tasks are still too complex they are subdivided further into simpler sub-tasks, and so on until the sub-tasks become simple enough to be programmed easily.

Function are highest level of the building blocks given to us in C and correspond to the sub-tasks or logical units referred to above. The identification of fonction in program design is an important step and will in general be a continuous process subiect to modification as more becomes know about the programming

Problem in progress.

Syntax:
```
return_type function_name(parameter_list)
{
  body of function;
}
```

The above is termed the function definition

For example: An example Function program

```
#include <stdio.h>        /* Standard I/O function prototype*/
 void Fun1 (void);        /* Prototype*/
 Void main(void)
 {
    Fun1();             /* function call*/
 }
 Void Fun1()              /* function definition */
 {
  Printf("inside the Function\n");
 }
```

Pointers

pointer are one of the most important mechanisms in C. They are the means by which we implement call by reference function, they are closely related to arrays and strings in C, they are fundamental to utilize C's dynamic memory allocation features, and they can lead to faster and more efficient code when used correctly.

A pointer is a variable that is used to store a memory address. Most commonly the address is the location of another variable in memory.

If one variable holds the address of another then it is said to point to the second variable

Address	value	Variable
1000		
1004	1012	ivar_ptr
1008		
1012	23	ivar
1016		

In the above illustration ivar is a variable of type int with a value 23 and stored at memory location 1012. ivar_ptr is a variable of type pointer to int which has a value of 1012 and is stored at memory location 1004. Thus ivar_ptr is said to point to the variable ivar and allows us to refer indirectly to it in memory.

example

```
#include <stdio.h>
 int main() {
  int nom =10;
  int *ptr;
  ptr = &nom;
  printf("value of nom: %d\n", *ptr);
   *ptr =20;
  printf("updated value of num: %d\n", nom);
   return 0;
 }
```

2. Discuss in detail about Arrays with matrix Multiplication program.

An Array is a collection of data storage Location. each having the same data type and the same name.

An Array can be visualzed as a row in a table. whose each successive block can be through of as memory bytes containing one element

Each Storage location in an array is called an array element.

Arrays may be represented in Row-major form or Columm-major form. In Row-major form all the elements of the first row are printed, then the elements of second row and so on up to the last row. In Column-major form, all the elements of the first column are printed, then the elements of the second column and so on up to the last column.

P.T.O

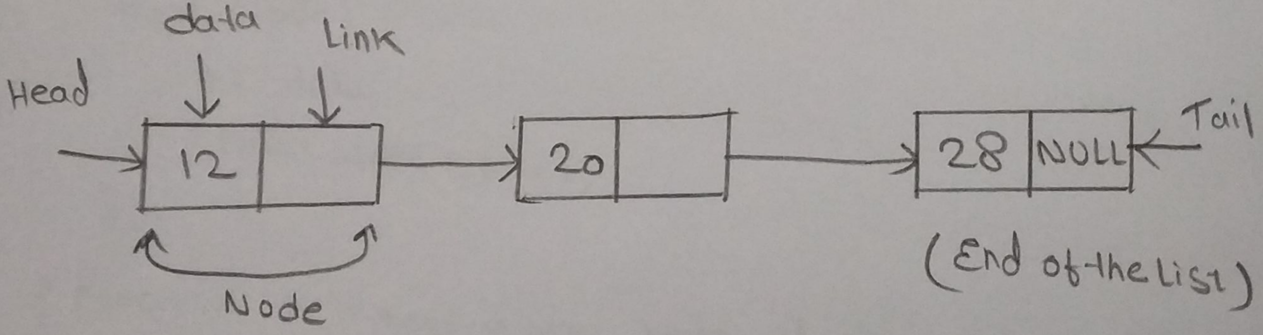
Matrix Multiplication program.

```
#include <stdio.h>
#define ROWS_A 3
#define ROWS_B 2
#define COLS_A 2
#define COLS_B 3

void matrix_multiply(int A[ROW_A][COLS_A],
      int B[ROWS_B][COLS_B],
      int c[ROWS_A][COLS_B]) {
      int i,j,k;
   for(i=0; i< ROWS_A; i++)
      {
      for(j=0; j<COLS_B; j++)
      {
       c[i][j]=0;
      for(k=0; k<COLS_A; k++) {
         c[i][j]+= A[i][k]* B[k][j];
      }
    }
  }
 }
void display_matrix(int rows, intcols, int matrix[rows][cols])
    {
     int i,j;
for (i= 0; i<rows; i++) {
    for (j=0; j<cols; j++) {
    printf ("%d\t", matrix[i][j]); }
    printf("\n");
  }
}
 int main() {
 int A[ROWS_A][COLS_A] = {{1,2},
                          {3,4},
                          {5, 6}};
int B[ROWS_B][COLS_B] = { {7,8,9},
                          {10, 11, 12}};
int c[ROWS_A][COLS_B];
matrix_multiply(A,B,c);
printf("matrix A:\n");
display_matrix(ROWS_A, COLS_A, A);
printf("\nmatrix B: \n");
display_matrix( ROWS_B, COLS_B, B);
printf("\n Resultant matrix c(A*B): \n");
display_matrix(ROWS_A, COLS_B, C);
return 0;
}
```

