VIDYAVARDHAKA COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

DATA STRUCTURES LABORATORY

20CS37

List of Experiments

SL No.	Experiments/Programs
1	Pattern Matching
2	Infix to postfix conversion and evaluation of the same
3	Priority Queue
4	Linked list for union and intersection of two lists
5	Operation on Binary Search Tree
6	Operations on Doubly Linked List.
7	Implementation of AVL Tree
8	Implementation of Red Black Tree
9	Operations on Graphs (G) of cities
10	Implementation of Hashing technique & resolving collision using linear probing.

1. Design, Develop and Implement a Program in C for the following operations on Strings
   1. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)
   2. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in
      STR.
   3. Pattern Matching Algorithm: Brute Force / KMP
   4. Support the program with functions for each of the above operations. Don't use Built-in functions
   5. Check the following test cases.

Test Case 1: STR = "VVCE MYSURU", PAT=" MYSURU", REP=" KARNATAKA", OUTPUT=" VVCE KARNATAKA"

Test Case 2: STR = "COMPUTER SCIENCE", PAT=" COMPUTER", REP=" BASIC", OUTPUT=" BASIC SCIENCE"

```
#include<stdio.h>
#include<stdlib.h>
char str[100], pat[50], rep[50], ans[100];
int i, j, c, m, k, flag=0;
void stringmatch()
{
i = m = c = j = 0;
while(str[c] ! = '\0')
{
        if(str[m] = = pat[i]) // ..... matching
        {
                i++; m++;
                if(pat[i] = = '\0') //.....found occurrences.
                {
                        flag = 1;
                        //.... copy replace string in ans string.
                        for(k = 0; rep[k] != '\0'; k++, j++)
                                ans[j] = rep[k];
                        i = 0;
                        c = m;
                }
        }
        else //... mismatch
        {
                ans[j] = str[c];
                j++;
                c++;
                m = c;
                i = 0;
        }
}
```

```
        ans[j] = '\0';
}
        void main()
        {
                printf("\nEnter a main string \n");
                gets(str);
                printf("\nEnter a pattern string \n");
                gets(pat);
                printf("\nEnter a replace string \n");
                gets(rep);
                stringmatch();
                if(flag = = 1)
                        printf("\nThe resultant string is\n %s" , ans);
                else
                        printf("\nPattern string NOT found\n");
        }
```

OUTPUT:

```
Enter a main string
Test
Enter a pattern string
Te
Enter a replace string
Re
The resultant string is
Rest

Enter a main string
This is Data Structure lab
Enter a pattern string
Data Structure
Enter a replace string
Data structure with C
The resultant string is
This is Data structure with C lab

Enter a main string
This is Data Structure lab
Enter a pattern string
Date
Enter a replace string
DATA
Patter\n string NOT found
```

2. Design, Develop and Implement a Program in C for the following operations on expression.

a. Read infix expression String (INFIX)
b. Convert the infix expression (INFIX) to a postfix expression using stacks.
c. Evaluate the postfix expression using stacks.
d. Check the following test cases.

Test Case 1: Infix = "(1+ (2-3) *4)", Postfix="123-4*+", Result = -3

Test Case 2: Infix = "4/2-2+3*3-4*2", Postfix="42/233*42*-+-", Result = -1

Note: Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

```
#include<stdio.h>
#include<string.h>
int F(char symbol)
{
        switch(symbol)
        {
                case '+' :
                case '-': return 2;
                case '*':
                case '/': return 4;
                case '^':
                case '$': return 5;
                case '(': return 0;
                case '#': return -1;
                default: return 8;
        }
}

int G(char symbol)
{
        switch(symbol)
        {
                case '+':
                case '-': return 1;
                case '*':
                case '/': return 3;
                case '^':
                case '$': return 6;
                case '(': return 9;
                case ')': return 0;
                default: return 7;
        }
}
```

```
void infix_postfix(char infix[], char postfix[])
{
        int top, j, i;
        char s[30], symbol;
        top = -1;
        s[++top] = '#';
        j = 0;
        for(i=0; i < strlen(infix); i++)
        {
                symbol = infix[i];
                while(F(s[top]) > G(symbol))
                {
                        postfix[j] = s[top--];
                        j++;
                }
                if(F(s[top]) != G(symbol))
                        s[++top] = symbol;
                else
                        top--;
        }
        while(s[top] != '#')
        {
                postfix[j++] = s[top--];
        }
        postfix[j] = '\0';
}

void main()
{
        char infix[20], postfix[20];
        printf("\nEnter a valid infix expression\n");
         gets(infix);
        infix_postfix(infix,postfix);
        printf("\nThe infix expression is:\n");
        printf ("%s",infix);
        printf("\nThe postfix expression is:\n");
        printf ("%s",postfix);

}
```

OUTPUT:

```
Enter a valid infix expression
(a+(b-c)*d)
The infix expression is:
(a+(b-c)*d)
The postfix expression is: abc-d*+
```

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
float s[25];
int top=-1;
float operation(char op, float op1,float op2)
{
	switch(op)
	{
		case '+': return(op1+op2);
		case '-': return(op1-op2); break;
		case '*': return(op1*op2);
		case '/': return(op1/op2);
		case '^': return(pow(op1,op2));
		case '%': return((int)op1%(int)op2);
		default: return(0);
	}
}
void push(float symbol)
{
	s[++top] = symbol;
}
float pop()
{
	return(s[top--]);
}
void main()
{
	char postfix[25],symbol;
	float op1,op2,res;
	int i;
	printf("Enter the Postfix Expression\n");
	scanf("%s",postfix);
	for( i=0; postfix[i]!='\0'; i++ )
	{
		symbol = postfix[i];
		if(isdigit(symbol))
			push(symbol-'0');
		else
		{
			op2 = pop();
			op1 = pop();
			res = operation(symbol,op1,op2);
			push(res);
		}
	}
	res = pop();
```

```
    printf("Result=%.2f ",res);
}
```

3. Design, Develop and implement menu driven program to simulate processing of batch jobs by a computer system. The scheduling of these jobs should be handled using a priority queue.

   Note:

   Program should allow users to add or remove items from the queue.

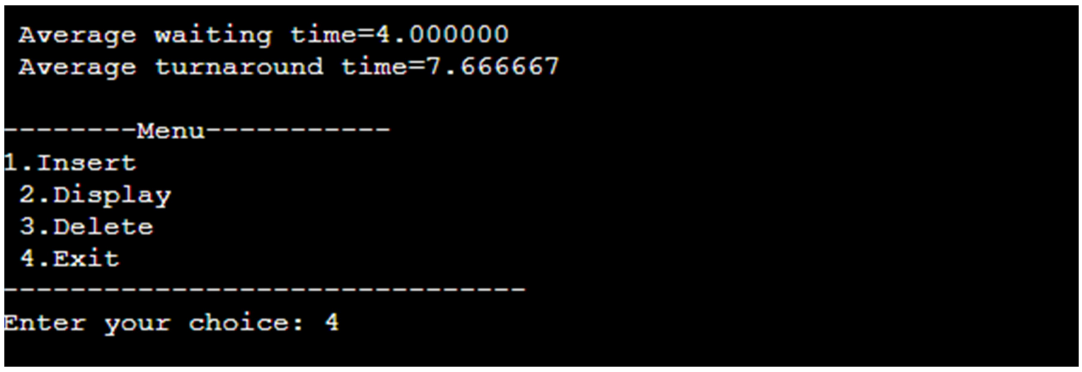
   It should also display current status i.e. the total number of items in the queue The.

