Data Structures
Practical File

Name : Balasara Krishna Arvindbhai

Enrollment no. : 210470107008

Batch : G1

Roll No. : 21ce151

V.V.P. ENGINEERING COLLEGE

COMPUTER ENGINEERING DEPARTMENT

V.V.P. Engineering College – Rajkot

Department of Computer Engineering

Subject: Data Structures (3130702) ODD 2022-2023

Name of Faculty:Miss Brinda D. Chanv **Sem/Div. : 3G(ODD)**

Subject : Data Structure(3130702) **Batch:G1/G2/G3/G4**

Sr. No.	Experiment	CO	CL
1	1) Write a c program to implement function Swap using two different parameter passing mechanism. 2) Write a c program to store 5 values in appropriate data structure and compute addition for the same, modify the size to store 10 values and compute addition. 3) Write a c program to get record (Player name, team name & runs of innings) of any three players from Indian cricket team. Print the record according to name of players in ascending order. 4) Write a c program to create calculator (use user defined function named Calculator).	Pre requisite	U
2	1) Write a C Program to perform addition on 2 complex number using structure. (Use user defined function addition). 2) Write a C Program to implement searching on unordered array of 10 integer value. 3) Write a C Program to find largest value from array of 10 integers. 4) Write a C Program to Swap Max and Min Value from array of 10 integer value. 5) Write a C Program to insert a value in array of 10 integers at specific position. 6) Write a C Program to delete a value in array of 10 integers from specific position.	CO2	U
3	1) Write a C Program to implement searching on ordered array of 10 integer value. 2) Write a C Program to implement Stack with all necessary overflow and underflow condition (Use array as data structure). 1)PUSH 2) POP 3) DISPLAY. 3) Write a C Program to convert infix notation into its equivalent postfix notation using stack. 4) Write a c Program to implement Tower of Hanoi problem. 5) Write a C Program to implement Infix to prefix conversion using stack.	CO1, CO2	An

V.V.P. Engineering College – Rajkot

Department of Computer Engineering

Subject: Data Structures (3130702) ODD 2022-2023

Name of Faculty: Miss Brinda D. Chanv **Sem/Div. : 3/G(ODD)**

Subject : Data Structure(3130702) **Batch:G2/G3/G4**

Sr. No.	Experiment	CO	CL
4	1) Write a program to implement Queue Sort. 2) Write a program to implement Merge Sort. 3) Write a program to implement Bubble Sort. 4) Write a program to implement selection sort. 5) Write a program to implement Binary Search.	CO1, CO4	U
5	1) Write a C program to implement simple queue (Insertion, deletion and traversal). 2) Write a C program to implement circular queue (Insertion, deletion and traversal). 3) Write a C program to implement priority queue (Insertion , deletion and traversal).	CO2	U
6	1) Write a c program to implement singly linked list for the following function. (a) Insert a node at the front of the linked list. (b) Insert a node at the end of the linked list. (c) Insert a node in order. (d) Delete any node from the linked list. (e) Count total no of nodes in list. (f) Insert a node at any position in linked list. (g) Display all nodes (traversal of Linked list). 2) Write a program to implement following operations on the Doubly linked list. (a) Insert a node at the front of the linked list. (b) Insert a node at the end of the linked list. (c) Insert a node at any position in linked list. (d) Delete any node from the linked list. (e) Count total no of nodes in list. (f) Insert a node at any position in linked list. (g) Display all nodes (traversal of Linked list).	CO2	U

V.V.P. Engineering College – Rajkot

Department of Computer Engineering

Subject: Data Structures (3130702) ODD 2022-2023

Name of Faculty: Miss Brinda D. Chanv **Sem/Div. : 3/G(ODD)**

Subject : Data Structure(3130702) **Batch:G2/G3/G4**

Sr. No.	Experiment	CO	CL
7	1) Write a program to implement following operations on the Circular linked list. (a) Insert a node at the front of the linked list. (b) Insert a node at the end of the linked list. (c) Insert a node in order. (d) Delete any node from the linked list. (e) Count total no of nodes in list. (f) Display all nodes (traversal of Linked list).	CO2	U
8	1) Write a c program to implement stack using Linked list. (PUSH, POP, DISPLAY). 2) Write a c program to implement queue using Linked List. (enqueue, dequeue, display). 3) Write a c program to implement Double-ended queue using Linked List. (enqueue, dequeue , display). 4) Write a c program to swap max and min element from singly linked list without swapping address. 5) Write a c program to remove duplicate values from singly Linked list.	CO2	An
9	1) Write a C program to create a Binary Search Tree (Insertion, deletion and traversal). 2) Write a C program to implement tree traversing methods inorder, preorder and post-order traversal.	CO3	U
10	1) Write a c program to implement DFS graph traversal. 2) Write a c program to implement BFS graph traversal.	CO3	U

PRACTICAL 1

1) Write a c program to implement function Swap using two different parameter passing mechanism.

```
#include <stdio.h>
void swap(int a ,int b)
{
   int temp;
   temp=a;
   a=b;
   b=temp;
   printf("After swapping a=%d, b=%d" ,a ,b);
}
int main()
{
   int a=2,b=3;
   printf("Before swapping a=%d, b=%d\n" ,a, b);
   swap(a,b);
}
```

OUTPUT

Before swapping a=2, b=3

After swapping a=3, b=2

2) Write a c program to store 5 values in appropriate data structure and compute addition for the same, modify the size to store 10 values and compute addition.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int* ptr;
   int n1, n2, i;
   printf("Enter size: ");
   scanf("%d", &n1);
   printf("Enter number of elements: %d\n", n1);
   // Dynamically allocate memory using malloc()
   ptr = (int*) malloc(n1 * sizeof(int));
   // Check if the memory has been successfully
   // allocated by malloc or not
   if (ptr == NULL) {
      printf("Memory not allocated.\n");
      exit(0);
   }
   else {
      printf("Memory successfully allocated using calloc.\n");
      for (i = 0; i < n1; ++i) {
         ptr[i] = i + 1;
      }
      printf("The elements of the array are: ");
      for (i = 0; i < n1; ++i) {
         printf("%d, ", ptr[i]);
      }
```

```
        printf("\nEnter the new size: ");
        scanf("%d", &n2);
        ptr = realloc(ptr, n1 * sizeof(int));
        printf("Memory successfully re-allocated using realloc.\n");
        // Get the new elements of the array
        for (i = 5; i < n2; ++i) {
            ptr[i] = i + 1;
        }
        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n2; ++i) {
            printf("%d, ", ptr[i]);
        }
        free(ptr);
    }
return 0;
}
```

OUTPUT

Enter size: 5

Enter number of elements: 5

Memory successfully allocated using calloc.

The elements of the array are: 1, 2, 3, 4, 5,

Enter the new size: 10

Memory successfully re-allocated using realloc.

The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

3) Write a c program to get record (Player name, team name & runs of innings) of any three players from Indian cricket team. Print the record according to name of players in ascending order.

```
#include<stdio.h>
#include<string.h>
struct crickter{
   char name[10];
   char team[10];
   int runs;
}player[100],temp;
int main()  {
   printf("Enter info of 3 players as follow name, team and runs\n");
   for(int i=0; i<3; i++){
      scanf("%s %s %d",player[i].name, player[i].team, &player[i].runs);
   }
   for(int i=0; i<2; i++){
      for(int j=i+1; j<3; j++) {
         if(strcmp(player[i].name, player[j].name)>0) {
            temp=player[i];
            player[i]=player[j];
            player[j]=temp;
         }
      }
   }
   printf("Record of players in ascending  order\n");
   printf("-----------------------------------------\n");
   printf("Name\t\t Team name\t\t Runs\n");
   printf("-----------------------------------------\n");
```

```
    for(int i=0; i<3; i++){
        printf("%s\t\t %s\t\t\t%d\n", player[i].name ,player[i].team, player[i].runs);
    }
}
```

OUTPUT

Enter info of 3 players as follow name, team and runs

Virat India 200

Rohit India 150

Rahul India 100

Record of players in ascending order

Name Team name Runs

Rahul India 100

Rohit India 150

Virat India 200

4) Write a c program to create calculator (use user defined function named Calculator).