3. Write notes on Linked Lists, Insertion operation on Linked Lists and their Applications

Linked Lists can be defined as collection of objects called nodes that are randomly stored in the memory. Anode contains two field i-e data stored at that particular address and the pointer which contains the address of the next node in the memory.

The last node of the contains pointer to the null.



Uses of Linked List

The list is not required to be contiguously present in the memory. The node can reside anywhere in the memory and linked together to make a List. This achieves optimized utilization of space.

List size is limited to the memory size and doesn't need to be declared in advance.

Empty node cannot be present in the linked list.

we can store values of primitive types or objects in the singly linked List.

Insertion operation on Linked.

```
#include <Stdio.h>
# include <Std lib.h>

// Define the structure for a node

Struct Node {
        int data;
        Struct Node * next;
 };

// Function to insert a new node at the begingging of
                                              the List
void insertAtBeginning (struct Node** head_ref, int new_
                                                 data) {
 // Allocate memory for the new node
 Struct Node * new_node = (struct Node*) malloc (sizeof(
                                    Struct Node));
```

```
// Assign data to the new node
  new_node->data = new_data;
// Set the next of the new node to the current head
  new-node->next = *head_ref;
// Move the head to point to the new node
  *head_ref = new_node;
}
// Function to print the linked List
 void PointList(Struct Node* node) {
  While (node != NULL) {
   printf("%d", Node->data);
    node = node->next;
  }
  printf("\n");
}
// main function to test the insertion operation
  int main() {
  // Initialize an empty linked List
   Struct Node * head = NULL;
// Insert some elements at the beginning of the List
```

```
insertAtBeginning(&head,3);
insertAtBeginning(&head,5);
insertAtBeginning(&head,7);

// print the Linked list

 printf("Linked list after insertion:");
  printList(head);

  return 0;
}
```

Applications of Linked Lists

(i) Implementation of stacks and queues

(ii) Implementation of graph: Adjacency list representation of graph is most popular which is uses linked list to store adjacent vertices.

(iii) Dynamic memory allocation: we use linked list of free blocks.

(iv) maintaining directory of names

(v) performing arithmetic operation on long integers

(vi) manipulation of polynomials by storing constants in the node of linked list.

4. Discuss about Binary Trees, Various Binary Tree Representation and Binary Tree Traversals With examples.

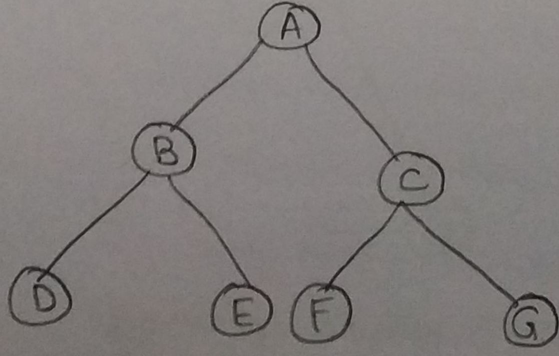
Binary Trees

In a normal tree, every node can have any number of children. A binary tree is a special type of tree data structure in which every node can have a maximum of 2 childred. one is known as as a left child and the other is know as right child

A tree in which every node can have maximum of two children is called Binary tree.