```
#include<stdio.h>
#include<stdlib.h>
struct Process
{
   int pid;  // Process ID
   int bt;   // CPU Burst time required
   int priority; // Priority of this process
};
 typedef struct  Process proc;
proc pro[30],temp;
int n=0;
  void insert()
 {
    int i=0,j=0;
      printf("\nEnter the process id, burst time and priority of the queue\n ");
      scanf("%d %d %d", &temp.pid,&temp.bt,&temp.priority);
  while(temp.priority<pro[i].priority)  //while loop to locate the position to insert
      i++;
 for(j=n-1; j>=i; j--) // to shift the items towards right to insert the element
      {
       pro[j+1] = pro[j];
      }
         pro[i] = temp;  // inserting
         n = n+1;        // increment number of items
      }
     void del() //deleting an array element
    {
      int i,elem,pid;
      for(i=0; i<=n+1; i++)
      {
      printf("Deleted element is pid =%d\t\tburst
time=%d\t\tpriority=%d\t\t\n",  pro[i].pid,pro[i].bt,pro[i].priority);
      pro[i] =\ pro[i+1];
      n = n-1;
      }
```

```
   }

          void findWaitingTime(int wt[])
     {
   // waiting time for first process is 0
      wt[0] = 0;

   // calculating waiting time
     for (int  i = 1; i < n ; i++ )
      wt[i] =  pro[i-1].bt + wt[i-1] ;
     }

// Function to calculate turn around time
void findTurnAroundTime(int wt[], int tat[])
 {
   // calculating turnaround time by adding
   // bt[i] + wt[i]
   for (int  i = 0; i < n ; i++)
      tat[i] = pro[i].bt + wt[i];
  }

//Function to calculate average time
void display()
 {
    if(n==0)
    {
    printf("Priority Queue is empty\n");
    return;
    }
   int wt[n], tat[n], total_wt = 0, total_tat = 0;

   //Function to find waiting time of all processes
   findWaitingTime( wt);

   //Function to find turn around time for all processes
   findTurnAroundTime(wt, tat);

   //Display processes along with all details
   printf("\nProcesses     Burst time      Waiting time   Turn around time\n");

   // Calculate total waiting time and total turn
   // around time
   for (int  i=0; i<n; i++)
   {
      total_wt = total_wt + wt[i];
      total_tat = total_tat + tat[i];
      printf("   %d\t\t%d\t\t%d\t\t%d\n",  pro[i].pid,pro[i].bt,wt[i],tat[i]);
```

```
    }

    printf("\n Average waiting time=%f\n Average turnaround time=%f",
(float)total_wt/(float)n,(float)total_tat / (float)n);
}

    void main()
    {
       int ch;
          do{
          printf("\n\n--------Menu----------\n");
          printf("1Insert\n 2.Display 3.Delete\n 5.Exit\n");
          printf("------------------------------");
          printf("\nEnter your choice: ");
          scanf("%d", &ch);
          switch(ch)
          {
         case 1: insert();
          break;
          case 2: display();
          break;
          case 3: del();
          break;
          case 4: exit(0);
          break;
          default: printf("\nInvalid choice:\n");
          break;
       }
    }while(ch!=5);
}
```

```
--------Menu-----------
1.Insert
 2.Display
 3.Delete
 4.Exit
------------------------------
Enter your choice: 1

Enter the process id, burst time and priority of the queue
 3 3 1


--------Menu-----------
1.Insert
 2.Display
 3.Delete
 4.Exit
------------------------------
Enter your choice: 1

Enter the process id, burst time and priority of the queue
 2 4 2
--------Menu-----------
1.Insert
 2.Display
 3.Delete
 4.Exit
------------------------------
Enter your choice: 1

Enter the process id, burst time and priority of the queue
 4
4 5


--------Menu-----------
1.Insert
 2.Display
 3.Delete
 4.Exit
------------------------------
Enter your choice: 2

Processes     Burst time      Waiting time    Turn around time
   4             4               0               4
   2             4               4               8
   3             3               8               11
```



4. Design, Develop and implement c program for the following scenario

a.There are two linked list A and B containing the following data:

A: 3,7,10,15,16,09,22,17,32 and B: 16,02,09,13,37,08,10,01,28

b.Create a linked list C that contains only those elements that are common in linked list A and B

c.Create a linked list D which contains all elements of A as well as B ensures that there is no repetition of elements.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;

int search(int key, NODE first)
{
   NODE cur;
   if(first==NULL)
   return 0;

   cur=first;
   while(cur!=NULL)
   {
      if(key==cur->info)
      return 1;
      cur=cur->link;
   }
   return 0;
}
```

```
void display(NODE first)
{
   NODE cur;
   if(first==NULL)
  {
      printf("List is empty\n");
      return;
    }
  printf("The contents of singly linked list is\n");
  cur=first;
  while(cur!=NULL)
  {
    printf("%d\n",cur->info);
    cur=cur->link;
     }
 printf("\n");
 }

NODE getnode()
{
      NODE x;
      x=(NODE) malloc(sizeof(struct node));
      if(x==NULL)
      {
        printf("Out of memory\n");
        exit(0);
        }
    return x;
}

NODE insert_rear(int item,NODE first)
{
  NODE temp;
  NODE cur;
  temp=getnode();
  temp->info=item;
  temp->link=NULL;

 if(first==NULL)
 return temp;

 cur=first;
 while(cur->link!=NULL)
{
  cur=cur->link;
 }
 cur->link=temp;
```

```
 return first;
 }

 NODE remove_duplicate(NODE first)
{
   NODE a, b;
   int flag;
   if(first==NULL)
    return NULL;
    b=NULL;
        a=first;
    while(a!=NULL)
    {
       flag=search(a->info,b);
       if(flag==0)
       b=insert_rear(a->info,b);
       a=a->link;
     }
    return b;
}
NODE union_of_list(NODE first, NODE second)
{
   NODE a, third;
   int flag;
   a=first;
   third=NULL;
   while(a!=NULL)
   {
      third=insert_rear(a->info,third);
      a=a->link;
   }
      a=second;
   while(a!=NULL)
   {
      flag=search(a->info,third);
      if(flag==0)
      third=insert_rear(a->info,third);
      a=a->link;
   }
   return third;
}

NODE intersection_of_list(NODE first, NODE second)
{
    NODE a,b,third;
    int flag;
    a=first;
```

```
    b=second;
    third=NULL;
        while(a!=NULL)
    {
       flag=search(a->info,b);
       if(flag==1)
       third=insert_rear(a->info,third);
       a=a->link;
    }
    return third;
    }

void main()
{
  NODE first,second,third;
int choice,item,i,n;
for(;;)
{
   printf("1:Create first list\n");
    printf("2:Create second list\n");
    printf("3:Remove duplicates of list 1\n");
     printf("4:Remove duplicates of list 2\n");
    printf("5:Union of two lists\n");
    printf("6: Intersection of two lists\n");
    printf("7:Exit\n");
   printf("Enter the choice\n");
   scanf("%d",& choice);
   switch(choice)
   {
    case 1:
     printf("enter the number of nodes in the LIST 1\n");
      scanf("%d",&n);
    first=NULL;
   for(i=1;i<=n;i++)
    {
     printf("enter the item\n");
     scanf("%d",&item);
     first=insert_rear(item,first);
     }
    break;
case 2:
printf("Enter the number of nodes in second list\n");
    scanf("%d",&n);
    second=NULL;
   for(i=1;i<=n;i++)
  {
     printf("enter the item\n");
```

```
    scanf("%d",&item);
    second=insert_rear(item,second);
    }
    break;
  case 3:
  printf("The first list before removing duplicate  is:\n");
  display(first);
  first=remove_duplicate(first);
  printf("The first list after removing duplicates\n");
  display(first);
  break;
  case 4:
  printf("The second list before removing duplicate  is:\n");
  display(second);
  second=remove_duplicate(second);
  printf("The first list after removing duplicates\n");
  display(second);
  break;
  case 5:
  printf("The first list is \n");
  display(first);
  printf("The second list is \n");
  display(second);
  third=union_of_list(first,second);
  printf("The union of two lists\n");
  display(third);
  break;
    case 6:
  printf("The first list \n");
  display(first);
  printf("The second list\n");
  display(second);
    third=intersection_of_list(first,second);
  printf("The intersection of two lists \n");
  display(third);
  break;
  default:
  exit(0);
}
}
}
```

```
Input:
   List1: 10->15->4->20
   List2:  8->4->2->10
Output:
   Intersection List: 4->10
   Union List: 2->8->20->4->15->10
```