```
#include <stdio.h>
int calculator(double val1, double val2, char ope)
   {
   if(ope == '+') {
   printf("Addition of two numbers is %lf ", val1 + val2);
   }
   else if(ope == '-') {
   printf("Subtraction of two numbers is %lf ", val1 - val2);
   }
   else if(ope == '*')  {
   printf("Multiplication of two numbers is %lf ", val1 * val2);
   }
   else if(ope == '/')  {
   printf("Division of two numbers is %lf ", val1 / val2);
   }  else {
   printf("Invalid operator");
   }
}
void main()
   {
   double val1 , val2;
   char ope;
   printf("Enter First Number ");
   scanf("%lf", &val1);
   printf("Enter Second Number ");
   scanf("%lf", &val2);
   printf("Enter '+' for Add\n '-' for Sub\n '*' for Mul\n '/' for Div\n");
```

```
    scanf(" %c", &ope);
    calculator(val1,val2,ope);
}
```

OUTPUT

Enter First Number 5

Enter Second Number 7

Enter '+' for Add

'-' for Sub

'*' for Mul

'/' for Div

+

Addition of two numbers is 12.000000

Question 1: What do you mean by scope of variables? What is the scope of variable sin C?

In simple terms, scope of a variable is its lifetime in the program. This means that the scope of a variable is the block of code in the entire program where the variable is declared, used, and can be modified. The following example declares the variable x on line 1, which is different from the x it declares on line 2. The declared variable on line 2 has function prototype scope and is visible only up to the closing parenthesis of the prototype declaration. A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language − Inside a function or a block which is called local variables.

Question 2: How can we store negative integer in C?

The C standard doesn't mandate any particular way of representing negative signed numbers. In most implementations that you are likely to encounter, negative signed integers are stored in what is called two's complement. The other major way of storing negative signed numbers is called one's complement. The two's complement of an N-bit number x is defined as 2^N - x. For example, the two's complement of 8-bit 1 is 2^8 - 1, or 1111 1111. The two's complement of 8-bit 8 is 2^8 - 8, which in binary is 1111 1000. This can also be calculated by flipping the bits of x and adding one.

For example:

```
 1        = 0000 0001
~1        = 1111 1110 (1's complement)
~1 + 1    = 1111 1111 (2's complement)
-1        = 1111 1111
```

Question 3: Differentiate between actual parameters and formal parameters.

Parameter

- A parameter is a special kind of variable, used in a function to refer to one of the pieces of data provided as input to the function to utilise.
- These pieces of data are called arguments.
- Parameters are Simply Variables.

Formal Parameter

- Parameter Written in Function Definition is Called “Formal Parameter.
- Formal parameters are always variables, while actual parameters do not have to be variables.

Actual Parameter

- Parameter Written in Function Call is Called “Actual Parameter”.
- One can use numbers, expressions, or even function calls as actual parameters.

Example

```
void display(int para1)
{
printf( “ Number %d “ , para1);
}
 void main()
 {
 int num1; display(num1);
 }
```

In above,
para1 is called the Formal Parameter
num1 is called the Actual Parameter.

PRACTICAL 2

1) Write a C Program to perform addition on 2 complex number using structure. (Use user defined function addition).

```
#include <stdio.h>
typedef struct complex {
   float real;
   float imag;
} complex;
complex addition(complex n1, complex n2);
int main() {
   complex n1, n2, result;
   printf("Enter the real and imaginary parts of two numbers: \n");
   scanf("%f %f", &n1.real, &n2.imag);
   scanf("%f %f", &n2.real, &n2.imag);
   result = addition(n1, n2);
   printf("---------------------------------\n");
   printf("Sum = %.1f + %.1fi", result.real, result.imag);  }
complex addition(complex n1, complex n2) {
   complex temp;
   temp.real = n1.real + n2.real;
   temp.imag = n1.imag + n2.imag;
   return (temp);
}
```

OUTPUT

Enter the real and imaginary parts of two numbers:

5 2

7 3

Sum = 12.0 + 5.0i

2) Write a C Program to implement searching on unordered array of 10 integer value.

```
#include<stdio.h>
int main() {
   int arr[10], x, count=0;
   printf("Enter any 10 numbers: \n");
   for(int i=0; i<10; i++) {
      scanf("%d",&arr[i]);
   }
   printf("Enter number that you want to search: \n");
   scanf("%d", &x);
   for(int i=0; i<10; i++)  {
      if(arr[i]==x)  {
         printf("-------------------------\n");
         printf("The number is at index: %d\n", i);
      }
   }
}
```

OUTPUT

Enter any 10 numbers:

2 4 6 8 1 3 5 7 9 10

Enter number that you want to search:

3

The number is at index: 5

3) Write a C Program to find largest value from array of 10 integers.

```
#include<stdio.h>
int main()  {
   int arr[10];
   printf("Enter any 10 numbers: \n");
   for(int i=0; i<10; i++) {
      scanf("%d",&arr[i]);
   }
   for(int i=0; i<10; i++) {
      printf("%d\t",arr[i]);
   }
   int max=arr[0];
   for(int i=0; i<10; i++)  {
      if(arr[i]>max){
         max=arr[i];   }
   }
   printf("\n---------------------------------------------------------------\n");
   printf("The Largest number from this array is : %d\n", max);
   printf("\n---------------------------------------------------------------\n");
   return 0;
}
```

OUTPUT

Enter any 10 numbers:

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10

The Largest number from this array is : 10

---

4) Write a C Program to Swap Max and Min Value from array of 10 integer value.

```
#include<stdio.h>
int main()
{
   int arr[10], max=0, min=arr[0],temp;
   printf("Enter any 10 integer:  \n");
   for(int i=0; i<10; i++) //input
   {
      scanf("%d",&arr[i]);
   }

   for(int i=0; i<10; i++) //check max value
   {
      if(arr[i]>max)
      {
         max=arr[i];
      }
   }
   for(int i=0; i<10; i++) //check min value
   {
      if(arr[i]<min)
      {
         min=arr[i];
      }
   }
   printf("\n-----------------------------------------------------\n");
   printf("Before swapping maximum = %d\tminimum = %d\n", max, min);
   temp=max; //swapping
```

```
    max=min;
    min=temp;
    printf("\n-----------------------------------------------------\n");
    printf("After swapping maximum = %d\tminimum = %d\n", max, min);
}
```

OUTPUT

Enter any 10 integer:

10 20 30 40 50 60 70 80 90 100

Before swapping maximum = 100 minimum = 10

After swapping maximum = 10 minimum = 100

5) Write a C Program to insert a value in array of 10 integers at specific position.

```
#include<stdio.h>
int main()
{
   int arr[11], value, pos;
   printf("Enter any 10 numbers: \n"); //input 10 number
   for(int i=0; i<10; i++)
   {
      scanf("%d", &arr[i]);
   }
   printf("-----------------------------------------------\n");
   printf("Enter position:\t"); //position at which you want to add number
   scanf("%d", &pos);
   printf("Enter value:\t"); //number that you want to add
   scanf("%d", &value);
   for(int i=10; i>=pos; i--) //shifting to next side
   {
      arr[i]=arr[i-1];
   }
   arr[pos-1]=value;
   printf("-----------------------------------------------\n");
   printf("-----------------------------------------------\n");
   for(int i=0; i<11; i++) //output  {
      printf("%d\t",arr[i]);
   }
}
```

OUTPUT

Enter any 10 numbers:

1 2 3 4 5 6 7 8 9 10

Enter position: 3

Enter value: 90

----

1 2 90 3 4 5 6 7 8 9 10

6) Write a C Program to delete a value in array of 10 integers from specific position.

```
#include<stdio.h>
int main()
{
   int arr[10], pos;
   printf("Enter any 10 numbers: \n"); //input
   for(int i=0; i<10; i++){
      scanf("%d", &arr[i]);
   }
   printf("Enter position: "); //position at which you want to delete element
   scanf("%d", &pos);
   int temp=arr[pos-1]; //here we make pos empty
   for(int i=pos; i<=9; i++) //shifting  {
      arr[i-1]=arr[i];
   }
   printf("\n-----------------------------------------------------------------\n");
   for(int i=0; i<9; i++){ //output
      printf("%d\t", arr[i]);
   }
}
```

OUTPUT

Enter any 10 numbers:

1 2 3 4 5 6 7 8 9 10

Enter position: 4

----

1 2 3 5 6 7 8 9 10

Question 1: What do you mean by nested structure?

A nested structure in C is a structure within structure. One structure can be declared inside another structure in the same way structure members are declared inside a structure.

Syntax:

```
struct name_1
{
   member1;
  member2;
  .
  .
  membern;
  struct name_2
  {
     member_1;
     member_2;
     .
     .
     member_n;
  }, var1
} var2;
```

Question 2: What is embedded C programming and how it is different from C programming?

Embedded C is generally used to develop microcontroller-based applications. C is a high-level programming language. Embedded C is just the extension variant of the C language. On the other hand, embedded C language is truly hardware dependent. The compilers in embedded C are OS independent. Here, we need a specific compiler that can help in generating micro-controller based results. Famous compilers used in embedded C are BiPOM Electronic, Green Hill Software and more.

Question 3: Explain reference and de-reference operator in pointers.

The reference operator (@expression) lets you refer to functions and variables indirectly, by name. It uses the value of its operand to refer to variable, the fields in a record, function, method, property or child window.

A dereference operator, which is also known as an indirection operator, operates on a pointer variable. It returns the location value, or l-value in memory pointed to by the variable's value. The deference operator is denoted with an asterisk (*).

Question 4: Explain 3 types of pointers.

Null Pointer

You create a null pointer by assigning the null value at the time of pointer declaration. This method is useful when you do not assign any address to the pointer. A null pointer always contains value 0.

Void Pointer

It is a pointer that has no associated data type with it. A void pointer can hold addresses of any type and can be typecast to any type. It is also called a generic pointer and does not have any standard data type. It is created by using the keyword void.

Wild Pointer

Wild pointers are also called uninitialized pointers. Because they point to some arbitrary memory location and may cause a program to crash or behave badly. This type of C pointer is not efficient. Because they may point to some unknown memory location which may cause problems in our program. This may lead to the crashing of the program. It is advised to be cautious while working with wild pointers.

PRACTICAL 3

1) Write a C Program to implement searching on ordered array of 10 integer value.

```
#include<stdio.h>
int main()
{
   int arr[10], x, count=0;
   printf("Enter any 10 numbers: \n");
   for(int i=0; i<10; i++)  {
      scanf("%d",&arr[i]);
   }
   printf("Enter number that you want to search: \n");
   scanf("%d", &x);
   for(int i=0; i<10; i++)
   {
      if(arr[i]==x)  {
         printf("-------------------------\n");
         printf("The number is at index: %d\n", i);
      }
   }
}
```

OUTPUT

Enter any 10 numbers:

1 2 3 4 5 6 7 8 9 10

Enter number that you want to search:

5

The number is at index: 4

PRACTICAL 3

2) Write a C Program to implement Stack with all necessary overflow and underflow condition (Use array as data structure). 1)PUSH 2) POP 3) DISPLAY.