In a binary tree, every node can have either 0 or 1 child or 2 children but not more than 2 children.

example

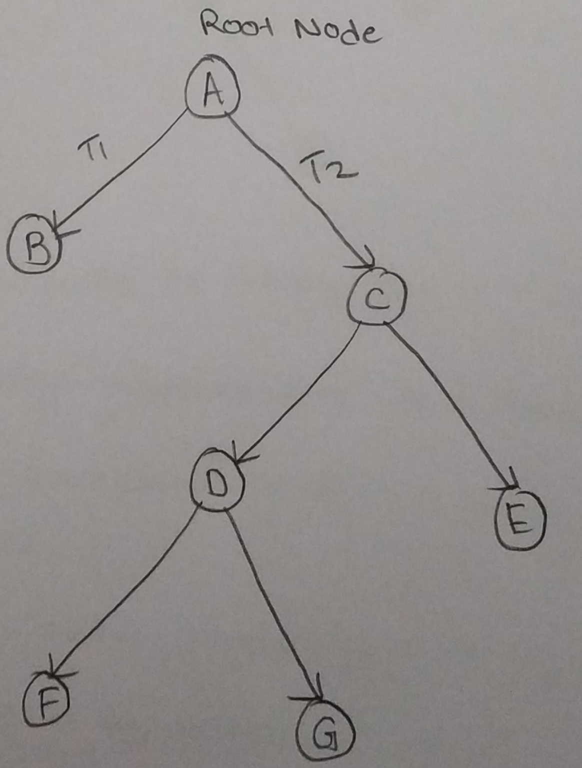


There are different types of binary tree and they are

Strictly Binary Tree

In a binary tree, every node can have a ~~xx~~ maximum of two children. But in strictly binary tree, every node should have exactly two children or none. That means every internal node must have exactly two children. A strictly Binary Tree can be defined as follow.

A binary tree in which every node has either two or zero number of children is called strictly Binary Tree



Strictly Binary Tree

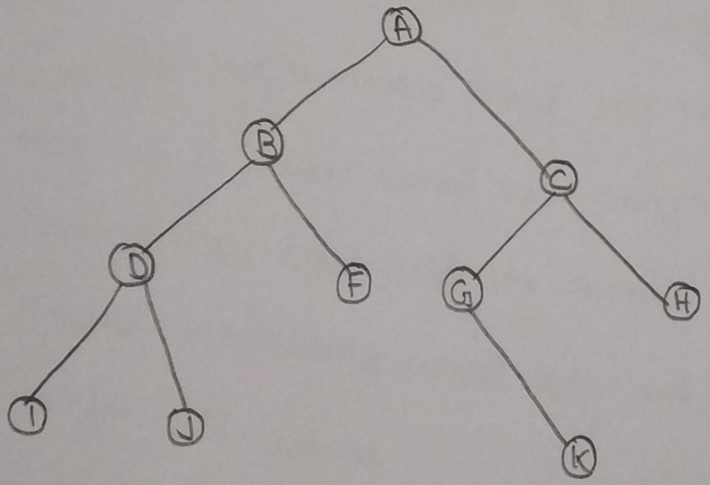
Binary Tree Representations

A Binary tree data structures is represented using two methods. Those methods are as follow

Array Representation

Linked List Representation

Consider the following binary tree



Array Representation of Binary Tree

In ~~the~~ array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.

Consider the above example of a binary tree and it is represented as follows

A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-

To represent a binary tree of depth 'n' using Array representation, we need one dimensional array with a maximum size of 2n+1.