5. Design, Develop and Implement a menu driven Program in C for the following operations on **Binary Search Tree (BST)** of Integers

a. Create a BST of **N** Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (**KEY**) and report the appropriate message
d. Exit.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};
typedef struct node *NODE;
NODE getnode()
{
    NODE x;
    x=(NODE)malloc(sizeof(struct node));
    if(x==NULL)
    {
            printf("Insufficient memory");
            exit(0);
    }
    return x;
}

void preorder(NODE root)
{
    if(root==NULL) return;
    printf("%d\t",root->info);
    preorder(root->llink);
    preorder(root->rlink);
```

```
}
void postorder(NODE root)
{
    if(root==NULL) return;
    postorder(root->llink);
    postorder(root->rlink);
    printf("%d\t",root->info);
}
void inorder(NODE root)
{
    if(root==NULL) return;
    inorder(root->llink);
    printf("%d\t",root->info);
    inorder(root->rlink);
}
NODE insert(int item,NODE root)
{
    NODE temp,cur,prev;
    temp=getnode();
    temp->info=item;
    temp->llink=NULL;
    temp->rlink=NULL;
    if(root==NULL)
            return temp;
    prev=NULL;
    cur=root;
    while(cur!=NULL)
    {
            prev=cur;
       if(item==cur->info)
            {
                    printf("Duplicate item are not allowed\n");
                    free(temp);
                    return root;
            }
            if(item<cur->info)      cur=cur->llink;
            else                    cur=cur->rlink;
    }
    if(item<prev->info)
            prev->llink=temp;
    else
            prev->rlink=temp;
    return root;
}

NODE search(int item, NODE root)
{
```

```
    NODE cur;
    if(root==NULL) return NULL;
    cur=root;
    while(cur!=NULL)
    {
            if(item==cur->info) return cur;
            if(item<cur->info)
                    cur=cur->llink;
            else
                    cur=cur->rlink;
    }
    return NULL;
}
void main()
{
    int item,ch,n,i;
    NODE root,cur;
    root=NULL;
    while (1)
    {
            printf("\nEnter
            thechoice\n1.Insert\n2.Preorder\n3.Inorder\n4.Postorder\n5.Sea
            rch an Element\n6:Exit\n");
            scanf("%d",&ch);
            switch(ch)
            {
                    case 1:
                            printf("Enter the number of items\n");
                            scanf("%d",&n);
                            printf("Enter the item to be inserted\n");
                            for(i=0;i<n; i++)
                            {
                            scanf("%d",&item);
                            root=insert(item,root);
                            }
                            break;
                    case 2: if(root==NULL)
                            {
                                    printf("Tree is empty\n");
                                    break;
                            }
                            printf("The given tree in tree form is\n");
                            printf("Preorder traversal is \n");
                            preorder(root);
                            printf("\n");
                            break;
                    case 3: if(root==NULL)
```

```
                {
                        printf("Tree is empty\n");
                        break;
                }
                printf("Inorder traversal is \n");
                inorder(root);
                printf("\n");
                break;
        case 4: if(root==NULL)
                {
                        printf("Tree is empty\n");
                        break;
                }
                printf("Postorder traversal is \n");
                postorder(root);
                printf("\n");
                break;
        case 5: printf("Enter the item to be searched\n");
                        scanf("%d",&item);
                        cur=search(item,root);
                        if(cur==NULL)
                                printf("Item not found\n");
                        else
                                printf("Item found\n");
                break;
        case 6: exit(0);
        default: printf("Enter valid choice\n");
    }
  }
}
```

Output:

```
Enter the choice
    1. Insert
    2. Preorder
    3. Inorder
    4. Postorder
    5. Search an element
    6. Exit
1
Enter the number of items
12
Enter the item to be inserted
6 9 5 2 8 15 24 14 7 8 5 2
Enter the choice
    1. Insert
```

2. Preorder
3. Inorder
4. Postorder
5. Search an element
6. Exit

2
Preorder traversal is
6 5 2 2 5 9 8 7 8 15 14 24
Enter the choice

1. Insert
2. Preorder
3. Inorder
4. Postorder
5. Search an element
6. Exit

3
Inorder traversal is
2 2 5 5 6 7 8 8 9 14 15 24
Enter the choice

1. Insert
2. Preorder
3. Inorder
4. Postorder
5. Search an element
6. Exit

4
Postorder traversal is
2 2 5 5 7 8 8 14 24 15 9 6
Enter the choice

1. Insert
2. Preorder
3. Inorder
4. Postorder
5. Search an element
6. Exit

5
Enter the item to be searched
12
Item not found
Enter the choice

1. Insert
2. Preorder
3. Inorder
4. Postorder
5. Search an element
6. Exit

Enter the item to be searched

6
Item found
Enter the choice
1. Insert
2. Preorder
3. Inorder
4. Postorder
5. Search an element
6. Exit
6

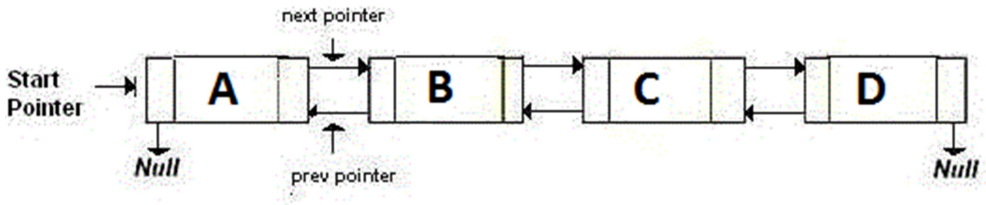
6. Design, Develop and implement C program for the following operations on doubly linked list.

a. Create doubly linked list of N nodes with integer data by adding each node at the front.

b. Delete the node of a given data if it is found, otherwise display appropriate message.

c. insert a node to the left of the node whose key value is read as input.

d. Display the contents of the list.