```
#include<stdio.h>
int stack[10],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()  {
   top=-1;
   printf("Enter the size of STACK:\n");
   scanf("%d",&n);
   printf("STACK OPERATIONS USING ARRAY\n");
   printf("--------------------------------\n");
   printf("1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
   do   {
      printf("Enter your Choice from 1 to 4: \n");
      scanf("%d",&choice);
      switch(choice)  {
         case 1:   {
            push();
            break;
         }
         case 2: {
            pop();
            break;
         }
         case 3:  {
            display();
            break;
```

```
        }
        case 4:  {
           printf("\n\t EXIT POINT\n");
           break;
        }
        default:   {
           printf ("\n\t Please Enter a Valid Choice(1/2/3/4)\n");
        }
      }
   }
   while(choice!=4);
   return 0;
}
void push()  {
   if(top>=n-1)  {
      printf("STACK is over flow\n");
  }
   else  {
      printf("Enter a value to be pushed:");
      scanf("%d",&x);
      top++;
      stack[top]=x;
   }
}
void pop() {
   if(top<=-1)   {
      printf("Stack is under flow\n");
   }
   else   {
```

```
      printf("The popped elements is %d\n",stack[top]);
      top--;
   }
}
void display()  {
   if(top>=0)  {
      printf("The elements in STACK \n");
      for(i=top; i>=0; i--) {
         printf("|");
         printf("__%d__|\n",stack[i]);
      }
      printf("Press Next Choice\n");
   }
   else   {
      printf("The STACK is empty\n");
for(int i=0; i<n; i++) {
      printf("|____|\n");
   }
 }
}
```

OUTPUT

Enter the size of STACK:

5

STACK OPERATIONS USING ARRAY

1.PUSH

2.POP

3.DISPLAY

4.EXIT

Enter your Choice from 1 to 4:

1

Enter a value to be pushed:5

Enter your Choice from 1 to 4:

1

Enter a value to be pushed:4

Enter your Choice from 1 to 4:

1

Enter a value to be pushed:3

Enter your Choice from 1 to 4:

1

Enter a value to be pushed:2

Enter your Choice from 1 to 4:

1

Enter a value to be pushed:1

Enter your Choice from 1 to 4:

3

The elements in STACK

|__1__|

|__2__|

|__3__|

|__4__|

|__5__|

Press Next Choice

Enter your Choice from 1 to 4:

2

The popped elements is 1

Enter your Choice from 1 to 4:

2

The popped elements is 2

Enter your Choice from 1 to 4:

3

The elements in STACK

|__3__|

|__4__|

|__5__|

Press Next Choice

Enter your Choice from 1 to 4:

2

The popped elements is 3

Enter your Choice from 1 to 4:

2

The popped elements is 4

Enter your Choice from 1 to 4:

2

The popped elements is 5

Enter your Choice from 1 to 4:

2

Stack is under flow

Enter your Choice from 1 to 4:

3

The STACK is empty

|____|

|____|

|____|

|____|

|____|

Enter your Choice from 1 to 4:

4

EXIT POINT

3) Write a C Program to convert infix notation into its equivalent postfix notation using stack.

```
#include<stdio.h>
#include<ctype.h>
char stack[100];
int top = -1;
void push(char x)
{
   top++;
   stack[top] = x;
}
char pop()
{
   if(top == -1)
      return -1;
   else
      return stack[top--];
}
int priority(char x)
{
   if(x == '(')
      return 0;
   if(x == '+' || x == '-')
      return 1;
   if(x == '*' || x == '/')
      return 2;
   return 0;
}
int main()
```

```
{
   char exp[20];
   char *e, x;
   printf("Enter the expression : ");
   scanf("%s",exp);
   printf("----------------------------------------\n");
   printf("Expression in Infix form: %s\n", exp);
   printf("----------------------------------------\n");
   printf("Expression in Postfix form: ");
   e = exp;
   while(*e != '\0') {
      if(isalnum(*e)) { // it checks that is char is alphabet or not
        printf("%c ",*e);
      }
      else if(*e == '(') {
         push(*e);
      }
      else if(*e == ')') {
         while((x = pop()) != '('
           printf("%c ", x);
      }
      else {
         while(priority(stack[top]) >= priority(*e))
         printf("%c ",pop());
         push(*e);
      } e++;
   }
   while(top != -1)
   {
```

```
      printf("%c ",pop());
   }
}
```

OUTPUT

Enter the expression : A+B-C*D/E*F

Expression in Infix form: A+B-C*D/E*F

Expression in Postfix form: A B + C D * E / F * -

4) Write a c Program to implement Tower of Hanoi problem.

```
#include<stdio.h>
void towerofhanoi(int n, char src[6], char helper[6], char dest[6] )   {
   if(n==1) {
      printf("Transfer disc %d from %s to %s\n", n, src, dest);
      return;
   }
   towerofhanoi(n-1, src, dest, helper);
   printf("Transfer disc %d from %s to %s\n", n, src, dest);
   towerofhanoi(n-1, helper, src, dest);
}
int main()  {
   int n;
   do {
   printf("-----------------------------------------------\nEnter the number of disc: ");
   scanf("%d", &n);
   towerofhanoi(n, "S", "H", "D");
   } while(n!=0);
}
```

OUTPUT

Enter the number of disc: 1

Transfer disc 1 from S to D

Enter the number of disc: 2

Transfer disc 1 from S to H

Transfer disc 2 from S to D

Transfer disc 1 from H to D

Enter the number of disc: 3

Transfer disc 1 from S to D

Transfer disc 2 from S to H

Transfer disc 1 from D to H

Transfer disc 3 from S to D

Transfer disc 1 from H to S

Transfer disc 2 from H to D

Transfer disc 1 from S to D

Enter the number of disc: 0

5) Write a C Program to implement Infix to prefix conversion using stack.

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include <stdlib.h>
#define MAX 20
void push(int);
char pop();
void infix_to_prefix();
int precedence (char);
char stack[20],infix[20],prefix[20];
int top = -1;
int main()  {
   printf("\nINPUT THE INFIX EXPRESSION : ");
   scanf("%s",infix);
   infix_to_prefix();
   return 0;
}
void push(int pos) {
   if(top == MAX-1) {
   printf("\nOVERFLOW\n");
   }
   else {
   top++;
   stack[top] = infix[pos];
   }
}
char pop() {
   char ch;
```

```
    if(top < 0) {
        printf("\nSTACK UNDERFLOW\n");
    }
    else {
        ch = stack[top];
        top--;
        return(ch);
    }
    return 0;
}
void infix_to_prefix() {
    int i = 0,j = 0;
    strrev(infix);
        while(infix[i] != '\0') {
            if(infix[i] >= 'a' && infix[i] <= 'z') {
                prefix[j] = infix[i];
                j++;
                i++;
            }
            else if(infix[i] == ')') {
                push(i);
                i++;
            }
            else if(infix[i] == '(') {
                while(stack[top] != ')')    {
                        prefix[j] = pop();
                        j++;
                    }
                    pop();
```

```
            i++;
         }
         else {
            if(top == -1) {
               push(i);
               i++;
            }
            else if( precedence(infix[i]) < precedence(stack[top]))  {
               prefix[j] = pop();
               j++;
            while(precedence(stack[top]) > precedence(infix[i])) {
               prefix[j] = pop();
               j++;
               if(top < 0) {
                  break;
               }
            }
            push(i);
            i++;
         }
         else if(precedence(infix[i]) >= precedence(stack[top])) {
            push(i);
            i++;
         }
      }
   }
   while(top != -1) {
      prefix[j] = pop();
      j++;
```

```
    }
    strrev(prefix);
    printf("EQUIVALENT PREFIX NOTATION : %s ",prefix);
}
int precedence(char alpha) {
       if(alpha == '+' || alpha =='-') {
          return(1);
       }
       if(alpha == '*' || alpha =='/') {
          return(2);
       }
          return 0;
}
```

Output

INPUT THE INFIX EXPRESSION : a+b*c/d-e*f

EQUIVALENT PREFIX NOTATION : -+a/*bcd*ef

Question 1: Explain why stack is recursive data structure?

Thus, in recursion last function called needs to be completed first. Now Stack is a LIFO data structure i.e. (Last In First Out) and hence it is used to implement recursion. The High-level Programming languages, such as Pascal, C etc. that provides support for recursion use stack for book keeping.

Question 2: Real time applications of stack (at least 10 applications).

1. Converting infix to postfix expressions.
2. Undo/Redo button/operation in word processors.
3. Syntaxes in languages are parsed using stacks.
4. It is used in many virtual machines like JVM.
5. Forward-backward surfing in the browser.
6. History of visited websites.
7. Message logs and all messages you get are arranged in a stack.
8. Call logs, E-mails, Google photos' any gallery, YouTube downloads, Notifications (latest appears first).
9. Scratch cards earned after Google pay transaction.
10. Wearing/Removing Bangles, Pile of Dinner Plates, Stacked chairs.
11. Changing wearables on a cold evening, first in, comes out at last.
12. Last Hired, First Fired - which is typically utilized when a company reduces its workforce in an economic recession.
13. Loading bullets into the magazine of a gun. The last one to go in is fired first. Bam!
14. Java Virtual Machine.
15. Recursion.
16. Used in IDEs to check for proper parentheses matching.
17. Media playlist. To play previous and next song.

Question 3: What is complexity of push and pop for a stack implemented using array?

In push operation you add one element at the top of the stack so you make one step, so it takes constant time so push takes O(1).

