DATA STRUCTURE USING 'C' LAB
(CSE-231-F)

LAB MANUAL

III SEMESTER

DRONACHARYA
College of Engineering

Department Of Electronics & Communication Engg
Dronacharya College Of Engineering
Khentawas, Gurgaon – 123506

CONTENTS

SUBJECT: DATA STRUCTURE USING 'C' LAB (CSE-231-F)

EXPERIMENT NO.1

AIM: - Write a program to search an element in a two-dimensional array using linear search.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,item,loc=0,loc1=0;
int a[2][2];
clrscr();
printf("\n\tThis Program is Used To seaech an element in 2Dimensional Array using Linear
Search\n");
printf("\n\tEnter The Value Of Array:");
for(i=1;i<=2;i++)
{
for(j=1;j<=2;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\n\tEnter The Value To Be Serched:");
scanf("%d",&item);
for(i=1;i<=2;i++)
{
for(j=1;j<=2;j++)
{
if(item==a[i][j])
{
loc=i;
loc1=j;
break;
}
}
}

printf("\n\tThe Item is at %d Row And %d Coloumn.",loc,loc1);
printf("\n\n\t\tSearch Completed.");
printf("\n\n\n\t\t\tTHANKYOU!");
getch();
}
```

QUIZ

Q1. How does an array differ from an ordinary variable?

Ans.

Array Vs. Ordinary Variable

Array is made up of similar data structure that exists in any language. Array is set of similar data types. Array is the collection of similar elements. These similar elements could be all int or all float or all char etc. Array of char is known as string. All elements of the given array must be of same type. Array is finite ordered set of homogeneous elements. The number of elements in the array is pre-specified.

An ordinary variable of a simple data type can store a single element only.

For example: *Ordinary variable*: - int a

Array: - int a[10]

Q2. Two dimensional arrays are also called

a. tables arrays
b. matrix arrays
c. both of above
d. none of above

Ans. Matrix arrays

Q3. Which data structure can't store the non-homogeneous data elements?

Ans. Array

Q4. What is the main difference between ARRAY and STACK?

Ans:Stack follows LIFO. Thus the item that is first entered would be the last removed.In array the items can be entered or removed in any order. Basically each member access is done using index and no strict order is to be followed here to remove a particular element .Array may be multi dimensional or one dimensional but stack should be one-dimensional.

Size of array is fixed, while stack can be grow or shrink. We can say stack is dynamic data structure.

Q5. What is linear search technique?

Ans: **linear search** or **sequential search** is a method for finding a particular value in a list that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found.

Q6. What is the use of clrscr() function?

Ans: To clear the previous result from output screen.

Q7. What is an array?

Ans: An array is defined as a collection of homogeneous elements.

Q8. How to declare a two dimensional array?

Ans: data_type arr_name[][];

Q9. What are the input and output functions used in this program?

Ans: printf() and scanf()

Q10. What is sparse matrix?

Ans: A **sparse matrix** is a matrix populated primarily with zeros

EXPERIMENT NO.2

AIM: Using iteration & recursion concepts write programs for finding the element in the array using Binary Search Method

PROGRAM USING RECURSION

```
#include <stdio.h>
binarysearch(int a[],int n,int low,int high)
{
 int mid;
 if (low > high)
return -1;
mid = (low + high)/2;
if(n == a[mid])
{
 printf("The element is at position %d\n",mid+1);
return 0;
}
if(n < a[mid])
{
high = mid - 1;
binarysearch(a,n,low,high);
}
 if(n > a[mid])
{
low = mid + 1;
binarysearch(a,n,low,high);
}
}
main()
{
int a[50];
int n,no,x,result;
printf("Enter the number of terms : ");
scanf("%d",&no);
printf("Enter the elements :\n");
for(x=0;x&ltno;x++)
scanf("%d",&a[x]);
printf("Enter the number to be searched : ");
scanf("%d",&n);
```

```
result = binarysearch(a,n,0,no-1);
if(result == -1)
printf("Element not found");
 return 0;
}
```

PROGRAM USING ITERATION

```
#include<stdio.h>
#include<conio.h>
int nr_bin_search(int[],int,int);
void main()
{
int key,i,n,index,l[20];
printf("\n enter the number of elements in the list:");
scanf("%d",n);
printf("\n enter the elements of the list:");
for(i=0;i<n;i++)
scanf("%d",&l[i]);
printf("\n enter the key element to be searched in the
list:");
scanf("%d",&key);
index=nr_bin_search(l,n,key);
if(index==-1)
printf("\n search completed,element%d found in the list at
position %d",key,index);
getch();
}
int nr_bin_search(ints[],int n,int e)
{
int low_val,mid_val,high_val;
low_val=0;
high_val=0;
while(high_val>=low_val)
{
mid_val=(low_val+high_val)/2;
if(s[mid_val]==e)
return(mid_val);
if(s[mid_val]<e)
low_val=mid_val+1;
else
high_val=mid_val-1;
}
return-1;
}
```

QUIZ

Q1. The complexity of searching an element from a set of n elements using Binary search algorithm is
(A) O(n) (B) O(log n)
(C) O(n2) (D) O(n log n)
Ans:B
Q2. In binary search, average number of comparison required for searching an element in a list if n numbers is
(A) log2 n . (B) n / 2 .
(C) n. (D) n – 1.
Ans. (A)
Q3 Which of the following is not the required condition for binary search algorithm?
a. The list must be sorted
b. there should be the direct access to the middle element in any sublist
c. There must be mechanism to delete and/or insert elements in list
d. none of above
Ans:There must be mechanism to delete and/or insert elements in list
Q4 Which of the following is not a limitation of binary search algorithm?
a. must use a sorted array
b. requirement of sorted array is expensive when a lot of insertion and deletions are needed
c. there must be a mechanism to access middle element directly
d. binary search algorithm is not efficient when the data elements are more than 1000.
Ans: requirement of sorted array is expensive when a lot of insertion and deletions are needed
Q5. What is binary search technique?
Ans: In computer science, a **binary search** or **half-interval search** algorithm finds the position of a specified value (the input "key") within a sorted array.
Q6. What is the maximum number of leaves in a binary tree of height H?
Ans: 2^H
Q7. What is the difference between recursion and iteration?
Ans: During recursion, a global variable could be used to control the depth of recursion, or data pushed onto some stack could provide a termination condition; and in the latter case this termination condition is usually related to the form of the data structure being traversed.
Q8. What is the main difference between linear and binary search?
Ans: A linear search looks down a list, one item at a time, without jumping. In complexity terms this is an O(n) search - the time taken to search the list gets bigger at the same rate as the list does. A binary search is when you start with the middle of a sorted list, and see whether that's greater than or less than the value you're looking for, which determines whether the value is in the first or second half of the list.
Q9. Give a real world example of binary search technique.
Ans: Number guessing game, word lists.
Q10. What is noisy binary search?
Ans: **binary search** solves the same class of projects as regular binary search, with the added complexity that any given test can return a false value at random.

EXPERIMENT NO.3

AIM: Write a program to perform following operations on tables using functions only
a) Addition b) Subtraction c) Multiplication d) Transpose

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int a[4][4],b[4][4],c[4][4],i,j;
printf("enter the elements of matrix a");
for(i=0;i<=3;i++)
for(j=0;j<=3;j++)
scanf("%d",&a[i][j]);
printf("the first matrix is");
for(i=0;i<=3;i++)
{
printf("\n");
for(j=0;j<=3;j++)
printf("%d",a[i][j]);
}
printf("Enter the elements of second matrix");
for(i=0;i<=3;i++)
for(j=0;j<=3;j++)
scanf("%d",&b[i][j]);
printf("the second matrix is");
for(i=0;i<=3;i++)
{
printf("\n");
for(j=0;j<=3;j++)
printf("%d",b[i][j]);
}
for(i=0;i<=4;i++)
for(j=0;j<=4;j++)
c[i][j]=a[i][j] + b[i][j];
printf("the addition of matrix is");
for(i=0;i<=3;i++)
{
for(j=0;j<=3;j++)
printf("%d\t",c[i][j]);
```

```
printf("\n");
}
printf("\nSubtraction of matrix is\n");
for(i=0;i<r1;i++)
{
for(j=0;j<c2;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
}
else
printf("\nSubraction is not possible pls enter same order");
}
Printf("Multiplication is");
for(i=0;i<=3;i++)
{
for(j=0;j<=3;j++)
{
for(k=0;k<=3;k++)
{
c[i][j]=c[i][j]=c[k][j]+a[i][k]*b[k][j];
}
}
printf("multiplication matrix is");
for(i=0;i<=5;i++)
for(j=0;j<=5;j++)
printf("%d",c[i][j]):
}
printf("Transpose of the Matrix :\n\n");
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
t=m[i][j];
m[i][j]=m[j][i];
m[j][i]=t;
}
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("\t%d",m[i][j]);
```

```
}
printf("\n");
}
getch();
}
```