```
#include <stdio.h>
#include<stdlib.h>
struct node
{
   int info;
   struct node *llink;
   struct node *rlink;
};
typedef struct node * NODE;
NODE getnode()
{
      NODE x;
      x=(NODE) malloc(sizeof(struct node));
      if(x==NULL)
      {
        printf("Out of memory\n");
        exit(0);
        }
    return x;
```

```
}

NODE insert_left(int item,NODE head)
{
   NODE temp,prev,cur;
   if(head->rlink==head)
   {
      printf("list empty");
      return head;

   }
   cur=head->rlink;
   while(cur!=head)
   {
     if(item==cur->info)
     break;
     cur=cur->rlink;
   }
   if(cur==head)
   {
      printf("key not found");
      return head;
   }
   prev=cur->llink;
   printf("enter item");
   scanf("%d",&item);
   temp=getnode();
   temp->info=item;
   prev->rlink=temp;
   temp->llink=prev;
   cur->llink=temp;
   temp->rlink=cur;
   return head;
}
NODE insert_front(int item,NODE head)
{
   NODE temp,cur;
   temp=getnode();
   temp->info=item;
   cur=head->rlink;
   head->rlink=temp;
   temp->llink=head;
   temp->rlink=cur;
   cur->llink=temp;
   return head;
}
```

```
NODE del_info(int item,NODE head)
{
   NODE prev,cur,next;
   if(head->rlink==head)
   {
      printf("list is empty");
      return head;
   }
   cur=head->rlink;
   while(cur!=head)
   {
      if(item==cur->info)
      break;
      cur=cur->llink;
   }
   if(cur==head)
   {
      printf("item not found");
      return head;
   }
   prev=cur->llink;
   next=cur->rlink;
   prev->rlink=next;
   next->llink=prev;
   free(cur);
   return head;
}

void display(NODE head)
{
   NODE temp;
   if(head->rlink==head)
   {
      printf("List is empty");
      return;
   }
   printf("contents of list ");
   temp=head->rlink;
   while(temp!=head)
   {
      printf("%d\n",temp->info);
      temp=temp->rlink;
   }
}
void main()
{
   NODE head;
```

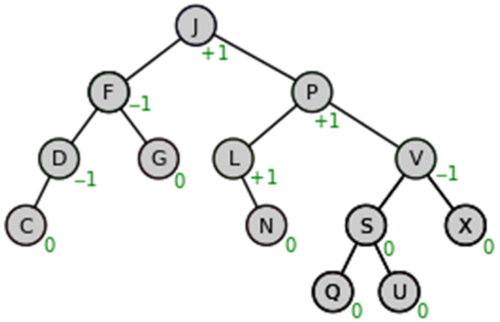
```
   int ch,key,item;
   head=getnode();
   head->rlink=head;
   head->llink=head;
   for(;;)
   {
      printf("1:insert_front 2.display 3.insert_before4.del_info 5.exit");
      printf("enter choice");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1:printf("enter item");
         scanf("%d",&item);
         insert_front(item,head);
         break;
         case 2:  display(head);
         break;
         case 3: printf("enter key");
                 scanf("%d",&key);
                head=insert_left(key,head);
         break;
         case 4: printf("enter item");
         scanf("%d",&item);
         head=del_info(item,head);
         break;
        default:exit(0);

      }
   }
}
```

Output:

```
Choices:
    1-Insert
    2-Insert left
    3-Delete node
    4-Display
    5-Exit
Enter your choice: 1

Enter the element to be inserted: 10
```

```
Choices:
    1-Insert
    2-Insert left
    3-Delete node
    4-Display
    5-Exit
Enter your choice: 4
20 10
```

7. Design, Develop a program in C to implement **AVL tree**.




```
#include<stdio.h>
typedef struct node
{
int data;
struct node *left,*right;
int ht;
}node;

node *insert(node *,int);
node *Delete(node *,int);
void preorder(node *);
void inorder(node *);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int BF(node *);
int main()
{
node *root=NULL;
```

```
int x,n,i,op;
do
{
printf("\n1)Create:");
printf("\n2)Insert:");
printf("\n3)Delete:");
printf("\n4)Print:");
printf("\n5)Quit:");
printf("\n\nEnter Your Choice:");
scanf("%d",&op);
switch(op)
{
case 1: printf("\nEnter no. of elements:");
scanf("%d",&n);
printf("\nEnter tree data:");
root=NULL;
for(i=0;i<n;i++)
{
scanf("%d",&x);
root=insert(root,x);
}
break;
case 2: printf("\nEnter a data:");
scanf("%d",&x);
root=insert(root,x);
break;
case 3: printf("\nEnter a data:");
scanf("%d",&x);
root=Delete(root,x);
break;
case 4: printf("\nPreorder sequence:\n");
preorder(root);
printf("\n\nInorder sequence:\n");
inorder(root);
printf("\n");
break;
}
}while(op!=5);
return 0;
}

node * insert(node *T,int x)
{
if(T==NULL)
{
T=(node*)malloc(sizeof(node));
T->data=x;
```

```
T->left=NULL;
T->right=NULL;
}
else
if(x > T->data) // insert in right subtree
{
T->right=insert(T->right,x);
if(BF(T)==-2)
if(x>T->right->data)
T=RR(T);
else
T=RL(T);
}
else
if(x<T->data)
{
T->left=insert(T->left,x);
if(BF(T)==2)
if(x < T->left->data)
T=LL(T);
else
T=LR(T);
}
T->ht=height(T);
return(T);
}

node * Delete(node *T,int x)
{
node *p;
if(T==NULL)
{
return NULL;
}
else
if(x > T->data) // insert in right subtree
{
T->right=Delete(T->right,x);
if(BF(T)==2)
if(BF(T->left)>=0)
T=LL(T);
else
T=LR(T);
}
else
if(x<T->data)
{
```