In pop operation you remove one element from the top of the stack so you make one step, so it takes constant time so pop takes O(1).

1) Write a program to implement Quick Sort.

```
#include<stdio.h>
int partition(int arr[], int lb, int ub)
   {
      int pivot = arr[ub];
      int i = (lb - 1);
      for (int j = lb; j < ub; j++) {
         if (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
         }
      }
         int temp = arr[i+1];
         arr[i+1] = arr[ub];
         arr[ub] = temp;
         return(i+1);
}
void QuickSort(int arr[], int lb, int ub) {
   if (lb < ub) {
      int pi = partition(arr, lb, ub);
      QuickSort(arr, lb, pi - 1);
      QuickSort(arr, pi + 1, ub);
      }
}
int main() {
   int n=5,lb,ub;
   int arr[] = {5,4,3,2,1};
```

```
    QuickSort(arr, 0, n-1);
    for(int i=0; i<5; i++) {
        printf("%d ", arr[i]);
    }
}
```

OUTPUT

1 2 3 4 5

2) Write a program to implement Merge Sort.

```
#include <stdio.h>
void conquer(int arr[], int si, int mid, int ei)
{
   int merge[ei-si+1];
   int idx1 = si;
   int idx2 = mid+1;
   int x=0;
   while(idx1 <= mid && idx2 <= ei) {
      if(arr[idx1] < arr[idx2]) {
         merge[x++] = arr[idx1++];
      } else {
          merge[x++] = arr[idx2++];
      }
   }
   while(idx1 <= mid) {
      merge[x++] = arr[idx1++];
   }
   while(idx2 <= ei) {
      merge[x++] = arr[idx2++];
   }
   for(idx1=si,x=0; x<ei-si+1; x++,idx1++ ) {
      arr[idx1] = merge[x];
   }
}
void devide(int arr[], int si, int ei)
{
   if (si < ei) {
      int mid = si + (ei-si)/2;
```

```
        devide(arr, si, mid);
        devide(arr, mid + 1, ei);
        conquer(arr, si, mid, ei);
    }
}
int main()
{
    int n=5;
    int arr[] = {5,4,3,2,1};
    devide(arr, 0, n-1);
    for(int i=0; i<5; i++) {
        printf("%d ", arr[i]);
    }
}
```

OUTPUT

1 2 3 4 5

3) Write a program to implement Bubble Sort.

```
#include<stdio.h>
void bubblesort(int arr[]) {
   int temp, n=5;
   for(int i=0; i<n-1; i++) {
      for(int j=0; j<n-i-1; j++) {
         if(arr[j]>arr[j+1]) {
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }
      }
   }
   printf("-----------------\nSorted array is:\n-----------------\n");
   for(int i=0; i<5; i++) {
   printf(" %d ", arr[i]);
   }
}
int main() {
   int arr[] = {6, 5, 4, 3, 2};
   bubblesort(arr);
}
```

OUTPUT

Sorted array is:

----

2 3 4 5 6

4) Write a program to implement selection sort.

```
#include<stdio.h>
void selection_sort(int arr[])
 {
   int n=5,temp, min;
   for(int i=0; i<n-1; i++)
      {
      min=i;
      for(int j=i+1; j<n; j++) {
         if(arr[min]>arr[j]) {
            min = j;
         }
         if(min != i)
        {
            temp = arr[i];
            arr[i] = arr[min];
            arr[min] = temp;
         }
      }
   }
   printf("-----------------\nSorted array is:\n-----------------\n");
   for(int i=0; i<n; i++) {
      printf("%d ", arr[i]);
   }
}
int main()
{
   int arr[] = {6, 5, 4, 3, 2};
   selection_sort(arr);
```

}

OUTPUT

Sorted array is:

----

2 3 4 5 6

5) Write a program to implement Binary Search.

```
#include<stdio.h>
int BinarySearch(int a[], int i, int j, int key) {
   while(i<=j) {
      int mid = (i+j)/2;
      if(a[mid]==key) {
         return mid;
      }
      if(key > a[mid]) {
        return BinarySearch(a, mid+1, j, key);
      }
      if(key < a[mid]) {
         return BinarySearch(a, i, mid-1, key);
      }
   }
   return -1;
}
void main() {
   int a[] = {2, 3, 4, 5, 6};
   int n = sizeof(a) / sizeof(a[0]);
   int key = 4;
   int result = BinarySearch(a, 0, n-1, key);
   printf("The Element is found at position: %d", result);
}
```

OUTPUT

The Element is found at position: 2

Question 1: Explain worst case, average case and best-case time complexity with proper example.

Worst Case Analysis:

- In the worst-case analysis, we calculate the upper limit of the execution time of an algorithm. It is necessary to know the case which causes the execution of the maximum number of operations.
- For linear search, the worst case occurs when the element to search for is not present in the array. When x is not present, the search () function compares it with all the elements of arr [] one by one. Therefore, the temporal complexity of the worst case of linear search would be Θ (n).

Average Case Analysis:

- In the average case analysis, we take all possible inputs and calculate the computation time for all inputs. Add up all the calculated values and divide the sum by the total number of entries.
- We need to predict the distribution of cases. For the linear search problem, assume that all cases are uniformly distributed. So, we add all the cases and divide the sum by (n + 1).

Best Case Analysis:

- In the best-case analysis, we calculate the lower bound of the execution time of an algorithm. It is necessary to know the case which causes the execution of the minimum number of operations. In the linear search problem, the best case occurs when x is present at the first location.
- The number of operations in the best case is constant. The best-case time complexity would therefore be Θ (1).
- Most of the time, we perform worst-case analysis to analyse algorithms. In the worst analysis, we guarantee an upper bound on the execution time of an algorithm which is good information.

Question 2: Why sorting algorithm are important?

Since sorting can often reduce the complexity of a problem, it is an important algorithm in Computer Science. These algorithms have direct applications in searching algorithms, database algorithms, divide and conquer methods, data structure algorithms, and many more.

Reason 1: we can learn various problem-solving approaches using sorting algorithms.

Reason 2: we can use sorting to solve several coding problems.

Reason 3: analysis of sorting algorithms is the best idea to learn analysis of algorithms.

Reason 4: sorting is a good problem to understand several variations of a coding problem, boundary conditions, and code optimization technique.

Question 3: Classify sorting algorithms based on 5 parameters.

Sorting algorithms can be categorized based on the following parameters:

1) **The number of swaps or inversions required:** This is the number of times the algorithm swaps elements to sort the input. Selection sort requires the minimum number of swaps.
2) **The number of comparisons:** This is the number of times the algorithm compares elements to sort the input.
3) **Whether or not they use recursion:** Some sorting algorithms, such as quick sort, use recursive techniques to sort the input. Other sorting algorithms, such as selection sort or insertion sort, use non-recursive techniques. Finally, some sorting algorithms, such as merge sort, make use of both recursive as well as non-recursive techniques to sort the input.
4) **Whether they are stable or unstable:** Stable sorting algorithms maintain the relative order of elements with equal values, or keys. Unstable sorting algorithms do not maintain the relative order of elements with equal values / keys.
5) **The amount of extra space required:** Some sorting algorithms can sort a list without creating an entirely new list. These are known as in-place sorting algorithms.

Question 4: Differentiate each and every sorting algorithm with real time applications.

- A real-world example of a bubble sort algorithm is how the contact list on your phone is sorted in alphabetical order. Or the sorting of files on your phone according to the time they were added.
- Quick Sort is used in operational research and event-driven simulation. Numerical computations and in scientific research, for accuracy in calculations most of the efficiently developed algorithm uses priority queue and quick sort is used for sorting.
- Selection sort : For example, picking apples from a tree. We can see all the apples on the tree, find and pluck the biggest one, and drop it into our basket (which, we can assume, always contains the apples in a sorted manner).
- Merge sort is clearly the ultimate easy example of this. In real life, we tend to break things up along useful lines. If we're sorting change, we first divide the coins up by denominations, then total up each denomination before adding them together.

1) Write a C program to implement simple queue (Insertion, deletion and traversal).

```
#include<stdio.h>
int choice,front=-1,rear = -1,n,Queue[20],x;
void enqueue();
void dequeue();
void display();
int main() {
   printf("Enter size of array : ");
   scanf("%d", &n);
   do {
      printf("\nEnter choice : ");
      scanf("%d", &choice);
      switch(choice) {
         case 1 :  {
            enqueue();
            break;
         }
         case 2 :  {
            dequeue();
            break;
         }
         case 3 :  {
            display();
            break;
         }
         case 4 :  {
            printf("Exit");
            break;
```

```
            }
            default: {
               printf("Enter valid number");
               break;
            }
         }
      } while(choice!=4);
   }
   void enqueue() {
      if(rear == n-1) {
         printf("Overflow\n");
      }
      else {
         if(front == -1) {
            front = 0;
         }
            rear++;
            printf("Enter Element  : ");
            scanf("%d", &x);
            Queue[rear] = x;
         }
      }
   void dequeue() {
      if(front == -1 || front > rear) {
         printf("Overflow\n");
      }
      else {
         printf("Element deleted");
         front++;
```

```
    }
  }
  void display()  {
    if(front == -1 || front > rear) {
      printf("Overflow\n");
    }
    else {
      for(int i=front; i<=rear; i++) {
        printf(" %d ", Queue[i]);
      }
    }
  }
```

OUTPUT :