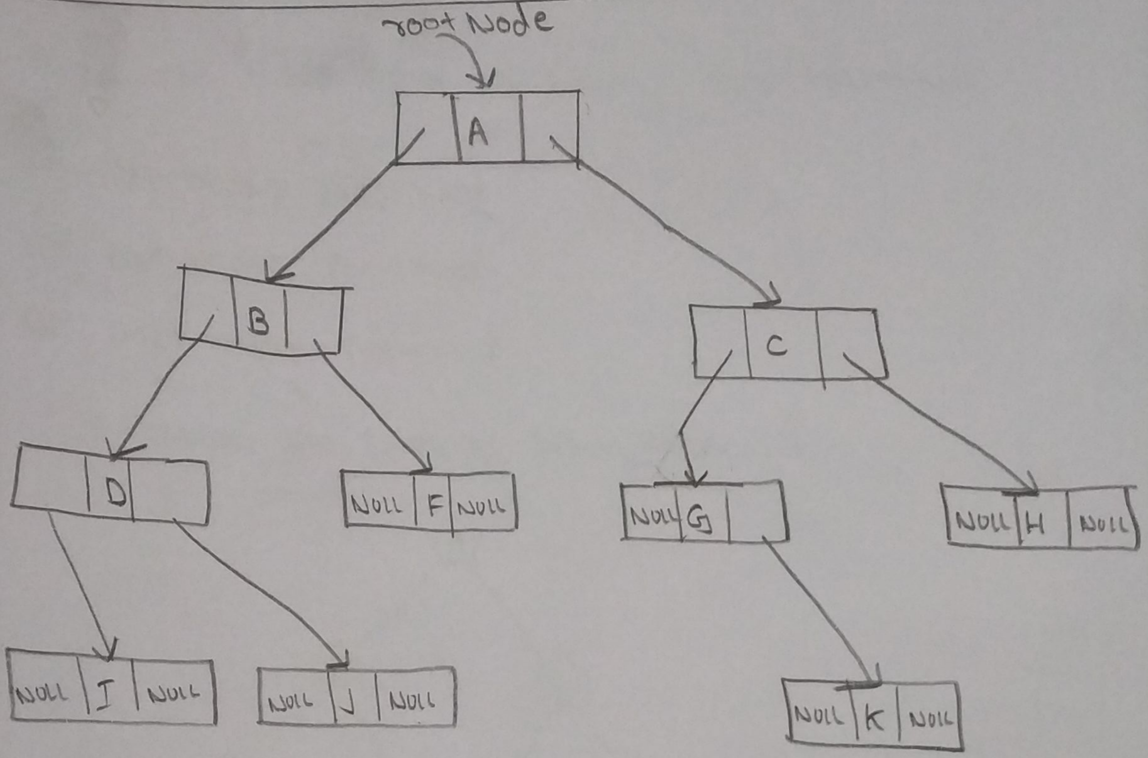
Linked List Representation of Binary Tree

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address

In this linked list representation, a node has the following structure

Left child Address	Data	Right child Address

The above example of binary tree represented using Linked list representation is show as follows



Binary Tree Traversals

When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed. In any binary tree, displaying order of nodes depends on the traversal method

Diplaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.

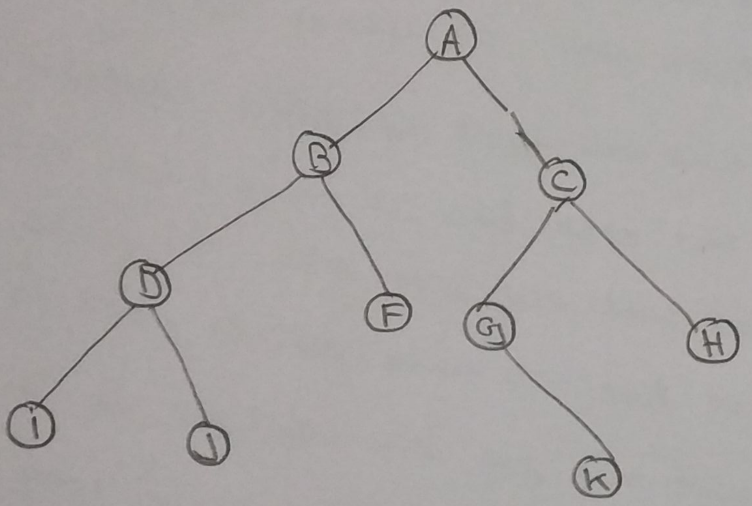
There are three types of binary tree traversals.

(1) In-order Traversal

(2) pre-order Traversal

(3) Post-order Traversal

Consider the following binary tree..