QUIZ

Q1. What is a matrix?
Ans: A matrix is a representation of certain rows and columns, to persist homogeneous data.
Q2. What are the uses of matrix?
Ans: Uses:
- To represent class hierarchy using Boolean square matrix
- For data encryption and decryption
- To represent traffic flow and plumbing in a network
- To implement graph theory of node representation

Q3. How we declare two-dimensional array?
Ans: int a[2][2];
Q4. What is sparse matrix?
Ans: A **sparse matrix** is a matrix populated primarily with zeros.
Q5. What is transpose of a matrix?
Ans: The **transpose** of a matrix **A** is another matrix $\mathbf{A}^T$ (also written $\mathbf{A}'$, $\mathbf{A}^{tr}$ or $\mathbf{A}^t$) created by any one of the following equivalent actions:

- reflect **A** over its main diagonal (which runs top-left to bottom-right) to obtain $\mathbf{A}^T$
- write the rows of **A** as the columns of $\mathbf{A}^T$
- write the columns of **A** as the rows of $\mathbf{A}^T$

Q6 Define rank of matrix.
Ans: The rank of a matrix **A** is the maximum number of linearly independent row vectors of the matrix, which is the same as the maximum number of linearly independent column vectors.
Q7. What is invertible matrix?
Ans: A square matrix **A** is called invertible or non-singular if there exists a matrix **B** such that

$$\mathbf{AB} = \mathbf{I}_n$$

Q8.What is diagonal matrix?
Ans: If all entries outside the main diagonal are zero, **A** is called a diagonal matrix.
Q9. What is symmetric matrix?
Ans: A square matrix **A** that is equal to its transpose, i.e., $\mathbf{A} = \mathbf{A}^T$, is a symmetric matrix.
Q10. Write a real world application of matrix.
Ans: Game theory.

EXPERIMENT NO.4

AIM: Using iteration & recursion concepts write the programs for Quick Sort Technique

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define max 100
int a[max],n,i,h,l;
void main()
{
void input(void);
input();
getch();
}
void input(void)
{
void quicksort(int a[],int l,int h);
void output(int a[],int n);
printf("how many:");
scanf("%d",&n);
printf("\n");
printf("Enter the elements:\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
l=0;
h=n-1;
quicksort(a,l,h);
printf("sorted array:\n");
output(a,n);
}
void quicksort(int a[],int l,int h)
{
int temp,key,low,high;

low=l;
high=h;
key=a[(low+high)/2];
do{
while(key>a[low])
{
```

```
low++;
}
while(key<a[high])
{
high--;
}
if(low<=high)
{
temp=a[low];
a[low++]=a[high];
a[high--]=temp;
}
}
while(low<=high);
if(l<high)
quicksort(a,l,high);
if(low<h)
quicksort(a,low,h);
}
void output(int a[],int n)
{
for(i=0;i<n;i++)
{
printf("%d\n",a[i]);
}
}
```

QUIZ

Q1. What would be the order of Quick Sort in worst case?
Ans: O (n2/2)
Q2. The quick sort algorithm exploits _____ design technique
(A) Greedy **(B)** Dynamic programming
(C) Divide and Conquer **(D)** Backtracking
Ans: C
Q3. Quick sort is also known as
(A) merge sort (B) heap sort
(C) bubble sort (D) none of these
Ans:D
Q4. What is quick sort?
Ans *:*Quick sort is one ***of*** the fastest sorting algorithm used for sorting a list. A pivot point is chosen. Remaining elements are portioned or divided such that elements less than the pivot point are in left and those greater than the pivot are on the right. Now, the elements on the left and right can be recursively sorted by repeating the algorithm.
Q5. What is divide and conquer algorithm?
Ans: Algorithms that solve (conquer) problems by dividing them into smaller sub-problems until the problem is so small that it is trivially solved.
Q6. What is in place algorithm?
Ans: In place sorting algorithms don't require additional temporary space to store elements as they sort; they use the space originally occupied by the elements.
Q7. Define sorting algorithm.
Ans: A **sorting algorithm** is an algorithm that puts elements of a list in a certain order.
Q8. What are the other techniques for sorting?
Ans: Bubble sort, insertion sort, selection sort, merge sort, radix sort, heap sort.
Q9. What is the possible way to classify sorting algorithms?
Ans: Complexity
Q10. What is swap based sorting?
Ans: Swap-based sorts begin conceptually with the entire list, and exchange particular pairs of elements (adjacent elements or elements with certain step like in Shell sorts) moving toward a more sorted list.

EXPERIMENT NO.5

AIM: Write a program to implement the various operations on string such as length of string concatenation, reverse of a string & copy of a string to another.

PROGRAM