```
T->left=Delete(T->left,x);
if(BF(T)==-2) //Rebalance during windup
if(BF(T->right)<=0)
T=RR(T);
else
T=RL(T);
}
else
{
//data to be deleted is found
if(T->right!=NULL)
{ //delete its inorder succesor
p=T->right;
while(p->left!= NULL)
p=p->left;
T->data=p->data;
T->right=Delete(T->right,p->data);
if(BF(T)==2)//Rebalance during windup
if(BF(T->left)>=0)
T=LL(T);
else
T=LR(T);\
}
else
return(T->left);
}
T->ht=height(T);
return(T);
}
 int height(node *T)
{
int lh,rh;
if(T==NULL)
return(0);
if(T->left==NULL)
lh=0;
else
lh=1+T->left->ht;
if(T->right==NULL)
rh=0;
else
rh=1+T->right->ht;
if(lh>rh)
return(lh);
return(rh);
}
```

```
node * rotateright(node *x)
{
node *y;
y=x->left;
x->left=y->right;
y->right=x;
x->ht=height(x);
y->ht=height(y);
return(y);
}

node * rotateleft(node *x)
{
node *y;
y=x->right;
x->right=y->left;
y->left=x;
x->ht=height(x);
y->ht=height(y);
return(y);
}

node * RR(node *T)
{
T=rotateleft(T);
return(T);
}

node * LL(node *T)
{
T=rotateright(T);
return(T);
}

node * LR(node *T)
{
T->left=rotateleft(T->left);
T=rotateright(T);
return(T);
}

node * RL(node *T)
{
T->right=rotateright(T->right);
T=rotateleft(T);
return(T);
}
```

```
int BF(node *T)
{
int lh,rh;
if(T==NULL)
return(0);

if(T->left==NULL)
lh=0;
else
lh=1+T->left->ht;

if(T->right==NULL)
rh=0;
else
rh=1+T->right->ht;

return(lh-rh);
}

void preorder(node *T)
{
if(T!=NULL)
{
printf("%d(Bf=%d)",T->data,BF(T));
preorder(T->left);
preorder(T->right);
}
}

void inorder(node *T)
{
if(T!=NULL)
{
inorder(T->left);
printf("%d(Bf=%d)",T->data,BF(T));
inorder(T->right);
}
}
```

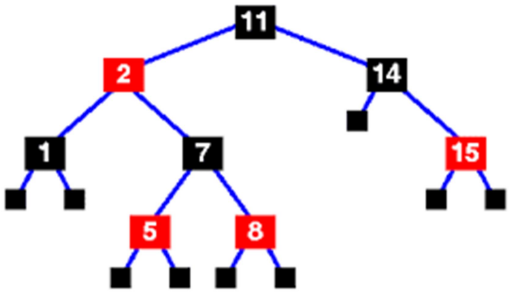
Output:

```
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:1

Enter no. of elements:5

Enter tree data:7 10 12 4 9
```

```
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:4

Preorder sequence:
10(Bf=1)7(Bf=0)4(Bf=0)9(Bf=0)12(Bf=0)

Inorder sequence:
4(Bf=0)7(Bf=0)9(Bf=0)10(Bf=1)12(Bf=0)
```

```
Enter Your Choice:3

Enter a data:10

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:4

Preorder sequence:
7(Bf=-1)4(Bf=0)12(Bf=1)9(Bf=0)

Inorder sequence:
4(Bf=0)7(Bf=-1)9(Bf=0)12(Bf=1)
```

8. Design, Develop a program in C to implement **Red-Black** Tree.



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

struct rbtNode{
   int key;
   char color;
   struct rbtNode *left, *right,*parent;
};

struct rbtNode* root = NULL;

void leftRotate(struct rbtNode *x){
   struct rbtNode *y;
   y = x->right; x->right = y->left;
   if( y->left != NULL)
   {
      y->left->parent = x;
   }
   y->parent = x->parent;
   if( x->parent == NULL){
      root = y;
   }
   else if((x->parent->left!=NULL) && (x->key == x->parent->left->key))
   {
      x->parent->left = y;
   }
   else x->parent->right = y;
   y->left = x; x->parent = y; return;
}

void rightRotate(struct rbtNode *y){
   struct rbtNode *x;
   x = y->left;
   y->left = x->right;
   if ( x->right != NULL)
   {
      x->right->parent = y;
   }
   x->parent = y->parent;
   if( y->parent == NULL)
   {
      root = x;
   }
   else if((y->parent->left!=NULL)&& (y->key == y->parent->left->key))
   {
```

```
      y->parent->left = x;
   }
   else
   y->parent->right = x;
   x->right = y; y->parent = x;
   return;
}
void colorinsert(struct rbtNode *z){
   struct rbtNode *y=NULL;
   while ((z->parent != NULL) && (z->parent->color == 'r'))
   {
      if ((z->parent->parent->left != NULL) && (z->parent->key == z->parent->parent-
>left->key))
      {
         if(z->parent->parent->right!=NULL)
            y = z->parent->parent->right;
         if ((y!=NULL) && (y->color == 'r'))
         {
            z->parent->color = 'b';
            y->color = 'b';
            z->parent->parent->color = 'r';
            if(z->parent->parent!=NULL)
               z = z->parent->parent;
         }
         else
         {
            if ((z->parent->right != NULL) && (z->key == z->parent->right->key))
            {
               z = z->parent;
               leftRotate(z);
            }
            z->parent->color = 'b';
            z->parent->parent->color = 'r';
            rightRotate(z->parent->parent);
         }
      }
      else
      {
         if(z->parent->parent->left!=NULL)
            y = z->parent->parent->left;
         if ((y!=NULL) && (y->color == 'r'))
         {
            z->parent->color = 'b';
            y->color = 'b';
            z->parent->parent->color = 'r';
            if(z->parent->parent!=NULL)
               z = z->parent->parent;
```

```
        }
        else
        {
           if ((z->parent->left != NULL) && (z->key == z->parent->left->key))
           {
              z = z->parent;
              rightRotate(z);
           }
           z->parent->color = 'b';
           z->parent->parent->color = 'r';
           leftRotate(z->parent->parent);
        }
     }
  }
  root->color = 'b';
}

void inorderTree(struct rbtNode* root){
  struct rbtNode* temp = root;
  if (temp != NULL)
  {
     inorderTree(temp->left);
     printf(" %d-%c ",temp->key,temp->color);
     inorderTree(temp->right);
  }
  return;
}



void traversal(struct rbtNode* root){
  if (root != NULL)
  {
     printf("root is %d- %c",root->key,root->color);
     printf("\nInorder tree traversal\n");
     inorderTree(root);
  }
  return;
}


void insert(int val){
  struct rbtNode *x, *y;
  struct rbtNode *z = (struct rbtNode*)malloc(sizeof(struct rbtNode));
  z->key = val;
  z->left = NULL;
  z->right = NULL;
```

```
    z->color = 'r';
    x=root;
    if ( root == NULL )
    {
       root = z;
       root->color = 'b';
       return;
    }
    while ( x != NULL)
    {
       y = x;
       if ( z->key < x->key)
       {
          x = x->left;
       }
       else
          x = x->right;
    }
    z->parent = y;
    if ( y == NULL)
    {
       root = z;
    }
    else if( z->key < y->key )
    {
       y->left = z;
    }
    else{
       y->right = z;
    }

    colorinsert(z);
    return;
}



void printTree(struct rbtNode *root, int space)
{
    if (root == NULL)
    {
       return;
    }

    else
    {
       space += 8;
```

```
   printTree(root->right, space);
   printf("\n");
   for (int i = 8; i < space; i++)
      printf(" ");
   printf("%d(%c)", root->key,root->color);
   printTree(root->left, space);
   }
}

void print(struct rbtNode *root)
{
if(root==NULL)
{
   printf("No Data");
}
else
{
   printTree(root, 0);
}
}
int main(){
   int choice,val,data,var,fl=0;
   while(1)
   {
     print(root);
printf("\nRed Black Tree Menu - \nEnter your choice :\n1:Insert\n 2:Traversal\n
3:Exit\n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
            printf("Enter the integer you want to add : ");
            scanf("%d",&val);
            insert(val);
         break;
          case 2:
            traversal(root);
            break;
         case 3:
            fl=1;
         break;
         default:
            printf("\nInvalid Choice\n");
      }
      if(fl==1){
         break;}
      }
```