In-order Traversal (leftchild - root - right child)

In In-order traversal, the root node is visited between the left child and right child. In this traversal, the left child node is visited first, then the root node is visited and later we go for visiting the right child node. Thus in-order traversal is applicable for every root node of all subtree in the tree.

In the above example of a binary tree, first we try to visit left chid of root node 'A', but A's left child 'B' is a root node for left subtree. So we try to visit its (B's) left child 'D' and again D is a root for subtree with nodes. D, I and J. So we try to visit its left child 'I' and it is leftmost child. So first we visit 'I' then go for its root node 'D' and later we visit D's right child 'J' with this we have combleted the left part of node B. Then visit 'B' and next B's right child 'F' is visited. with this we have completed left part of node A. Then visit root node 'A' with this we have completed left and root parts of node A. Then we go for the right part of the node A.

In right of A again there is a subtree with root C. So go for left child of C and again it is a subtree with root G. But G does not have left part so we visit 'G' and then visit G's right child K. with this we have completed the left part of node C. Then visit root node 'C' and next visit C's right 'H' hich is rightmost child in the tree. So we That means here we have visited in the order of I-D-J-B-F-A-G-K-C-H using in order Traversal

In-order Traversal for above xample binary tree is

I-D-J-B-F-A-G-K-C-H

Pre-order Traversal (root - leftChild - right child)

In pre order traversal, the root node is visited before the left Child and right child nodes. This Pre-order traversal is applicable for every root node of all sub tree in the tree. In the above excample of binary tree, first we visit root node 'A' then visit its left child 'B' which is a root for D and F. So we visit B's left child 'D' and again D is a root for I and J. So we visit D's left child 'I' Which is the leftmost child. So next we go for visiting D's right child 'J' With this we have completed root, left and right parts of node 'D' and root, left parts of node B. Next Visit B's right child 'F' With this we have completed root and left parts of node A So we go for A's right child 'C' which is a root node for G and H. After visiting C. we go for its left child 'G' which is a root for node K so next we visit left of G. but it doesn't have left child so we go for G's right child 'K' With this, we have completed node C's root and left parts. Next visit C's right child 'H' which is the rightmost child in the tree. So we stop the process.

That means here we visited in the order of A-B-D-I-J-F-C-G-K-H using pre order Traversal

Pre-order Traversal for above example binary tree is

A-B-D-I-J-F-C-G-K-H

Post-order Traversal (left child - right child - root)

In post-order traversal, the root node is visited after left child and right child. In this traversal, left child node is visited first, then its right child and then it's root node. This is recursively performed until the right

right most node visited

Here we have visted in the order of I-J-D-F-B-K-G-H-C-A using Post-order Traversal.

Post-order Traversal for above example binary tree is

I-J-D-F-B-K-G-H-C-A.

5. Write notes on searaching, Sorting and complete C program to implement Quick Sort.

Searching

Searching is a process of finding a specific value in a list of existing value. In other words, searching is the process of locating given value position in a list of value.

Sorting

Arranging a list of given elements in an order (ascending or descending) is called as Sorting. There are various sorting technique available, namely,

1. Bubble Sort
2. Selection Sort
3 Insertion Sort
4 Merge sort
5 Quick sort. e.t.c.

Program to implement Quick Sort.

```
#include <stdio.h>
void quickSort (int nomber[25], int first, int last) {
 int i, j, pivot, temp;
```

```
if(first<last){
pivot=first;
i=first;
j=last;
while(i<j){
while(number[i]<=number[pivot]&&i<last)
 i++;
while(number[j]>number[pivot]
 j--;
if(i<j){
temp=number[i];
 number[i]=number[j];
 number[j]=temp;
}
}
temp=number[pivot];
number[pivot]=number[j];
 number[j]=temp;
quickSort(number,first,j-1);
quicksort(number,j+1,last);
}
}
 int main(){
```

```
int i, count, number[25];
printf("Enter some elements (max. -25): ");
Scanf("%d", & count);
printf ("Enter %d elements: ", count);
 for (i=0; i<count; i++)
 Scanf ("%d", &number [i]);
 quicksort (number, 0, count-1);
 printf ("The sorted order is:");
 for (i=0; i<count; i++)
  printf (" %d, number [i]);
  return 0;
}
```