```
#include<conio.h>
#include<stdio.h>
void main()
{
char a[20], b[20];
int i;
clrscr();
gets(a);
i=0;
while(a[i]!='\0')
i++; //counts no of chars till encountering null char
printf(string length=%d,i);
printf("Before concatenation:"
" \n string1 = %s \n string2 = %s", string1, string2);
finalstr = strcat(string1, string2);
printf("\nAfter concatenation:");
printf("\n finalstr = %s", finalstr);
printf("\n string1 = %s", string1);
printf("\n string2 = %s", string2);
printf("Enter the string to be reversed : ");
 scanf("%s",str);
 for(i=strlen(str)-1;i>=0;i--)
 {
 revstr[j]=str[i];
 j++;
 }
 revstr[j]='\0';
 printf("Input String : %s",str);
 printf("\nOutput String : %s",revstr);
CHAR szTest[32] = {"The Quick Brown Fox"};
CHAR szCopy[32] = {0};
size_t thelen = strlen(szTest);
int i = 0;
for(;i<(int)thelen;i++)
{
szCopy[i] = szTest[i];
getch();
}
```

QUIZ

Q1. Define string.
Ans: A string is traditionally a sequence of characters, either as a literal constant or as some kind of variable.
Q2. What is literal constant?
Ans: A **literal** is a notation for representing a fixed value in source code.
Q3. When a string is said to be prefix?
Ans: A string s is said to be a prefix of t if there exists a string u such that t = su. If u is nonempty, s is said to be a proper prefix of t.
Q4. When a string is said to be suffix?
Ans: A string s is said to be a suffix of t if there exists a string u such that t = us. If u is nonempty, s is said to be a proper suffix of t.
Q5. Which function we use to calculate length of string?
Ans: strlen(s)
Q6. Which function we use for concatenation of strings?
Ans: strcat(s1,s2)
Q7. What is string datatype?
Ans: A **string datatype** is a datatype modeled on the idea of a formal string. Strings are such an important and useful datatype that they are implemented in nearly every programming language. In some languages they are available as primitive types and in others as composite types.
Q8. Which function we use for comparison of two strings?
Ans: strcmp(s1,s2).
Q9. How a string is declared?
Ans: char string[] = "Hello, world!";
Q10. How we mark the end of a string?
Ans: The end of the string is marked with a special character, the *null character*, which is simply the character with the value 0.

EXPERIMENT NO.6

AIM: Write a program for swapping of two numbers using 'call by value' and 'call by reference strategies.

Call By Value

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

main()
{
  int x, y, temp;
  printf("Enter the value of x and y ");
  scanf("%d%d",&x, &y);
  printf("Before Swapping\nx = %d\ny = %d\n",x,y);
  temp = x;
  x = y;
  y = temp;
  printf("After Swapping\nx = %d\ny = %d\n",x,y);
  getch();
  return 0;
}
```

Call By Reference

```
#include<stdio.h>
#include<conio.h>
main()
{
int i, j;
clrscr();
printf("Please Enter the First Number in A : ");
scanf("%d",&i);
printf("\nPlease Enter the Second Number in B : ");
scanf("%d",&j);
swapr(&i,&j);
printf("A is now in B : %d",i);
printf("B is now in A : %d",j);
}
swapr(int *x, int *y)
{
int t;
t=*x;
*x=*y;
*y=t; }
```

QUIZ

Q1. What is function?
Ans: A function definition specifies the name of the function, the types and number of parameters it expects to receive, and its return type. A function definition also includes a function body with the declarations of its local variables, and the statements that determine what the function does.
Q2. Difference between Call by value and call by reference.
Ans: The arguments passed to function can be of two types

1. Values passed

2. Address passed

The first type refers to call by value and the second type refers to call by reference.

Q3. What is function prototype?
Ans: A function prototype tells the compiler what kind of arguments a function is looking to receive and what kind of return value a function is going to give back. This approach helps the compiler ensure that calls to a function are made correctly and that no erroneous type conversions are taking place.
Q4. When should we declare a function?
Ans: Function declaration should be declared in the current source file along with the definition of the function.
Q5. What is the scope of function variable?
Ans: Variables declared within the calling function can't be accessed unless they are passed to the called function as arguments. The only other contact a function might have with the outside world is through global variables.
Q6. Define recursive function.
Ans: A recursive function is one which calls itself.
Q7. What is the use of making a function inline?
Ans: The point of making a function `inline` is to hint to the compiler that it is worth making some form of extra effort to call the function faster than it would otherwise - generally by substituting the code of the function into its caller.
Q8. What is the return type of the function with prototype: "int func(char x, float v, double t);"
Ans: int
Q9. Which of the following is a valid function call (assuming the function exists)?
A. funct;
B. funct x, y;
C. funct();
D. int funct();
Ans: C
Q10. Which is not a proper prototype?
A. int funct(char x, char y);
B. double funct(char x)
C. void funct();
D. char x();
Ans: B

EXPERIMENT NO.7

AIM: Write a program to implement binary search tree.
(Insertion and Deletion in Binary search Tree)

```
# include<stdio.h>
# include<conio.h>
struct node
{
int info;
struct node *lchild;
struct node *rchild;
}*root;
main()
{
int choice,num;
root=NULL;
while(1)
{
printf("\n");
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter the number to be inserted : ");
scanf("%d",&num);
insert(num);
break;
case 2:
printf("Enter the number to be deleted : ");
scanf("%d",&num);
del(num);
break;
case 3:
display(root,1);
break;
case 4:
exit();
default:
```

```
printf("Wrong choice\n");
}
}
}
find(int item,struct node **par,struct node **loc)
{
struct node *ptr,*ptrsave;
if(root==NULL) /*tree empty*/
{
*loc=NULL;
*par=NULL;
return;
}
if(item==root->info) /*item is at root*/
{
*loc=root;
*par=NULL;
return;
}
/*Initialize ptr and ptrsave*/
if(iteminfo)
ptr=root->lchild;
else
ptr=root->rchild;
ptrsave=root;
while(ptr!=NULL)
{
if(item==ptr->info)
{ *loc=ptr;
*par=ptrsave;
return;
}
ptrsave=ptr;
if(iteminfo)
ptr=ptr->lchild;
else
ptr=ptr->rchild;
}/*End of while */
*loc=NULL; /*item not found*/
*par=ptrsave;
}/*End of find()*/

insert(int item)
{
```

```
struct node *tmp,*parent,*location;
find(item,&parent,&location);
if(location!=NULL)
{
printf("Item already present");
return;
}
tmp=(struct node *)malloc(sizeof(struct node));
tmp->info=item;
tmp->lchild=NULL;
tmp->rchild=NULL;

if(parent==NULL)
root=tmp;
else
if(iteminfo)
parent->lchild=tmp;
else
parent->rchild=tmp;
}/*End of insert()*/

del(int item)
{
struct node *parent,*location;
if(root==NULL)
{
printf("Tree empty");
return;
}

find(item,&parent,&location);
if(location==NULL)
{
printf("Item not present in tree");
return;
}

if(location->lchild==NULL && location->rchild==NULL)
case_a(parent,location);
if(location->lchild!=NULL && location->rchild==NULL)
case_b(parent,location);
if(location->lchild==NULL && location->rchild!=NULL)
case_b(parent,location);
if(location->lchild!=NULL && location->rchild!=NULL)
case_c(parent,location);
```

```
free(location);
}/*End of del()*/

case_a(struct node *par,struct node *loc )
{
if(par==NULL) /*item to be deleted is root node*/
root=NULL;
else
if(loc==par->lchild)
par->lchild=NULL;
else
par->rchild=NULL;
}/*End of case_a()*/

case_b(struct node *par,struct node *loc)
{
struct node *child;

/*Initialize child*/
if(loc->lchild!=NULL) /*item to be deleted has lchild */
child=loc->lchild;
else /*item to be deleted has rchild */
child=loc->rchild;

if(par==NULL ) /*Item to be deleted is root node*/
root=child;
else
if( loc==par->lchild) /*item is lchild of its parent*/
par->lchild=child;
else /*item is rchild of its parent*/
par->rchild=child;
}/*End of case_b()*/

case_c(struct node *par,struct node *loc)
{
struct node *ptr,*ptrsave,*suc,*parsuc;

/*Find inorder successor and its parent*/
ptrsave=loc;
ptr=loc->rchild;
while(ptr->lchild!=NULL)
{
ptrsave=ptr;
ptr=ptr->lchild;
}
```

```
suc=ptr;
parsuc=ptrsave;

if(suc->lchild==NULL && suc->rchild==NULL)
case_a(parsuc,suc);
else
case_b(parsuc,suc);

if(par==NULL) /*if item to be deleted is root node */
root=suc;
else
if(loc==par->lchild)
par->lchild=suc;
else
par->rchild=suc;

suc->lchild=loc->lchild;
suc->rchild=loc->rchild;
}/*End of case_c()*/

display(struct node *ptr,int level)
{
int i;
if ( ptr!=NULL )
{
display(ptr->rchild, level+1);
printf("\n");
for (i = 0; i < level; i++)
printf(" ");
printf("%d", ptr->info);
display(ptr->lchild, level+1);
}/*End of if*/
}/*End of display()*/
```