Enter size of array : 3

Enter choice : 1

Enter Element : 10

Enter choice : 1

Enter Element : 20

Enter choice : 1

Enter Element : 30

Enter choice : 1

Overflow

Enter choice : 2

Element deleted

Enter choice : 3

20 30

Enter choice : 4 Exit

2) Write a C program to implement circular queue (Insertion, deletion and traversal).

```
#include<stdio.h>
int choice,front=-1,rear = -1,size,Queue[20],x;
void enqueue();
void dequeue();
void display();
int main() {
   printf("Enter size of array : ");
   scanf("%d", &size);
   do {
      printf("\nEnter choice : ");
      scanf("%d", &choice);
      switch(choice) {
         case 1 :  {
            enqueue();
            break;
         }
         case 2 :  {
            dequeue();
            break;
         }
         case 3 :   {
            display();
            break;
         }
         case 4 :    {
            printf("Exit");
            break;
```

```
            }
        }
    } while(choice!=4);
}
void enqueue() {
    if((rear+1)%size == front) {
        printf("Queue is overflow");
        return;
    }
    printf("enter Element : ");
    scanf("%d", &x);
    if(front == -1) {
        front = 0;
    }
    rear = (rear+1)%size;
    Queue[rear] = x;
}
void dequeue() {
    if(front == -1 && rear == -1) {
        printf("Queue is empty");
    }
    if(front == rear) {
        front = rear = -1;
    }
    printf("%d is deleted", Queue[front]);
    front = (front+1)%size;
}
void display() {
    if(front == -1 && rear == -1) {
```

```
        printf("Queue is empty");
    }
    else {
        for(int i = front; i != rear; i = (i+1)%size) {
            printf("%d ", Queue[i]);
        }
        printf("%d ", Queue[rear]);
    }
}
```

OUTPUT :

Enter size of array : 5

Enter choice : 1

enter Element : 10

Enter choice : 1

enter Element : 20

Enter choice : 1

enter Element : 30

Enter choice : 1

enter Element : 40

Enter choice : 1

enter Element : 50

Enter choice : 1

Queue is overflow

Enter choice : 3

10 20 30 40 50

Enter choice : 2

10 is deleted

Enter choice : 2

20 is deleted

Enter choice : 1

enter Element : 60

Enter choice : 3

30 40 50 60

Enter choice : 1

enter Element : 70

Enter choice : 3

30 40 50 60 70

Enter choice : 3

30 40 50 60 70

Enter choice : 1

Queue is overflow

Enter choice : 4

Exit

3) Write a C program to implement priority queue (Insertion , deletion and traversal).

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
void insert_by_priority(int);
void delete_by_priority(int);
void create();
void check(int);
void display_pqueue();
int pri_que[MAX];
int front, rear;
void main()
{
   int n, ch;
   printf("\n1 - Insert an element into queue");
   printf("\n2 - Delete an element from queue");
   printf("\n3 - Display queue elements");
   printf("\n4 - Exit");
   create();
   while (1) {
      printf("\nEnter your choice : ");
      scanf("%d", &ch);
      switch (ch) {
      case 1:
         printf("\nEnter value to be inserted : ");
         scanf("%d",&n);
         insert_by_priority(n);
         break;
```

```
        case 2:
            printf("\nEnter value to delete : ");
            scanf("%d",&n);
            delete_by_priority(n);
            break;
        case 3:
            display_pqueue();
            break;
        case 4:
            exit(0);
        default:
            printf("\nChoice is incorrect, Enter a correct choice");
        }
    }
}
void create() {
    front = rear = -1;
}
void insert_by_priority(int data) {
    if (rear >= MAX - 1) {
        printf("\nQueue overflow no more elements can be inserted");
        return;
    }
    if ((front == -1) && (rear == -1)) {
        front++;
        rear++;
        pri_que[rear] = data;
        return;
    }
```

```
    else
        check(data);
    rear++;
}
void check(int data) {
    int i,j;
    for (i = 0; i <= rear; i++) {
        if (data >= pri_que[i]) {
            for (j = rear + 1; j > i; j--) {
                pri_que[j] = pri_que[j - 1];
            }
            pri_que[i] = data;
            return;
        }
    }
    pri_que[i] = data;
}
void delete_by_priority(int data) {
    int i;
    if ((front==-1) && (rear==-1))  {
        printf("\nQueue is empty no elements to delete");
        return;
    }
    for (i = 0; i <= rear; i++) {
        if (data == pri_que[i]) {
            for (; i < rear; i++) {
                pri_que[i] = pri_que[i + 1];
            }
        pri_que[i] = -99;
```

```
        rear--;
        if (rear == -1)
           front = -1;
        return;
        }
    }
    printf("\n%d not found in queue to delete", data);
}
void display_pqueue() {
    if ((front == -1) && (rear == -1)) {
        printf("\nQueue is empty");
        return;
    }
    for (; front <= rear; front++) {
        printf(" %d ", pri_que[front]);
    }
    front = 0;
}
```

OUTPUT

1 - Insert an element into queue

2 - Delete an element from queue

3 - Display queue elements

4 - Exit

Enter your choice : 1

Enter value to be inserted : 30

Enter your choice : 1

Enter value to be inserted : 20

Enter your choice : 1

Enter value to be inserted : 10

Enter your choice : 1

Enter value to be inserted : 40

Enter your choice : 1

Enter value to be inserted : 50

Enter your choice : 3

50 40 30 20 10

Enter your choice : 2

Enter value to delete : 20

Enter your choice : 3

50 40 30 10

Enter your choice : 4

Q1: List some Queue real-life applications (at least 10 applications)

1. Job Scheduling
2. Multilevel Queue Scheduling
3. As a Buffer Space
4. Used in implementing Prim's algorithm
5. Data Compression in WINZIP / GZIP
6. Load balancing and Interrupt handling
7. CPU scheduling, Disk Scheduling.
8. When data is transferred asynchronously between two processes. Examples include IO Buffers, pipes, etc.
9. Memory management
10. Queues in routers/ switches
11. Mail Queues
12. Applied to add song at the end or to play from the front.
13. Applied as buffers on MP3 players and portable CD players.

Q2: What are some types of Queues?

There are five different types of queues that are used in different scenarios. They are:

Input Restricted Queue (this is a Simple Queue)

1. Output Restricted Queue (this is also a Simple Queue)
2. Circular Queue
3. Double Ended Queue (Deque)
4. Priority Queue
    - Ascending Priority Queue
    - Descending Priority Queue

Q3: Why and when should I use Stack or queue data structures instead of Arrays/Lists?

Because Stack and queue assist you in managing your data in a more specific manner than arrays and lists. It means you won't have to wonder if someone placed an element in the midst of your list at random, messing up certain invariants when troubleshooting an issue. Random access is the nature of arrays and lists. They're incredibly adaptable, but they're also easily corruptible. If you wish to keep track of your data,It's recommended to use those, previously implemented, collections when storing data as FIFO or LIFO.

- We use stack or queue instead of arrays/lists when we want the elements in a specific order i.e. in the order we put them (queue) or in the reverse order (stack).

- Queues and stacks are dynamic while arrays are static. So when we require dynamic memory we use queue or stack over arrays.

Q4: Implement a Queue using two stacks.

Enqueue operation:
1. Simply push the elements into the first stack.
Dequeue operation:
1. Pop from the second stack if the second stack is not empty.
2. If second stack is empty, pop from the first stack and push all the elements into second until the first stack becomes empty.
3. Now pop an element from the second stack.

1) Write a c program to implement singly linked list for the following function.

(a) Insert a node at the front of the linked list.

(b) Insert a node at the end of the linked list.

(c) Insert a node in order.

(d) Delete any node from the linked list.

(e) Count total no of nodes in list.

(f) Insert a node at any position in linked list.

(g) Display all nodes (traversal of Linked list).

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int data;
   struct node *nxt;
};
struct node *head;
void insertAtFirst() {
   struct node *NewNode;
   int value;
   NewNode = (struct node *) malloc(sizeof(struct node *));
   if(NewNode == NULL)
      printf("\nOVERFLOW");
   else
      printf("\nEnter value\n");
      scanf("%d",&value);
      NewNode->data = value;
      NewNode->nxt = head;
      head = NewNode;
}
```

```
void insertAtLast() {
   struct node *NewNode,*temp;
   int item;
   NewNode = (struct node*)malloc(sizeof(struct node));
   if(NewNode == NULL) {
      printf("\nOverFlow");
   }
   else {
      printf("\nEnter value\n");
      scanf("%d",&item);
      NewNode->data = item;
   if(head == NULL) {
      NewNode -> nxt = NULL;
      head = NewNode;
   }
   else
      temp = head;
   while (temp -> nxt != NULL) {
      temp = temp -> nxt;
   }
      temp->nxt = NewNode;
      NewNode->nxt = NULL;
   }
}
void RandomInsert() {
   int i,loc,item;
   struct node *Newnode, *temp;
   Newnode = (struct node *) malloc (sizeof(struct node));
   if(Newnode == NULL) {
```

```
        printf("\nOVERFLOW");
    }
    else {
        printf("\nEnter element value");
        scanf("%d",&item);
        Newnode->data = item;
        printf("\nEnter the location after which you want to insert ");
        scanf("\n%d",&loc);
        temp=head;
        for(i=0;i<loc;i++) {
            temp = temp->nxt;
            if(temp == NULL) {
                printf("\ncan't insert\n");
                return;
            }
        }
        Newnode ->nxt = temp ->nxt;
        temp ->nxt = Newnode;
        }
}
void display() {
    struct node *temp;
    temp = head;
    if(temp == NULL)
        printf("Nothing to print");
    else
        while (temp!=NULL) {
        printf(" %d -->",temp->data);
        temp = temp -> nxt;
```

```
        }
}
void count() {
    struct node *temp;
    temp = head;
    int count=0;
    if(temp == NULL)
        printf("Zero node!");
    else {
        while (temp!=NULL) {
        temp = temp -> nxt;
        count++;
        }
        printf(" %d ", count);
    }
}
void DeleteAtFirst() {
    struct node *ptr;
    if(head == NULL)
        printf("\nList is empty\n");
    else
        ptr = head;
        head = ptr->nxt;
        free(ptr);
    }
void DeleteAtLast()
    {
    struct node *ptr,*ptr1;
    if(head == NULL)
```

```
        printf("\nlist is empty");
    else if(head -> nxt == NULL) {
        head = NULL;
        free(head);
    }
    else {
        ptr = head;
        while(ptr->nxt != NULL) {
        ptr1 = ptr;
        ptr = ptr ->nxt;
        }
    ptr1->nxt = NULL;
    free(ptr);
    }
}
void RandomDelete() {
    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter the location of the node after which you want to perform deletion \n");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++) {
        ptr1 = ptr;
        ptr = ptr->nxt;
        if(ptr == NULL) {
            printf("\nCan't delete");
            return;
        }
    }
```

```
    ptr1 ->nxt = ptr ->nxt;
    free(ptr);
    printf("\nDeleted node %d ",loc+1);
    }
void main () {
    int choice =0;
    printf("Choose any number from following:\n");
    printf("1.Insert at First\n2.Insert at last\n3.Insert at Specific Position\n4.Delete at
First\n5.Delete at Last\n6.Random Delete\n7.display\n8.Count\n");
    while(choice != 9) {
    printf("\nEnter your choice?\n");
    scanf("\n%d",&choice);
    switch(choice) {
        case 1:
        insertAtFirst();
        break;
        case 2:
        insertAtLast();
        break;
        case 3:
        RandomInsert();
        break;
        case 4:
        DeleteAtFirst();
        break;
        case 5:
        DeleteAtLast();
        break;
        case 6:
        RandomDelete();
```

```
        break;
        case 7:
        display();
        break;
        case 8:
        count();
        break;
        case 9:
        exit(0);
        break;
        default:
        printf("Please enter valid choice..");
        break;
        }
    }
}
```