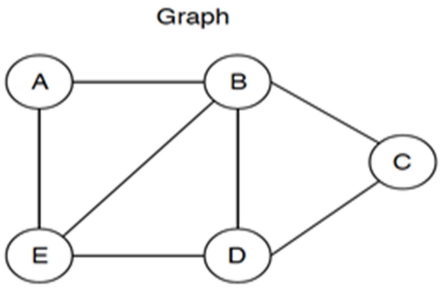
```
    return 0;
}
```

Output:

```
Red Black Tree Menu -
Enter your choice :
1:Insert
2:Delete
3:Search
4:Traversal
5:Exit
1
Enter the integer you want to add : 10
Red Black Tree Menu -
Enter your choice :
1:Insert
2:Delete
3:Search
4:Traversal
5:Exit
1
Enter the integer you want to add : 7
Search Element Found!!
```

```
Red Black Tree Menu -
Enter your choice :
1:Insert
2:Delete
3:Search
4:Traversal
5:Exit
4
root is 5- b
Inorder tree traversal
 2-b  5-b  7-r  8-b  9-r
postorder tree traversal
sh: 1: cls: not found
 2-b  7-r  9-r  8-b  5-b
```

9. Design, Develop and Implement a Program in C for the following operations on Graph (G) of Cities
    a. Create a Graph of N cities using Adjacency Matrix.
    b. Print all the nodes reachable from a given starting node in a digraph using the DFS / BFS method.



/* Traversal using BFS method*/

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

void bfs(int a[10][10], int n, int u)
{
        int     f,r,q[10],v;
        int     s[10]={0};//initialize all elem in s to 0, no node visited
        printf("The nodes visited from %d:",u);
        f=0, r=-1;// Q empty
        q[++r]=u; // Insert u into Q
        s[u]=1; // Insert u to s
        printf("%d",u); //pritn node visited
        while(f<=r)
        {
                u=q[f++];  //del an elem from Q
                for(v=0; v<n; v++)
                {
                        if(a[u][v]==1)          //If v is adjacent to u
                        {
                                if(s[v]==0)  // If v is not in S i.e, V has not been visited
                                {
                                        printf("%d",v); // print the node visited
                                        s[v]=1;         //add v to s,mark as visited
                                        q[++r]=v;       //insert v into Q
                                }
                        }
                }
        }
        printf("\n");
}

void main()
{
        int     n,a[10][10], source, i,j;
        printf("Enter the number of nodes:");
        scanf("%d",&n);
```

```
        printf("Enter the adjacency matrix:\n");
        for(i=0; i<n; i++)
        {
                for(j=0; j<n; j++)
                {
                        scanf("%d", &a[i][j]);
                }
        }
        for(source=0; source<n; source++)
        {
                bfs(a,n,source);
        }

}
```

OUTPUT:

```
Enter the number of nodes: 4
Enter the adjacency matrix:
0 1 1 0
0 0 1 1
0 0 0 1
0 0 0 0
The nodes visited from 0: 0 1 2 3
The nodes visited from 1: 1 2 3
The nodes visited from 2: 2 3
The nodes visited from 3: 3
```

/*Traversal using DFS method*/

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

Int a[10][10], s[10], n;

void read_AM( int a[10][10], int n)
{
                int     I,j;
                //read n value
                for( i=0; i<n; i++)
                {
                        for (j=0; j<n; j++)
                        {
                                scanf("%d", &a[i][j]);
                        }
                }
}
```

```
Void dfs(int u)
{
        int     v;
        s[u]=1;
        printf("%d",u);
        for(v=0; v<n; v++)
        {
                if(a[u][v]==1 && s[v]==0)
                        dfs(v);
        }
}

void main()
{
        int     i, source;
        printf("Enter the number of nodes in the     graph:");
        scanf("%d",&n);
        printf(Enter the adjacency matrix:");
        read_AM(a,n);
        for(source=0; source<n; source++)
        {
                for(i=0; i<n; i++)
                        s[i]=0;
                printf("The nodes reachable from %d", source);
                dfs(source);
        }
}
```

OUTPUT:

```
Enter the number of nodes: 4
Enter the adjacency matrix:
0 1 1 0
0 0 1 1
0 0 0 1
0 0 0 0
The nodes visited from 0: 0 1 2 3
The nodes visited from 1: 1 2 3
The nodes visited from 2: 2 3
The nodes visited from 3: 3
```

10. Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F.

a. Assume that file F is maintained in memory by a Hash Table (HT) of M memory locations with L as the set of memory addresses (2-digit) of locations in HT.

b.Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash Function H: K%L as I (remainder method), and implement hashing techniques to map a given key K to the address space L.

c.Resolve the collision (if any) using linear probing.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define HASH_SIZE 100

typedef struct employee
{
        int     id;
        char    name[20];
}EMPLOYEE;

/*Create initial hash table*/

void initialize_hash_table(EMPLOYEE a[])
{
        int     i;
        for(i=0; i<HASH_SIZE; i++)
        {
                a[i].id=0;
        }
}

/*Compute hash value using the function: H(K)=k%m*/

int H(int k)
{
        return k% HAHS_SIZE;
}


/*Insert an item into the empty slot using linear probing*/

void insert_table(int id, char name[], EMPLOYEE a[])
{
        int     i,index,h_value;
        h_value= H(id);
        for(i=0; i<HASH_SIZE; i++)
        {
                index=(h_value+i)% HASH_SIZE;
                if(a[index].id==0)              //empty slot found
                {
                        a[index].id=id;         //insert employee id
```

```
                        strcpy(a[index].name,name); //insert employee name
                        break;
                }
        }
        if(i==HASH_SIZE) printf("Hash table full\n"); // NO empty slot
}