QUIZ

Q1. What is the number of possible ordered trees with three nodes?
Ans: 12
Q2. What is strictly binary tree?
Ans: A binary tree in which every non leaf node has non empty left and right subtrees is called a strictly binary tree.
Q3. What is the depth of a complete binary tree with n nodes?
Ans: log(n+1)-1
Q4. which kind of traversal is similar to preorder?
Ans: Depth-first order
Q5. Which traversal technique lists the nodes of a binary search tree in ascending order?
Ans: In-order
Q6. What is the order of binary search algorithm?
Ans: log(n)
Q7. A binary tree has n leaf nodes. What would be the number of nodes of degree 2 in this tree?
Ans: n-1
Q8. What is the number of binary trees with 3 nodes which when traversed in post order?
Ans: 5
Q9. What is the infix priorities of +, *, ^, /
Ans: 5,2,2,4
Q10. What is 3-ary tree?
Ans: It is a tree in which every internal node has exactly 3 children.

EXPERIMENT NO.8

AIM: Write a program to create a linked list & perform operations such as insert, delete, update, reverse in the link list

```
#include"stdio.h"
//#define NULL 0
struct node
{
int data;
struct node *next;
}*p;

delnode(int num)
{
struct node *temp, *m;
temp=p;
while(temp!=NULL)
{
if(temp->data==num)
{
if(temp==p)
{
p=temp->next;
free(temp);
return;
}
else
{
m->next=temp->next;
free(temp);
return;
}
}else
{
m=temp;
temp= temp->next;
}

}
printf("
ELEMENT %d NOT FOUND
", num);
```

```
append( int num )
{
struct node *temp,*r;
temp= (struct node *)malloc(sizeof(struct node));
temp->data=num;
r=(struct node *)p;

if (p == NULL)
{
p=temp;
p->next =NULL;
}
else
{ /* GO TO LAST AND ADD*/

while( r->next != NULL)
r=r->next;
r->next =temp;
r=temp;
r->next=NULL;
}
}/* ADD A NEW NODE AT BEGINNING */

addbeg( int num )
{
/* CREATING A NODE AND INSERTING VALUE TO IT */

struct node *temp;
temp=(struct node *)malloc(sizeof(struct node));
temp->data=num;

/* IF LIST IS NULL ADD AT BEGINNING */
if ( p== NULL)
{
p=temp;
p->next=NULL;
}

else
{
temp->next=p;
p=temp;
}
}
```

```
/* ADD A NEW NODE AFTER A SPECIFIED NO OF NODES */

addafter(int num, int loc)
{
int i;
struct node *temp,*t,*r;
r=p; /* here r stores the first location */
if(loc > count()+1 || loc <= 0)
{
printf("
insertion is not possible :
");
return;
}
if (loc == 1)/* if list is null then add at beginning */
{
addbeg(num);
return;
}
else
{
for(i=1;i<loc;i++)
{
t=r; /* t will be holding previous value */
r=r->next;
}
temp=(struct node *)malloc(sizeof(struct node));
temp->data=num;
t->next=temp;
t=temp;
t->next=r;
return;
}
}/* THIS FUNCTION DISPLAYS THE CONTENTS OF THE LINKED LIST */

display(struct node *r)
{
r=p;
if(r==NULL)
{
printf("NO ELEMENT IN THE LIST :");
return;
}
/* traverse the entire linked list */
while(r!=NULL)
```

```
{
printf(" -> %d ",r->data);
r=r->next;
}
printf("<BR>);
}

//THIS FUNCTION REVERSES A LINKED LIST
reverse(struct node *q)
{
struct node *m, *n,*l,*s;
m=q;
n=NULL;
while(m!=NULL)
{
s=n;
n=m;
m=m->next;
n->next=s;
}
p=n;
}

/* THIS IS THE MAIN PROGRAM */

main()
{
int i;
p=NULL;
while(1) /* this is an indefinite loop */
{
printf("
1.INSERT A NUMBER AT BEGINNING;<BR>);
printf("
2.INSERT A NUMBER AT LAST:<BR>);
printf("
3.INSERT A NUMBER AT A PARTICULAR LOCATION INlIST:<BR>);
printf("
4.PRINT THE ELEMENTS IN THE LIST :<BR>);
printf("
5.PRINT THE NUMBER OF ELEMENTS IN THE LIST <BR>);
printf("
6.DELETE A NODE IN THE LINKED LIST:<BR>);
printf("
7.REVERSE A LINKED LIST :<BR>);
```

```
printf("
8.GET OUT OF LINKED LIST (BYEE BYEE);
printf("
PLEASE, ENTER THE NUMBER:");

scanf("%d",&i); /* ENTER A VALUE FOR SWITCH */

switch(i)
{
case 1:
{
int num;
printf("PLEASE ENTER THE NUMBER :-");
scanf("%d",&num);
addbeg(num);
break;
}
case 2:
{
int num;
printf("
PLEASE ENTER THE NUMBER :-");
scanf("%d",&num);
append(num);
break;
}

case 3:
{
int num, loc,k;
printf("
PLEASE ENTER THE NUMBER :-");
scanf("%d",&num);
printf("
PLEASE ENTER THE LOCATION NUMBER :-");
scanf("%d",&loc);
addafter(num,loc);
break;
} case 4:
{
struct node *n;
printf("

THE ELEMENTS IN THE LIST ARE : <BR>);
display(n);
```

```
break;
}
case 5:
{
struct node *n;
display(n);
printf(" TOTAL NO OF ELEMENTS IN THE LSIT ARE %d",count());
break;
} case 6:
{
int num;
printf("PLEASE ENTER A NUMBER FROM THE LIST :");
scanf("%d",&num);
delnode(num);
break;
}
case 7:
{
reverse(p);
display(p);
break;
}
case 8:
{
exit();
}
}/* end if switch */
}/* end of while */
}/* end of main *
```

QUIZ

Q1. In a linked list with n nodes, the time taken to insert an element after an element pointed by some pointer is
(A) 0 (1) **(B)** 0 (log n)
(C) 0 (n) **(D)** 0 (n 1og n)
Ans:A
Q2. Consider a linked list of n elements. What is the time taken to insert an element after an element pointed by some pointer?
(A) O (1) (B) O (log2 n)
(C) O (n) (D) O (n log2 n)
Ans:A
Q3. Clarify whether Linked List is linear or Non-linear data structure ?
Answer:Link list is always linear data structure because every element (NODE) having unique position and also every element has its unique successor and predecessor. Also,linear collection of data items called nodes and the linear order is given by means of pointers. Each node is divided into two parts. First part contains information of the element and another part contains the address of the next node in the list.
Q4. Explain the types of linked list.
Ans: The types of linked lists are:
Singly linked list: It has only head part and corresponding references to the next nodes.
Doubly linked list: A linked list which both head and tail parts, thus allowing the traversal in bi-directional fashion. Except the first node, the head node refers to the previous node.
Circular linked list: A linked list whose last node has reference to the first node.
Q5. How would you sort a linked list?
Ans: Step 1: Compare the current node in the unsorted list with every element in the rest of the list. If the current element is more than any other element go to step 2 otherwise go to step 3.