OUTPUT:

Choose any number from following:

1.Insert at First

2.Insert at last

3.Insert at Specific Position

4.Delete at First

5.Delete at Last

6.Random Delete

7.display

8.Count

Enter your choice?

1

Enter value

10

Enter your choice?

2

Enter value

20

Enter your choice?

3

Enter element value30

Enter the location after which you want to insert 1

Enter your choice?

7

10 --> 20 --> 30 -->

Enter your choice?

1

Enter value

40

Enter your choice?

1

Enter value

50

Enter your choice?

7

50 --> 40 --> 10 --> 20 --> 30 -->

Enter your choice?

4

Enter your choice?

5

Enter your choice?

6

Enter the location of the node after which you want to perform deletion

1

Deleted node 2

Enter your choice?

7

40 --> 20 -->

Enter your choice?

8

2

Enter your choice?

9

2) Write a program to implement following operations on the Doubly linked list.

(a) Insert a node at the front of the linked list.

(b) Insert a node at the end of the linked list.

(c) Insert a node at any position in linked list.

(d) Delete any node from the linked list.

(e) Count total no of nodes in list.

(f) Insert a node at any position in linked list.

(g) Display all nodes (traversal of Linked list).

```
#include<stdio.h>
#include<stdlib.h>
int data;
struct node {
	int data;
	struct node *next;
	struct node *prev;
};
struct node *head;
void addFirst() {
	struct node *newNode;
	newNode = (struct node *)malloc(sizeof(struct node));
	if(newNode == NULL){
		printf("\n--Overflow--");
} else {
	printf("Enter data : ");
	scanf("%d", &data);
	newNode -> data = data;
	if(head == NULL) {
		head = newNode;
```

```
                newNode -> next = NULL;
                head -> prev = NULL;
        } else {
                head -> prev = newNode;
                newNode -> next = head;
                newNode -> prev = NULL;
                head = newNode;
                }
        }
        printf("\n--Node inserted--\n");
}
void addLast() {
        struct node *ptr,*newNode;
        newNode = (struct node *)malloc(sizeof(struct node));
        if(newNode == NULL) {
                printf("\n--Overflow--");
        } else {
                printf("Enter data : ");
                scanf("%d", &data);
                newNode -> data = data;
                        if(head == NULL) {
                                head = newNode;
                                newNode -> next = NULL;
                                head -> prev = NULL;
                        } else {
                                ptr = head;
                                while(ptr -> next != NULL) {
                                ptr = ptr -> next;
                        }
```

```
                ptr -> next = newNode;
                newNode -> prev = ptr;
                newNode -> next = NULL;
                }
        }
        printf("\n--Node inserted--\n");
}
void removeFirst() {
        struct node *ptr;
                if(head == NULL) {
                        printf("\n--can't delete--");
                }
                if(head -> next == NULL) {
                        head = NULL;
                        free(head);
                        printf("--Node deleted from the begining--\n");
                        return;
                } else {
                        ptr = head;
                        head = ptr -> next;
                        head -> prev = NULL;
        free(ptr);
        printf("--Node deleted from the begining--\n");
        }
}
void removeLast() {
        struct node *ptr;
                if(head == NULL) {
                        printf("\n--can't delete--");
```

```
                }
        if(head -> next == NULL) {
                head = NULL;
                free(head);
                printf("--Node deleted from the last--\n");
                return;
        } else {
                ptr = head;
                while(ptr -> next != NULL) {
                ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        }
    printf("--Node deleted from the last--\n");
}
void count() {
    struct node *ptr;
    int count = 0;
    ptr = head;
        if(head == NULL) {
                printf("\nDoubly LL size : 0");
        } else {
                while(ptr != NULL) {
                count++;
                ptr = ptr -> next;
        }
    printf("Doubly LL size : %d ", count);
    }
```

```
}
void display() {
	struct node *ptr;
	ptr = head;
		if(head == NULL) {
			printf("\nnothing to print");
		} else {
			printf("\n printing values ... \n");
			printf("NULL <-->");
			while(ptr -> next != NULL) {
			printf(" %d <--> ", ptr -> data);
			ptr = ptr -> next;
		}
	printf(" %d <--> ", ptr -> data);
	printf("NULL\n");
	}
}
void main() {
	int choice = 0;
	printf("\n1.Insert in begining\t\t2.Insert at last\n3.Delete from Beginning\t\t4.Delete
	from last\n5.Display LL\t\t\t6.count nodes\ n7.EXIT\n");
	while(choice != 7) {
		printf("\nEnter your choice : ");
		scanf("\n%d",&choice);
		switch(choice) {
			case 1:
			addFirst();
			break;
			case 2:
```

```
                addLast();
                break;
                case 3:
                removeFirst();
                break;
                case 4:
                removeLast();
                break;
                case 5:
                display();
                break;
                case 6:
                count();
                break;
                case 7:
                printf("---EXIT---\n");
                break;
                default:
                printf("Please enter valid choice..");
            }
        }
}
```

OUTPUT:

1.Insert in begining **2.Insert at last**

3.Delete from Beginning **4.Delete from last**

5.Display LL **6.count nodes**

7.EXIT

Enter your choice : 1

Enter data : 10

--Node inserted--

Enter your choice : 2

Enter data : 20

--Node inserted--

Enter your choice : 3

--Node deleted from the begining--

Enter your choice : 5

printing values ...

NULL <--> 20 <--> NULL

Enter your choice : 1

Enter data : 30

--Node inserted--

Enter your choice : 1

Enter data : 40

--Node inserted--

Enter your choice : 1

Enter data : 50

--Node inserted--

Enter your choice : 5

printing values ...

NULL <--> 50 <--> 40 <--> 30 <--> 20 <--> NULL

Enter your choice : 6

Doubly LL size : 4

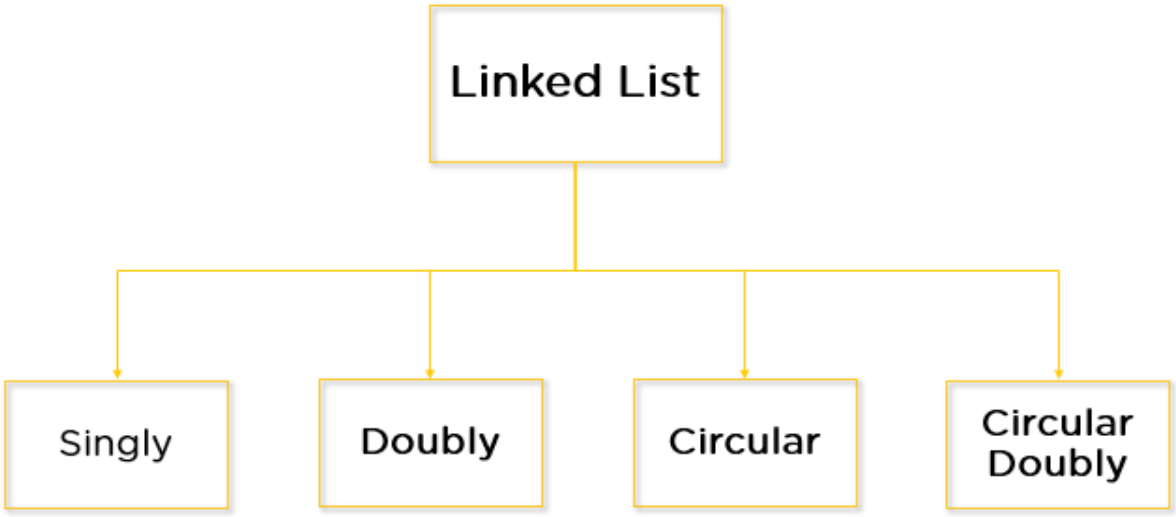
Enter your choice : 10

Please enter valid choice..