/*Display the hash table*/
void display_Hash_table(EMPLOYEE a[], int n)
{
        int     k,i;
        printf("H_Value\t Emp_id\t Emp_name\n");
        for(i=0; i<HASH_SIZE; i++)
        {
                If(a[i].id!=0)
                        printf("%d\t %d\t %s\n",i,a[i].id,a[i].name);
        }
}

void main()
{
        EMPLOYEE a[HASH_SIZE];
        char    name[20];
        int     key,id,choice;
        initialize_hash_table(a);       //Create intial hash table
        for(;;)
        {
                printf("1:Insert 2:Display 4:Exit\n");
                printf("Enter the choice:");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:printf("Enter emp id:");
                                scanf("%d",&id);
                                printf("Enter the name:");
                                scanf("%s",name);
                                insert_hash_table(id,name,a);
                                break;
                        case 2: printf("Contents of hash table\n");
                                display_hash_table(a,HASH_SIZE);
                                printf("\n");
                                break;
                        case 3: exit(0);
                        deafult: printf(Enter valid choice\n”);
                }
        }
}
```

OUTPUT:

```
1: Insert 2: Display 3: Exit
Enter the choice: 1
Enter the empid: 1234
Enter the name: Anu
1: Insert 2: Display 3: Exit
Enter the choice: 1
Enter the empid: 1236
Enter the name: Anil
1: Insert 2: Display 3: Exit
Enter the choice: 1
Enter the empid: 1334
Enter the name: Charan
1: Insert 2: Display 3: Exit
Enter the choice: 2
Contents of hash table
H_value Emp_id Emp_name
34        1234     Anu
35        1334     Charan
36        1236     Anil
1: Insert 2: Display 3: Exit
Enter the choice: 3
```

VIVA QUESTIONS AND ANSWERS

1) What is data structure?

Data structures refer to the way data is organized and manipulated. It seeks to find ways to make data access more efficient. When dealing with data structure, we not only focus on one piece of data, but rather different set of data and how they can relate to one another in an organized manner.

2) Differentiate file structure from storage structure.

Basically, the key difference is the memory area that is being accessed. When dealing with the structure that resides the main memory of the computer system, this is referred to as storage structure. When dealing with an auxiliary structure, we refer to it as file structures.

3) When is a binary search best applied?

A binary search is an algorithm that is best applied to search a list when the elements are already in order or sorted. The list is search starting in the middle, such that if that middle value is not the target search key, it will check to see if it will continue the search on the

lower half of the list or the higher half. The split and search will then continue in the same manner.

4) What is a linked list?
A linked list is a sequence of nodes in which each node is connected to the node following it. This forms a chain-like link of data storage.

5) How do you reference all the elements in a one-dimension array?
To do this, an indexed loop is used, such that the counter runs from 0 to the array size minus one. In this manner, we are able to reference all the elements in sequence by using the loop counter as the array subscript.

6) In what areas do data structures applied?
Data structures is important in almost every aspect where data is involved. In general, algorithms that involve efficient data structure is applied in the following areas: numerical analysis, operating system, A.I., compiler design, database management, graphics, and statistical analysis, to name a few.

7) What is LIFO?
LIFO is short for Last In First Out, and refers to how data is accessed, stored and retrieved. Using this scheme, data that was stored last , should be the one to be extracted first. This also means that in order to gain access to the first data, all the other data that was stored before this first data must first be retrieved and extracted.

8) What is a queue?
A queue is a data structure that can simulates a list or stream of data. In this structure, new elements are inserted at one end and existing elements are removed from the other end.

9) What are binary trees?
A binary tree is one type of data structure that has two nodes, a left node and a right node. In programming, binary trees are actually an extension of the linked list structures.

10) Which data structures is applied when dealing with a recursive function?
Recursion, which is basically a function that calls itself based on a terminating condition, makes use of the stack. Using LIFO, a call to a recursive function saves the return address so that it knows how to return to the calling function after the call terminates.

11) What is a stack?
A stack is a data structure in which only the top element can be accessed. As data is stored in the stack, each data is pushed downward, leaving the most recently added data on top.

12) Explain Binary Search Tree

A binary search tree stores data in such a way that they can be retrieved very efficiently. The left subtree contains nodes whose keys are less than the node's key value, while the right subtree contains nodes whose keys are greater than or equal to the node's key value. Moreover, both subtrees are also binary search trees.

13) Are linked lists considered linear or non-linear data structures?

It actually depends on where you intend to apply linked lists. If you based it on storage, a linked list is considered non-linear. On the other hand, if you based it on access strategies, then a linked list is considered linear.

14) How does dynamic memory allocation help in managing data?

Aside from being able to store simple structured data types, dynamic memory allocation can combine separately allocated structured blocks to form composite structures that expand and contract as needed.

15) What is FIFO?

FIFO is short for First-in, First-out, and is used to represent how data is accessed in a queue. Data has been inserted into the queue list the longest is the one that is removed first.

16) What is an ordered list?

An ordered list is a list in which each node's position in the list is determined by the value of its key component, so that the key values form an increasing sequence, as the list is traversed.

17) What is merge sort?

Merge sort takes a divide-and-conquer approach to sorting data. In a sequence of data, adjacent ones are merged and sorted to create bigger sorted lists. These sorted lists are then merged again to form an even bigger sorted list, which continuous until you have one single sorted list.

18) Differentiate NULL and VOID.

Null is actually a value, whereas Void is a data type identifier. A variable that is given a Null value simply indicates an empty value. Void is used to identify pointers as having no initial size.

19) What is the primary advantage of a linked list?

A linked list is a very ideal data structure because it can be modified easily. This means that modifying a linked list works regardless of how many elements are in the list.

20) What is the difference between a PUSH and a POP?

Pushing and popping applies to the way data is stored and retrieved in a stack. A push denotes data being added to it, meaning data is being "pushed" into the stack. On the other hand, a pop denotes data retrieval, and in particular refers to the topmost data being accessed.

21) What is a linear search?

A linear search refers to the way a target key is being searched in a sequential data structure. Using this method, each element in the list is checked and compared against the target key, and is repeated until found or if the end of the list has been reached.

22) How does variable declaration affect memory allocation?

The amount of memory to be allocated or reserved would depend on the data type of the variable being declared. For example, if a variable is declared to be of integer type, then 32 bits of memory storage will be reserved for that variable.

23) What is the advantage of the heap over a stack?

Basically, the heap is more flexible than the stack. That's because memory space for the heap can be dynamically allocated and de-allocated as needed. However, memory of the heap can at times be slower when compared to that stack.

24) What is a postfix expression?

A postfix expression is an expression in which each operator follows its operands. The advantage of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.

25) What is the difference between the HEAP and the STACK?

(Solution: HEAP is used to store dynamically allocated memory (malloc). STACK stores static data (int, const).)

26) Describe the data structures of a double-linked list.

(Solution: A double-linked list structure contains one pointer to the previous record in the list and a pointer to the next record in the list plus the record data.)

27) How do you insert a record between two nodes in double-linked list?

(Solution: Previous R; Data R; Next R; To insert a record (B) between two others (A and C): Previous.B = A; Next.B = C; Next.A = B; Previous.C = B;)

28) In which data structure, elements can be added or removed at either end, but not in the middle?

(Solution: queue)

29) Which one is faster? A binary search of an orderd set of elements in an array or a sequential search of the elements.
(Solution: binary search)

30) What is a balanced tree?
(Solution: A binary tree is balanced if the depth of two subtrees of every node never differ by more than one)

31) Which data structure is needed to convert infix notations to post fix notations?
(Solution: stack)

32) What is data structure or how would you define data structure?
(Solution: In programming the term data structure refers to a scheme for organizing related piece of information. Data Structure = Organized Data + Allowed Operations.)

33) Which data structures we can implement using link list?
(Solution: queue and stack)

34) List different types of data structures?
(Solution: Link list, queue, stack, trees, files, graphs)

SAMPLE PROGRAM EXECUTION

//Program on FIBONACCI

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int main()
{
   int t1,t2,n,i;
   long double t[20];
   scanf("%d%d%d",&t1,&t2,&n);
   if(n>20){return 0;}

   t[0]=t1;t[1]=t2;
   for(i=0;i<=n-1;i++)
   {
         t[i+2]=t[i]+t[i+1]*t[i+1];
   }
```

```
   printf("%Lf",t[n-1]);
   getch();
   return 0;
}
```