Step 2: Position the element with higher value after the position of the current element. Compare the next element. Go to step1 if an element exists, else stop the process.

Step 3: If the list is already in sorted order, insert the current node at the end of the list. Compare the next element, if any and go to step 1 or quit.
Q6. In general, linked lists allow:
a. Insertions and removals anywhere.
b. Insertions and removals only at one end.
c. Insertions at the back and removals from the front.
d. None of the above.
ANS a. Insertions and removals anywhere.
Q7. What kind of linked list begins with a pointer to the first node, and each node contains a pointer to the next node, and the pointer in the last node points back to the first node?
a. Circular, singly-linked list.
b. Circular, doubly-linked list.
c. Singly-linked list.
d. Doubly-linked list.

ANS a. Circular, singly-linked list.

Q8. How many pointers are contained as data members in the nodes of a circular, doubly linked list of integers with five nodes?

Ans: 10

Q9. Which part in a linked list index represents the position of a node in a linked list?

Ans: An integer

Q10. What is the value of first linked list index?

Ans: Zero

EXPERIMENT NO.9

AIM: Write the program for implementation of a file and performing operations such as insert, delete, update a record in the file.

```
#include<stdio.h>
#include<conio.h>
void append();
void list();
void search();
void modify();
void del();
struct employee
{
int no, sal;
char gen, name[20];
};
void main()
{ int a;
char ch;
do{
printf("\nEMPLOYEE DATABASE\n\n");
printf("1.Append Employee Record\n2.List Employee Record\n3.Modify Employee
Record\n4.Delete Employee Record\n5.Search Employee Record\n Enter Choice : ");
scanf("%d",&amp;a);
switch(a)
{
case 1:
append();
break;
case 2:
list();
break;
case 3:
modify();
break;
case 4:
del();
```

```
break;
case 5:
search();
break;
default :
printf("Invalid Choice!");
}
printf("\n More Actions ? (Y/N) :");
fflush(stdin);
scanf("%c", &amp;ch);
}while(ch=='y'|| ch=='Y');
}
void append()
{ int i,n;
struct employee e;
FILE *fp;
fp=fopen("Employee.dat", "a");
if(fp==NULL)
{
printf("File Creation Failed!");
exit(0);
}
printf("Enter the nos. of employees : ");
scanf("%d", &amp;n);
for(i=0;i<n;i++)
 {
printf("Enter the Employee Number : ");
scanf("%d", &amp;e.no);
printf("Enter the Employee Salary : ");
scanf("%d", &amp;e.sal);
printf("Enter the Employee gender: ");
fflush(stdin);
scanf("%c", &amp;e.gen);
printf("Enter the Employee Name : ");
fflush(stdin);
gets(e.name);
printf("\n\n");</n;i++)
```

```
fwrite((char *)&amp;amp;e, sizeof(e), 1, fp);
}
fclose(fp);
}
void list()
{ int nofrec=0;
struct employee e;
FILE *fp;
fp=fopen("Employee.dat", "rb");
if(fp==NULL)
{
printf("\n\tFile doesn't exist!!!\TRY AGAIN");
exit(0);
}
while((fread((char *)&amp;amp;e, sizeof(e), 1, fp))==1)
{ nofrec++;
printf("\nEmployee Number : %d", e.no);
printf("\nEmployee Salary : %d", e.sal);
printf("\nEmployee gender : %c",e.gen);
printf("\nEmployee Name : %s",e.name);
printf("\n\n");
}
printf("Total number of records present are : %d", nofrec);
fclose(fp);}

void modify()
{ int recno, nofrec=0;
char ch;

struct employee e;
FILE *fp;
fp=fopen("Employee.dat", "rb+");
printf("Enter the Employee Number to modify : ");
scanf("%d", &amp;amp;recno);
while((fread((char *)&amp;amp;e, sizeof(e), 1, fp))==1)
{ nofrec++;
if(e.no==recno)
```

```
{
printf("\nEmployee Number : %d", e.no);
printf("\nEmployee Salary : %d", e.sal);
printf("\nEmployee gender : %c",e.gen);
printf("\nEmployee Name : %s",e.name);
printf("\n");
printf("Do you want to modify this record : ? (Y/N)");
fflush(stdin);
scanf("%c", &amp;ch);
fseek(fp, ((nofrec-1)*sizeof(e)), 0);
if(ch=='Y'|| ch=='y')
{
printf("Enter the Employee Salary : ");
scanf("%d", &amp;e.sal);
printf("Enter the Employee gender: ");
fflush(stdin);
scanf("%c", &amp;e.gen);
printf("Enter the Employee Name : ");
fflush(stdin);
gets(e.name);
fwrite((char *)&amp;e, sizeof(e), 1, fp);
printf("Record Modified");
}
else
printf("No modifications were made");
fclose(fp);

}
}
}
void del()
{
int recno;
char ch;
struct employee e;
FILE *fp, *ft;
```

```
fp=fopen("Employee.dat", "rb");
ft=fopen("Temp.dat", "wb");
printf("Enter the Employee Number to delete : ");
scanf("%d", &amp;recno);
while((fread((char *)&amp;e, sizeof(e), 1, fp))==1)
{
if(e.no==recno)
{
printf("\nEmployee Number : %d", e.no);
printf("\nEmployee Salary : %d", e.sal);
printf("\nEmployee gender : %c",e.gen);
printf("\nEmployee Name : %s",e.name);
printf("\n");
printf("Do you want to delete this record : ? (Y/N)");
fflush(stdin);
scanf("%c", &amp;ch);
}
}
if(ch=='y'||ch=='Y')
{
rewind(fp);
while((fread((char *)&amp;e, sizeof(e), 1, fp))==1)
{
if(recno!=e.no)
{
fwrite((char *)&amp;e, sizeof(e), 1, ft);
}
}
}
else
printf("No Record was deleted");
fclose(fp);
fclose(ft);
remove("Employee.dat");
rename("Temp.dat", "Employee.dat");
}
void search()
```

```
{ int s,recno;
char sname[20];
struct employee e;
FILE *fp;
fp=fopen("Employee.dat", "rb");
printf("\n1.Search by Name\n2.Search by Employee No.\n Enter choice : ");
scanf("%d", &amp;amp;s);
switch(s)
{
case 1:
printf("Enter the Employee Name to Search : ");
fflush(stdin);
gets(sname);
while((fread((char *)&amp;amp;e, sizeof(e), 1, fp))==1)
{
if(strcmp(sname,e.name)==0)
{
printf("\nEmployee Number : %d", e.no);
printf("\nEmployee Salary : %d", e.sal);
printf("\nEmployee gender : %c",e.gen);
printf("\nEmployee Name : %s",e.name);
printf("\n");
}
}
break;
case 2:printf("Enter the Employee Number to Search : ");
scanf("%d", &amp;amp;recno);
while((fread((char *)&amp;amp;e, sizeof(e), 1, fp))==1)
{
if(e.no==recno)
{
printf("\nEmployee Number : %d", e.no);
printf("\nEmployee Salary : %d", e.sal);
printf("\nEmployee gender : %c",e.gen);
printf("\nEmployee Name : %s",e.name);
printf("\n");
```

```
}
}
break;
}
}
```

QUIZ

Q1. **What is File Handling?**

We frequently use files for storing information which can be processed by our programs. In order to store information permanently and retrieve it we need to use files .A file is collection of related data .Placed on the disk.Files are not only used for data. Our programs are also stored in files.