Enter your choice : 7

---EXIT---

Question 1: How many types of Link List exists? Explain each of them.



There are four key types of linked lists:

- Singly linked lists
- Doubly linked lists
- Circular linked lists
- Circular doubly linked lists

1. Singly Linked List

It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. A single linked list allows the traversal of data only in one way.

2. Doubly Linked List

A doubly linked list or a two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in sequence. Therefore, it contains three parts of data, a pointer to the next node, and a pointer to the previous node. This would enable us to traverse the list in the backward direction as well.

3. Circular Linked List

A circular linked list is that in which the last node contains the pointer to the first node of the list. While traversing a circular linked list, we can begin at any node and traverse the list in any direction forward and backward until we reach the same node we started. Thus, a circular linked list has no beginning and no end.

4. Doubly Circular linked list

A Doubly Circular linked list or a circular two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in the sequence. The difference between the doubly linked and circular doubly list is the same as that between a singly linked list and a circular linked list. The circular doubly linked list does not contain null in the previous field of the first node.

Question 2: What are the main differences between the Linked List and Linear array? (At least 4 points including pros and cons of both).

Difference between Linear Array and Linked List

S.No.	LINEAR ARRAY	LINKED LIST
1.	An array is a grouping of data elements of equivalent data type.	A linked list is a group of entities called a node. The node includes two segments: data and address.
2.	It stores the data elements in a contiguous memory zone.	It stores elements randomly, or we can say anywhere in the memory zone.
3.	In the case of an array, memory size is fixed, and it is not possible to change it during the run time.	In the linked list, the placement of elements is allocated during the run time.
4.	The elements are not dependent on each other.	The data elements are dependent on each other.
5.	The memory is assigned at compile time.	The memory is assigned at run time.
6.	It is easier and faster to access the element in an array.	In a linked list, the process of accessing elements takes more time.

Question 3: Mention few real time applications of Link List. (At least 4 applications)

1. Image viewer – Previous and next images are linked and can be accessed by the next and previous buttons.
2. Previous and next page in a web browser – We can access the previous and next URL searched in a web browser by pressing the back and next buttons since they are linked as a linked list.
3. Music Player – Songs in the music player are linked to the previous and next songs. So you can play songs either from starting or ending of the list.
4. Redo and undo functionality.
5. Use of the Back and forward button in a browser.
6. The most recently used section is represented by the Doubly Linked list.
7. Other Data structures like Stack, HashTable, and BinaryTree can also be applied by Doubly Linked List.

Question 4: Where will be the free node available while inserting new node in a Liked List?

If a new node is inserted in a linked list, the free node is found in Avail List.

1) Write a program to implement following operations on the Circular linked list.

(a) Insert a node at the front of the linked list.

(b) Insert a node at the end of the linked list.

(c) Insert a node in order.

(d) Delete any node from the linked list.

(e) Count total no of nodes in list.

(f) Display all nodes (traversal of Linked list).

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head;
void insertatFirst() {
   struct node *ptr,*temp;
   int item;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL) {
      printf("\nOVERFLOW");
   }
   else {
      printf("\nEnter the node data?");
      scanf("%d",&item);
      ptr -> data = item;
      if(head == NULL) {
         head = ptr;
         ptr -> next = head;
      } else {
         temp = head;
         while(temp->next != head)
            temp = temp->next;
            ptr->next = head;
            temp -> next = ptr;
            head = ptr;
      }
      printf("\nnode inserted\n");
   }
```

```
}
void insertatLast() {
   struct node *ptr,*temp;
   int item;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL) {
      printf("\nOVERFLOW\n");
   }
   else {
      printf("\nEnter Data?");
      scanf("%d",&item);
      ptr->data = item;
      if(head == NULL) {
         head = ptr;
         ptr -> next = head;
      }
      else  {
         temp = head;
         while(temp -> next != head) {
            temp = temp -> next;
         }
         temp -> next = ptr;
         ptr -> next = head;
      }
      printf("\nnode inserted\n");
   }
}
void begin_delete() {
   struct node *ptr;
   if(head == NULL) {
      printf("\nUNDERFLOW");
   }
   else if(head->next == head) {
      head = NULL;
      free(head);
      printf("\nnode deleted\n");
   }
   else {
      ptr = head;
      while(ptr -> next != head)
         ptr = ptr -> next;
         ptr->next = head->next;
         free(head);
         head = ptr->next;
         printf("\nnode deleted\n");
   }
}
```

```
void last_delete() {
   struct node *ptr, *preptr;
   if(head==NULL) {
      printf("\nUNDERFLOW");
   }
   else if (head ->next == head) {
      head = NULL;
      free(head);
      printf("\nnode deleted\n");
   }
   else {
      ptr = head;
      while(ptr ->next != head) {
         preptr=ptr;
         ptr = ptr->next;
      }
      preptr->next = ptr -> next;
      free(ptr);
      printf("\nnode deleted\n");
   }
}

void search() {
   struct node *ptr;
   int item,i=0,flag=1;
   ptr = head;
   if(ptr == NULL) {
      printf("\nEmpty List\n");
   }
   else {
      printf("\nEnter item which you want to search?\n");
      scanf("%d",&item);
      if(head ->data == item) {
         printf("item found at location %d",i+1);
         flag=0;
      }
      else {
         while (ptr->next != head) {
            if(ptr->data == item) {
               printf("item found at location %d ",i+1);
               flag=0;
               break;
            }
            else {
               flag=1;
            }
         i++;
```

```
            ptr = ptr -> next;
            }
         }
         if(flag != 0) {
            printf("Item not found\n");
         }
      }
   }

   void display() {
      struct node *ptr;
      ptr=head;
      if(head == NULL) {
         printf("\nnothing to print");
      }
      else {
         printf("\n printing values ... \n");
         while(ptr -> next != head) {
            printf("%d --> ", ptr -> data);
            ptr = ptr -> next;
         }
      printf("%d ", ptr -> data);
      }
   }
   void count() {
      struct node *temp;
      temp = head;
      int count=0;
      if(temp == NULL)
         printf("Zero node!");
      else {
         while (temp-> next != head) {
         temp = temp -> next;
         count++;
         }
         printf("%d ", count+1);
      }
   }

   void main () {
      int choice =0;
      printf("------------------------------------------------");
      printf("\nSelect Choice:");
      printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from
   last\n5.Search for an element\n6.Show\n7.count\n");
      printf("------------------------------------------------");
      while(choice != 8) {
```

```
      printf("\nEnter your choice?\n");
      scanf("\n%d",&choice);
      switch(choice) {
         case 1:
         insertatFirst();
         break;
         case 2:
         insertatLast();
         break;
         case 3:
         begin_delete();
         break;
         case 4:
         last_delete();
         break;
         case 5:
         search();
         break;
         case 6:
         display();
         break;
         case 7:
         count();
         break;
         case 8:
         exit(0);
         break;
         default:
         printf("Please enter valid choice..");
      }
   }
}
```

OUTPUT:

Select Choice:

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.count

Enter your choice?

1

Enter the node data?10

node inserted

Enter your choice?

2

Enter Data?20

node inserted

Enter your choice?

6

printing values ...

10 --> 20

Enter your choice?

7

2

Enter your choice?

1

Enter the node data?30

node inserted

Enter your choice?

1

Enter the node data?40

node inserted

Enter your choice?

6

printing values ...

40 --> 30 --> 10 --> 20

Enter your choice?

3

node deleted

Enter your choice?

4

node deleted

Enter your choice?

6

printing values ...

30 --> 10

Enter your choice?

5

Enter item which you want to search?

30

item found at location 1

Enter your choice?

7

2

Enter your choice?

8

Question 1: How will you convert a binary tree into doubly linked list?

Given a Binary Tree (Bt), convert it to a Doubly Linked List (DLL). The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be the same as in Inorder for the given Binary Tree.

Question 2: Why is merge sort is a better option than quick sort for linked list?

Merge Sort is the preferred choice over Quick Sort when:

- Data to be sorted is in Linked Lists
- When the sorting algorithm needs to run in parallel
- When a stable sorting is needed
- Data to be sorted is in Linked Lists.
- No other sorting algorithm performs better than Merge Sort on Linked Lists. This is because access takes linear time O(N) on Linked List which makes the performance of a traditional sorting algorithm worse.
- Merge Sort performs best on Linked Lists.
- When the sorting algorithm needs to run in parallel

Question 3: Given a 2-D matrix. You need to convert it into a linked list matrix such that each node is Liked to its nest right and down node and display it. (Write a program)

```
#include <bits/stdc++.h>

using namespace std;

struct Node {

        int data;

        Node* right, *down;

};

Node* construct(int arr[][4], int i, int j,

int m, int n, vector<vector<Node*>> &visited)

{

if (i > m - 1 || j > n - 1)
```

```
        return NULL;
if(visited[i][j]){
        return visited[i][j];
}
        Node* temp = new Node();
        visited[i][j] = temp;
        temp->data = arr[i][j];
        temp->right = construct(arr, i, j + 1, m, n, visited);
        temp->down = construct(arr, i + 1, j, m, n, visited);
        return temp;
}
void display(Node* head)
{
        Node* Rp;
        Node* Dp = head;
        while (Dp) {
        Rp = Dp;
        while (Rp) {
        cout << Rp->data << " ";
        Rp = Rp->right;
}
        cout << "\n";
        Dp = Dp->down;
}
}
int main()
{
        int arr[][4] = {
        { 1, 2, 3, 0},
```

```
        { 4, 5, 6 , 1},
        { 7, 8, 9 , 2},
        { 7, 8, 9 , 2}
};
        int m = 4, n = 4;
        vector<vector<Node*>> visited(m, vector<Node*>(n));
        Node* head = construct(arr, 0, 0, m, n, visited);
        display(head);
        return 0;
}
```

Output

```
1 2 3 0
4 5 6 1
7 8 9 2
7 8 9 2
```

1) Write a c program to implement stack using Linked list.
(PUSH, POP, DISPLAY).