//Program on Concatination of strings

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 100
char s1[MAX]="amobile",*s=s1;
char s2[MAX]="uto",*t=s2;

void strnins(char *s,char *t,int i)
{
        char string[MAX],*temp=string;
        if(i<0 && i>strlen(s))
        {
            printf("Position is out of boundary\n");
            exit(0);
        }
        if(!strlen(s))
                strcpy(s,t);
        else if(strlen(t))
        {
                strncpy(temp,s,i);
                strcat(temp,t);
                strcat(temp,(s+i));
                strcpy(s,temp);
        }
}



void main()
{
        int i=1;
        strnins(s1,s1,i);
        printf("The concat string is:%s",s1);
        getch();
}
```

//Program on Sparse matrix

```
#include<stdio.h>
#include<conio.h>
#define MAX 101
```

```
typedef struct
{
	int row,col,val;
}TERM;

void read_sparse_matrix(TERM a[],int m,int n,int v)
{
	int i,j,k, item;
	a[0].row=m;
	a[0].col=n;
	a[0].val=v;
	k=1;
    //  printf("enter the no. of non zero entries\n");
    //  scanf("%d",&value);
	//a[0].val=value;
	for(i=0; i<m; i++)
	{
		for(j=0; j<n; j++)
		{
			scanf("%d",&item);
			if(item==0)
				continue;
			a[k].row=i;
			a[k].col=j;
			a[k].val=item;
			k++;
		}
	}
	a[0].val=k-1;
}

void search(int item, TERM a[])
{
	int i,j;
	for(i=0; i<=a[0].val; i++)
	{
		if(item==a[i].val)
		{
			printf("Search is successful");
			exit(0);
		}
	}
	printf("Search is unsuccessful");
}

void main()
```

```
{
        int m,n,item,v;
        TERM a[MAX];
        clrscr();
        printf("Enter the number of rows");
        scanf("%d",&m);
        printf("Enter the number of cols");
        scanf("%d",&n);
        printf("Enter the number of non zero elements");
        scanf("%d",&v);
        read_sparse_matrix(a,m,n,v);
        printf("Enter the element to be searched\n");
        scanf("%d",&item);
        search(item,a);
        getch();
}
```

//Program on MALLOC, CALLOC

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int *p;
        int n,i;
        clrscr();
        printf("Enter the number of elements\n");
        scanf("%d",&n);
        p=(int *)malloc(n*sizeof(int));
        if(p==NULL)
        {
                printf("Enough memory not available\n");
                exit(0);
        }
        printf("Enter array elements\n");
        for(i=0; i<n; i++)
        {
                scanf("%d",p+i);
        }
        printf("Array elements are\n");
        for(i=0; i<n; i++)
        {
                printf("%d\t",*(p+i));
        }
        getch();
}
```

```c
#include<stdio.h>
#include<conio.h>
void main()
{
	int *p;
	int n,i;
	clrscr();
	printf("Enter the number of elements\n");
	scanf("%d",&n);
	p=(int *)calloc(n,sizeof(int));
	if(p==NULL)
	{
		printf("Enough memory not available\n");
		exit(0);
	}
	printf("Enter array elements\n");
	for(i=0; i<n; i++)
	{
		scanf("%d",p+i);
	}
	printf("Array elements are\n");
	for(i=0; i<n; i++)
	{
		printf("%d\t",*(p+i));
	}
	getch();
}
```

//Program on UNIONS

```c
#include<stdio.h>
void main()
{
	typedef union
	{
		int marks;
		char grade;

	}student;
	student x;

	clrscr();
	x.marks=100;
		x.grade='A';
	printf("marks=%d\n",x.marks);

	printf("Grade=%c\n",x.grade);
```

```
        getch();
}
```

//Program on array operations (Update,search, delete, insert)
//Insert operation

```
#include <stdio.h>

main()
{
        int LA[] = {1,3,5,7,8};
        int item = 10, k = 3, n = 5;
        int i = 0, j = n;

         printf("The original array elements are :\n");

        for(i = 0; i<n; i++)
        {
                printf("LA[%d] = %d \n", i, LA[i]);
        }

        n = n + 1;

        while( j >= k) {
                 LA[j+1] = LA[j];
                j = j - 1;
        }
        LA[k] = item;

        printf("The array elements after insertion :\n");

        for(i = 0; i<n; i++) {
                printf("LA[%d] = %d \n", i, LA[i]);
        }
}
```

//Delete operation

```
#include <stdio.h>
main() {
  int LA[] = {1,3,5,7,8};
  int k = 3, n = 5;
  int i, j;

  printf("The original array elements are :\n");

  for(i = 0; i<n; i++) {
    printf("LA[%d] = %d \n", i, LA[i]);
```

```
  }

  j = k;

  while( j < n) {
    LA[j-1] = LA[j];
    j = j + 1;
  }

  n = n -1;

  printf("The array elements after deletion :\n");

  for(i = 0; i<n; i++) {
    printf("LA[%d] = %d \n", i, LA[i]);
  }
}
```

//Search operation

```
#include <stdio.h>
main() {
  int LA[] = {1,3,5,7,8};
  int item = 5, n = 5;
  int i = 0, j = 0;

  printf("The original array elements are :\n");

  for(i = 0; i<n; i++) {
    printf("LA[%d] = %d \n", i, LA[i]);
  }

  while( j < n){
    if( LA[j] == item ) {
      break;
    }

    j = j + 1;
  }

  printf("Found element %d at position %d\n", item, j+1);
}
```

//Update operation

```
#include <stdio.h>
main()
{
  int LA[] = {1,3,5,7,8};
  int k = 3, n = 5, item = 10;
```

```
  int i, j;

  printf("The original array elements are :\n");

  for(i = 0; i<n; i++) {
    printf("LA[%d] = %d \n", i, LA[i]);
  }

  LA[k-1] = item;

  printf("The array elements after updation :\n");

  for(i = 0; i<n; i++) {
    printf("LA[%d] = %d \n", i, LA[i]);
  }
}
```