Q2. **Why we use file Handling:**

The input and out operation that we have performed so far were done through screen and keyboard only. After the termination of program all the entered data is lost because primary memory is volatile . If the data has to be used later , then it becomes necessary to keep it in permanent storage device. So the c language provide the concept of file through which data can be stored on the disk or secondary storage device. The stored data can be read whenever required.

Q3. **Explain Types of File Handling in C:**

The file handling in c can be categorized in two types-

1. **High level (Standard files or stream oriented files)-** High level file handling is managed by library function. High level file handling commonly used because it is easier and hide most of the details from the programmer.
2. **Low level (system oriented files)-** low level files handling is managed by system calls.

Q4. A random access file is organized most like a(n):

a. Array.
b. Object.
c. Class.
d. Pointer.

ANS: a. Array.

Q5. What are the typical operations on file?

Ans: Open, Read, Write and close

Q6. How is a file stored?

Ans: Stored as sequence of bytes, logically contiguous (may not be physically contiguous on disk).

Q7. In C, what we use to set a pointer to a file?

Ans: File *

Q8. Which command is used to open a file?

Ans: fopen() and it returns null if unable to open a file.

Q9. What are the modes used in fopen()?

Ans: read, write and append

Q10. What is fprintf() function?

Ans: It works like printf() and sprint() except that its first argument is a file pointer.

EXPERIMENT NO.10

AIM: Create a linked list and perform the following operations on it
a) add a node b) Delete a node

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

struct node
{
   int data;
   struct node *next;
}*head;

void append(int num)
{
   struct node *temp,*right;
   temp= (struct node *)malloc(sizeof(struct node));
   temp->data=num;
   right=(struct node *)head;
   while(right->next != NULL)
   right=right->next;
   right->next =temp;
   right=temp;
   right->next=NULL;
}

 void add( int num )
{
   struct node *temp;
   temp=(struct node *)malloc(sizeof(struct node));
   temp->data=num;
   if (head== NULL)
   {
   head=temp;
   head->next=NULL;
   }
   else
   {
   temp->next=head;
   head=temp;
   }
}
```

```
void addafter(int num, int loc)
{
   int i;
   struct node *temp,*left,*right;
   right=head;
   for(i=1;i<loc;i++)
   {
   left=right;
   right=right->next;
   }
   temp=(struct node *)malloc(sizeof(struct node));
   temp->data=num;
   left->next=temp;
   left=temp;
   left->next=right;
   return;
}

 void insert(int num)
{
   int c=0;
   struct node *temp;
   temp=head;
   if(temp==NULL)
   {
   add(num);
   }
   else
   {
   while(temp!=NULL)
   {
      if(temp->data<num)
      c++;
      temp=temp->next;
   }
   if(c==0)
      add(num);
   else if(c<count())
      addafter(num,++c);
   else
      append(num);
   }
}
```

```
int delete(int num)
{
   struct node *temp, *prev;
   temp=head;
   while(temp!=NULL)
   {
   if(temp->data==num)
   {
      if(temp==head)
      {
      head=temp->next;
      free(temp);
      return 1;
      }
      else
      {
      prev->next=temp->next;
      free(temp);
      return 1;
      }
   }
   else
   {
      prev=temp;
      temp= temp->next;
   }
   }
   return 0;
}

 void  display(struct node *r)
{
   r=head;
   if(r==NULL)
   {
   return;
   }
   while(r!=NULL)
   {
   printf("%d ",r->data);
   r=r->next;
   }
   printf("\n");
}
```

```
int count()
{
   struct node *n;
   int c=0;
   n=head;
   while(n!=NULL)
   {
   n=n->next;
   c++;
   }
   return c;
}

int  main()
{
   int i,num;
   struct node *n;
   head=NULL;
   while(1)
   {
   printf("\nList Operations\n");
   printf("===============\n");
   printf("1.Insert\n");
   printf("2.Display\n");
   printf("3.Size\n");
   printf("4.Delete\n");
   printf("5.Exit\n");
   printf("Enter your choice : ");
   if(scanf("%d",&i)<=0){
      printf("Enter only an Integer\n");
      exit(0);
   } else {
      switch(i)
      {
      case 1:      printf("Enter the number to insert : ");
               scanf("%d",&num);
               insert(num);
               break;
      case 2:     if(head==NULL)
               {
               printf("List is Empty\n");
               }
               else
               {
               printf("Element(s) in the list are : ");
```

```
            }
            display(n);
            break;
        case 3:     printf("Size of the list is %d\n",count());
            break;
        case 4:     if(head==NULL)
            printf("List is Empty\n");
            else{
            printf("Enter the number to delete : ");
            scanf("%d",&num);
            if(delete(num))
               printf("%d deleted successfully\n",num);
            else
               printf("%d not found in the list\n",num);
            }
            break;
        case 5:     return 0;
        default:    printf("Invalid option\n");
        }
    }
    }
    return 0;
}
```

QUIZ

Q1. Consider a linked list of n elements. What is the time taken to insert an element after an element pointed by some pointer?
(A) O (1) (B) O (log2 n)
(C) O (n) (D) O (n log2 n)
Ans:A
Q2. In a linked list with n nodes, the time taken to insert an element after an element pointed by some pointer is
(A) 0 (1) **(B)** 0 (log n)
(C) 0 (n) **(D)** 0 (n 1og n)
Ans:A
Q3. Why linked list implementation of sparse matrices is superior to the generalized dope vector?
Ans: Because linked list implementation of sparse matrices is conceptually easier and completely dynamic.
Q4. Which list implementation could be used for concatenation of two lists in O(1) time?
Ans: Circular doubly linked list
Q5. Linked lists are not suitable for implementing:
(A) insertion sort (B) Binary search
(C) radix sort **(D)** polynomial manipulation
Ans: B
Q6. How many pointers are contained as data members in the nodes of a circular, doubly linked list of integers with five nodes?
Ans: 10
Q7. Which part in a linked list index represents the position of a node in a linked list?
Ans: An integer
Q8. What is the value of first linked list index?
Ans: Zero
Q9. Write an application of linked list in data structure.
Ans: Memory management
Q10. What are the uses of linked list?
Ans: They can be used to implement several other common abstract data types, including stacks, queues, associative arrays, and symbolic expressions

EXPERIMENT NO.11

AIM: Write a program to simulate the various searching & sorting algorithms and compare their timings for a list of 1000 elements.

SELECTION SORT

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,j,small,pos,temp;
int size,i;
int l[10];
printf("\n enter d size of list");
scanf("%d",&size);
printf("\n enter the elements");
for(i=0;i<size;i++)
{
scanf("%d",&l[i]);
}
for(i=0;i<size-1;i++)
{
small=l[i];
pos=i;
for(j=i+1;j<size;j++)
{
if(small>l[j])
{
small=l[j];
pos=j;
}
}
temp=l[i];
l[i]=l[pos];
l[pos]=temp;
}
printf("\n sorted list...");
for(i=0;i<size;i++)
{
printf("%d",l[i]);
}
getch();
}
```

BUBBLE SORT

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,j,small,pos,temp;
int size,i;
int l[10];
printf("\n enter d size of list");
scanf("%d",&size);
printf("\n enter the elements");
for(i=0;i<size;i++)
{
scanf("%d",&l[i]);
}
int f=0;
int c=0;
while(f==0)
{
f=1;
c++;
for(j=0;j<size-1;j++)
{
if(l[j]>l[j+1]}
{
temp=l[j];
l[j]=l[j+1];
l[j+1]=temp;
f=0;
}
}
}
printf("sorted list is.....");
for(i=0;i<size;i++)
{
printf("%d",l[i]);
printf("no. of passes made are %d",c);
getch();
}
for(i=0;i<n-1;i++)
{
small=l[i];
pos=i;
for(j=i+1;j<n;j++)
```

```
{
if(small>l[j])
{
small=l[j];
pos=j;
}
temp=l[i];
l[i]=l[pos];
l[pos]=temp;
}
}
printf("\n sorted list...");
for(i=0;i<n;i++)
{
printf("%d",l[i]);
}
getch();
}
```

INSERTION SORT

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,j,small,pos,temp;
int size,i;
int l[10];
printf("\n enter d size of list");
scanf("%d",&size);
printf("\n enter the elements");
for(i=0;i<size;i++)
{
scanf("%d",&l[i]);
}
for(i=1;i<size;i++)
{
temp=l[i];
j=i-1;
while((temp<l[j]) && (j>=0))
{
l[j+1]=l[j];
j=j-1;
}
l[j+1]=temp;
```

```
}
printf("sorted list is.....");
for(i=0;i<size;i++)
{
printf("%d",l[i]);
}
//printf("no. of passes made are %d",c);
getch();
}
```

SHELL SORT

```
#include<stdio.h>
#include<conio.h>
void main()
{
int list[20];
int size,i,j,k,p;
int temp,s,step;
int dimstep[]={5,3,1};
clrscr();
printf("\n enter size of list");
scanf("%d",&size);
printf("\n enter elements");
for (i=0;i<size;i++)
{
scanf("%d",&list[i]);
}
for (step=0;step<3;step++)
{
k=dimstep[step];
s=0;
for(i=s+k;i<size;i=i+k)
{
temp=list[i];
j=i-k;
while((temp<list[j])&&(j>=0))
{
list[j+k]=list[j];
j=j-k;
}
list[j+k]=temp;
s++;
}
```

```
}
printf("\n sorted list is..");
for(i=0;i<size;i++)
{
printf("%d",list[i]);
}
getch();
}
```

QUIZ

Q1. You have to sort a list L consisting of a sorted list followed by a few "random" elements. Which sorting method would be especially suitable for such a task?
Ans: Insertion Sort
Q2. A sort which relatively passes through a list to exchange the first element with any element less than it and then repeats with a new first element is called
(A) insertion sort. **(B)** selection sort.
(C) heap sort. **(D)** quick sort.
Ans:D
Q3. Which of the following sorting algorithms does not have a worst case running time of (2) O n
(A) Insertion sort (B) Merge sort
(C) Quick sort (D) Bubble sort
Ans:B
Q4. Which sorting algorithm is best if the list is already sorted? Why?
Ans. Insertion sort as there is no movement of data if the list is already sorted and complexity is of the order O(N).
Q5. **What are the various kinds of sorting techniques? Which is the best case?**
Ans: heap sort is the best sorting technique bcoz its complexity in best case ,worst and avg case is of O(nlogn).In worst case quick sort the complexity in best case and avg case is of O(nlogn) and worst case O(n^2).
Q6. Sorting is useful for:
(A) report generation (B) minimizing the storage needed
(C) making searching easier and efficient (D) responding to queries easily
Ans: D
Q7. Which sorting method is best if the number of swapping done is the only measure of efficiency?
Ans: Selection sort
Q8. Which sorting method is preferred for sorting 15 randomly generated numbers?
Ans: Bubble sort
Q9. What are the maximum numbers of comparisons needed to sort 7 items using radix sort?
Ans: 280
Q10. Which sorting algorithm has the worst time complexity of nlog(n) ?
Ans: Heap sort

EXPERIMENT NO.12

AIM: Write a program to simulate the various graph traversing algorithms.

```
#include<stdio.h>
#define MAX 20

typedef enum boolean{false,true} bool;
int adj[MAX][MAX];
bool visited[MAX];
int n; /* Denotes number of nodes in the graph */
main()
{
int i,v,choice;

create_graph();
while(1)
{
printf("\n");
printf("1. Adjacency matrix\n");
printf("2. Depth First Search using stack\n");
printf("3. Depth First Search through recursion\n");
printf("4. Breadth First Search\n");
printf("5. Adjacent vertices\n");
printf("6. Components\n");
printf("7. Exit\n");
printf("Enter your choice : ");
scanf("%d",&choice);

switch(choice)
{
case 1:
printf("Adjacency Matrix\n");
display();
break;
case 2:
printf("Enter starting node for Depth First Search : ");
scanf("%d",&v);
for(i=1;i<=n;i++)
visited[i]=false;
dfs(v);
break;
case 3:
printf("Enter starting node for Depth First Search : ");
```

```
scanf("%d",&v);
for(i=1;i<=n;i++)
visited[i]=false;
dfs_rec(v);
break;
case 4:
printf("Enter starting node for Breadth First Search : ");
scanf("%d", &v);
for(i=1;i<=n;i++)
visited[i]=false;
bfs(v);
break;
case 5:
printf("Enter node to find adjacent vertices : ");
scanf("%d", &v);
printf("Adjacent Vertices are : ");
adj_nodes(v);
break;
case 6:
components();
break;
case 7:
exit(1);
default:
printf("Wrong choice\n");
break;
}/*End of switch*/
}/*End of while*/
}/*End of main()*/

create_graph()
{
int i,max_edges,origin,destin;

printf("Enter number of nodes : ");
scanf("%d",&n);
max_edges=n*(n-1);

for(i=1;i<=max_edges;i++)
{
printf("Enter edge %d( 0 0 to quit ) : ",i);
scanf("%d %d",&origin,&destin);

if((origin==0) && (destin==0))
break;
```

```
if( origin > n || destin > n || origin<=0 || destin<=0)
{
printf("Invalid edge!\n");
i--;
}
else
{
adj[origin][destin]=1;
}
}/*End of for*/
}/*End of create_graph()*/

display()
{
int i,j;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
printf("%4d",adj[i][j]);
printf("\n");
}
}/*End of display()*/

dfs_rec(int v)
{
int i;
visited[v]=true;
printf("%d ",v);
for(i=1;i<=n;i++)
if(adj[v][i]==1 && visited[i]==false)
dfs_rec(i);
}/*End of dfs_rec()*/

dfs(int v)
{
int i,stack[MAX],top=-1,pop_v,j,t;
int ch;

top++;
stack[top]=v;
while (top>=0)
{
pop_v=stack[top];
top--; /*pop from stack*/
```

```
if( visited[pop_v]==false)
{
printf("%d ",pop_v);
visited[pop_v]=true;
}
else
continue;
for(i=n;i>=1;i--)
{
if( adj[pop_v][i]==1 && visited[i]==false)
{
top++; /* push all unvisited neighbours of pop_v */
stack[top]=i;
}/*End of if*/
}/*End of for*/
}/*End of while*/
}/*End of dfs()*/

bfs(int v)
{
int i,front,rear;
int que[20];
front=rear= -1;

printf("%d ",v);
visited[v]=true;
rear++;
front++;
que[rear]=v;

while(front<=rear)
{
v=que[front]; /* delete from queue */
front++;
for(i=1;i<=n;i++)
{
/* Check for adjacent unvisited nodes */
if( adj[v][i]==1 && visited[i]==false)
{
printf("%d ",i);
visited[i]=true;
rear++;
que[rear]=i;
}
}
```

```
}/*End of while*/
}/*End of bfs()*/

adj_nodes(int v)
{
int i;
for(i=1;i<=n;i++)
if(adj[v][i]==1)
printf("%d ",i);
printf("\n");
}/*End of adj_nodes()*/
components()
{
int i;
for(i=1;i<=n;i++)
visited[i]=false;
for(i=1;i<=n;i++)
{
if(visited[i]==false)
dfs_rec(i);
}
printf("\n");
}
```

QUIZ

Q. Let A be an adjacency matrix of a graph G. The th ij entry in the matrix K A , gives
(A) The number of paths of length K from vertex Vi to vertex Vj.
(B) Shortest path of K edges from vertex Vi to vertex Vj.
(C) Length of a Eulerian path from vertex Vi to vertex Vj.
(D) Length of a Hamiltonian cycle from vertex Vi to vertex Vj.
Ans:B
Q2. For an undirected graph with n vertices and e edges, what would be the sum of the degree of each vertex?
Ans:2e
Q3. An undirected graph G with n vertices and e edges is represented by adjacency list. What is the time required to generate all the connected components?
(A) O (n) (B) O (e)
(C) O (e+n) (D) O (2) e
Ans:C
Q4. A graph with n vertices will definitely have a parallel edge or self loop of the total number of edges are
(A) more than n (B) more than n+1
(C) more than (n+1)/2 (D) more than n (n-1)/2
Ans: D
Q5.Which data structure is required for Breadth First Traversal on a graph is
Ans: Queue
Q6. An adjacency matrix representation of a graph cannot contain information of:
(A) nodes **(B)** edges
(C) direction of edges **(D)** parallel edges
Ans:D
Q7. :Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes ?
Answer:Minimal spanning tree assures that the total weight of the tree is kept at its minimum but it doesn't mean that the distance between any two nodes involved in the minimum-spanning tree is minimum.
Q8. What is the maximum degree of any vertex in a simple graph with n vertices?
Ans: n-1
Q9. Which technique is useful in traversing a graph by breadth first search?
Ans: Queue
Q10. What is the minimum number of edges in a connected cyclic graph on n vertices?
Ans: n

EXPERIMENT NO.13

AIM: Write a program which simulates the various tree traversal algorithms.

```
#include<stdio.h>
#include<conio.h>
struct node
{
int data;
struct node *right, *left;
}*root,*p,*q;

struct node *make(int y)
{
struct node *newnode;
newnode=(struct node *)malloc(sizeof(struct node));
newnode->data=y;
newnode->right=newnode->left=NULL;
return(newnode);
}

void left(struct node *r,int x)
{
if(r->left!=NULL)
printf("\n Invalid !");
else
r->left=make(x);
}

void right(struct node *r,int x)
{
if(r->right!=NULL)
printf("\n Invalid !");
else
r->right=make(x);
}

void inorder(struct node *r)
{
if(r!=NULL)
```

```
{
inorder(r->left);
printf("\t %d",r->data);
inorder(r->right);
}
}

void preorder(struct node *r)
{
if(r!=NULL)
{
printf("\t %d",r->data);
preorder(r->left);
preorder(r->right);
}
}

void postorder(struct node *r)
{
if(r!=NULL)
{
postorder(r->left);
postorder(r->right);
printf("\t %d",r->data);
}
}

void main()
{
int no;
int choice;
clrscr();
printf("\n Enter the root:");
scanf("%d",& no);
root=make(no);
p=root;
while(1)
{
printf("\n Enter another number:");
```

```
scanf("%d",& no);
if(no==-1)
break;
p=root;
q=root;
while(no!=p->data && q!=NULL)
{
p=q;
if(nodata)
q=p->left;
else
q=p->right;
}
if(nodata)
{
printf("\n Left branch of %d is %d",p->data,no);
left(p,no);
}
else
{
right(p,no);
printf("\n Right Branch of %d is %d",p->data,no);
}
}
while(1)
{
printf("\n 1.Inorder Traversal \n 2.Preorder Traversal \n 3.Postorder Traversal \n 4.Exit");
printf("\n Enter choice:");
scanf("%d",&choice);
switch(choice)
{
case 1 :inorder(root);
break;
case 2 :preorder(root);
break;
case 3 :postorder(root);
break;
```

```
case 4 :exit(0);
default:printf("Error ! Invalid Choice ");
break;
}
getch();
}
```

QUIZ

Q1. If a node having two children is deleted from a binary tree, it is replaced by which type of predecessor/successor?
Ans:Inorder Successor
Q2. How many leaf nodes are there in a full binary tree with 2n+1 nodes?
Ans: n non-leaf nodes
Q3. If a node in a BST has two children, then its inorder predecessor has
(A) no left child (B) no right child
(C) two children (D) no child
Ans:B
Q4. What we call to a binary tree in which if all its levels except possibly the last, have the maximum number of nodes and all the nodes at the last level appear as far left as possible.
Ans:Full Binary Tree
Q5. How many nodes are there in a full binary tree with n leaves?
Ans: 2n –1 nodes
Q6. A BST is traversed in the following order recursively: Right, root, left
The output sequence will be in
(A) Ascending order (B) Descending order
(C) Bitomic sequence (D) No specific order
Ans:B
Q7. The pre-order and post order traversal of a Binary Tree generates the same output. The tree can have maximum
(A) Three nodes (B) Two nodes
(C) One node (D) Any number of nodes
Ans:C
Q8. What is the maximum possible number of nodes in a binary tree at level 6?
Ans. $2^6 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 64$
Q9. How to find the number of possible tree in the given tree.
Ans: The number of possible tree = (2 power of n) - n.
For example: A tree contain three node. So if n=3,the possible number of trees = 8 - 3 = 5.
Q10. What is almost complete binary tree?
Ans: An almost complete binary tree is a tree in which each nodethat has a right child also has a left child. Having a left child does not require a node to have a right child. Stated alternately, an almost complete binary tree is a tree where for a right child, there is always a left child, but for a left child there may not be a right child. The number of nodes in a binary tree can be found using this formula: n = 2^h Where n is the amount of nodes in the tree, and h is the height of the tree.