```
#include <stdio.h>
#include <stdlib.h>
        struct node {
        int val;
        struct node *next;
};
struct node *head;
void push ()  {
        int val;
        struct node *ptr = (struct node*)malloc(sizeof(struct node));
        if(ptr == NULL) {
        printf("not able to push the element");
        } else  {
                printf("Enter the value : ");
                scanf("%d",&val);
                if(head==NULL)  {
                        ptr->val = val;
                        ptr -> next = NULL;
                        head=ptr;
                } else  {
                        ptr->val = val;
                        ptr->next = head;
                        head=ptr;
                }
                printf("--Item pushed--");
        }
}
void pop()  {
        int item;
        struct node *ptr;
        if (head == NULL)  {
                printf("Underflow");
        }  else {
                item = head->val;
                ptr = head;
                head = head->next;
                free(ptr);
                printf("-- %d is popped--", item);
        }
}
void display()  {
        int i;
        struct node *ptr;
```

```
        ptr=head;
        if(ptr == NULL) {
                printf("Stack is empty\n");
        } else {
                printf("\n-- Stack --\n");
                while(ptr!=NULL)  {
                        printf(" %d\n",ptr->val);
                        ptr = ptr->next;
                }
        }
}
void main () {
        int choice=0;
        printf("--STACK USING LINKLIST--\n");
        printf("\n1.Push\t\t\t2.Pop\n3.Display\t\t4.Exit\n");
        while(choice != 4) {
                printf("\nEnter your choice : ");
                scanf("%d",&choice);
                switch(choice)  {
                        case 1:
                                push();
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                printf("--EXIT--");
                                break;
                        default:
                                printf("Please Enter valid choice ");
                }
        }
}
```

OUTPUT :

--STACK USING LINKLIST--

1.Push 2.Pop
3.Display 4.Exit

Enter your choice : 1
Enter the value : 3

--Item pushed--
Enter your choice : 1
Enter the value : 2
--Item pushed--
Enter your choice : 1
Enter the value : 1
--Item pushed--
Enter your choice : 3
-- Stack --
1
2
3
Enter your choice : 2
-- 1 is popped--
Enter your choice : 3
-- Stack --
2
3
Enter your choice : 4
--EXIT--

2) Write a c program to implement queue using Linked List. (enqueue, dequeue, display).

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

void enqueue() {
        struct node *newNode,*temp;
        int item;
        newNode = (struct node*)malloc(sizeof(struct node));
        if(newNode == NULL) {
                printf("OVERFLOW");
        } else {
                printf("Enter value : ");
                scanf("%d",&item);
                newNode->data = item;
                if(head == NULL) {
                        newNode -> next = NULL;
                        head = newNode;
                        printf("--element added--");
                } else {
                        temp = head;
                        while (temp -> next != NULL) {
                                temp = temp -> next;
                        }
                        temp->next = newNode;
                        newNode->next = NULL;
                        printf("--element added--");
                }
        }
}
void dequeue() {
        struct node *ptr;
        int val;
        if(head == NULL) {
                printf("\nQueue Empty\n");
} else {
        ptr = head;
        val = ptr -> data;
        head = ptr->next;
        free(ptr);
        printf("-- %d is deleted from queue--", val);
```

```
        }
}

void display()
{
        int i;
        struct node *ptr;
        ptr=head;
        if(ptr == NULL)
        {
                printf("Queue is empty\n");
        }
        else
        {
                printf("\n-- Queue --\n");
                while(ptr!=NULL)
                {
                                printf(" %d", ptr->data);
                                ptr = ptr->next;
                }
        }
}

void main ()
{
        int choice=0;
        printf("--Queue USING LINKLIST--\n");
        printf("\n1.enqueue\t\t2.dequeue\n3.Display\t\t4.Exit\n");
        while(choice != 4) {
                printf("\nEnter your choice : ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:
                        enqueue();
                        break;
                        case 2:
                        dequeue();
                        break;
                        case 3:
                        display();
                        break;
                        case 4:
                        printf("--EXIT--");
                        break;
                        default:
                        printf("Please Enter valid choice ");
                }
```

```
        }
}
```

OUTPUT :

--Queue USING LINKLIST--

1.enqueue 2.dequeue
3.Display 4.Exit

Enter your choice : 1
Enter value : 1
--element added--
Enter your choice : 1
Enter value : 2
--element added--
Enter your choice : 1
Enter value : 3
--element added--
Enter your choice : 3

-- Queue --
1 2 3
Enter your choice : 2
-- 1 is deleted from queue--
Enter your choice : 3

-- Queue --
2 3
Enter your choice : 4
--EXIT--

3) Write a c program to implement Double-ended queue using Linked List. (enqueue, dequeue , display).

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

void enqueuefirst() {
	struct node *newNode;
	int item;
	newNode = (struct node *) malloc(sizeof(struct node *));
	if(newNode == NULL) {
		printf("--OVERFLOW--");
	} else {
		printf("Enter value : ");
		scanf("%d",&item);
		newNode->data = item;
		newNode->next = head;
		head = newNode;
		printf("--element is added at first--");
	}
}

void enqueuelast() {
	struct node *newNode,*temp;
	int item;
	newNode = (struct node*)malloc(sizeof(struct node));
	if(newNode == NULL) {
		printf("OVERFLOW");
	} else {
		printf("Enter value : ");
		scanf("%d",&item);
		newNode->data = item;
		if(head == NULL) {
		newNode -> next = NULL;
		head = newNode;
		printf("--element is added at last--");
	} else {
	temp = head;
	while (temp -> next != NULL) {
		temp = temp -> next;
	}
```

```
                temp->next = newNode;
                newNode->next = NULL;
                printf("--element is added at last--");
                }
        }
}

void dequeuefirst() {
        struct node *ptr;
        if(head == NULL) {
                printf("\nDeQue is empty\n");
        } else {
                ptr = head;
                head = ptr->next;
                free(ptr);
                printf("--element is deleted from first--");
        }
}

void dequeuelast() {
        struct node *ptr,*ptr1;
        if(head == NULL) {
                printf("DeQue is empty");
        } else if(head -> next == NULL) {
                head = NULL;
                free(head);
                printf("--element is deleted from last--");
        } else {
                ptr = head;
                while(ptr->next != NULL) {
                        ptr1 = ptr;
                        ptr = ptr ->next;
                }
                ptr1->next = NULL;
                free(ptr);
                printf("--element is deleted from last--");
        }
}

void display()  {
        int i;
        struct node *ptr;
        ptr=head;
        if(ptr == NULL)  {
                printf("Queue is empty\n");
        }  else  {
```

```
                printf("\n-- Queue --\n");
                while(ptr!=NULL) {
                        printf(" %d ", ptr->data);
                        ptr = ptr->next;
                }
        }
}

void main ()  {
        int choice=0;
        printf("--STACK USING LINKLIST--\n");
        printf("\n1.Add element at First\t\t2.Add element at Last\n3.Delete from
        First\t\t4.Delete from Last\n5.Display\t\t\t6.EXIT\n");
        while(choice != 6) {
                printf("\nEnter your choice : ");
                scanf("%d",&choice);
                switch(choice) {
                        case 1:
                                enqueuefirst();
                                break;
                        case 2:
                                enqueuelast();
                                break;
                        case 3:
                                dequeuefirst();
                                break;
                        case 4:
                                dequeuelast();
                                break;
                        case 5:
                                display();
                                break;
                        case 6:
                                printf("--EXIT--");
                                break;
                        default:
                                printf("Please Enter valid choice ");
                }
        }
}
```

OUTPUT :

1.Add element at First 2.Add element at Last
3.Delete from First 4.Delete from Last
5.Display 6.EXIT

Enter your choice : 1

Enter value : 10
--element is added at first--

Enter your choice : 2
Enter value : 20
--element is added at last--

Enter your choice : 1
Enter value : 50
--element is added at first--

Enter your choice : 5
-- Queue --
50 10 20

Enter your choice : 3
--element is deleted from first--

Enter your choice : 4
--element is deleted from last--

Enter your choice : 5
-- Queue --
10

Enter your choice : 6
EXIT

4) Write a c program to swap max and min element from singly linked list without swapping address.

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
   int data;
   struct Node *next;
} *head;
int getdata(){
   int item;
   printf("Enter Element : ");
   scanf("%d", &item);
   return item;
}

void Display(struct Node *head){
   struct Node *ptr;
   ptr = head;
   while (ptr != NULL){
      printf("%d -> ", ptr->data);
      ptr = ptr->next;
   } printf("\n");
}

void max_min_swap(){
   struct Node *max , *min , *ptr;
   ptr = head;
   max = head;
   min = head;
   while (ptr != NULL){
      if(ptr->data > max->data)
            max = ptr;
      if(ptr->data < min->data)
            min = ptr;
      ptr = ptr->next;
   }
   int temp;
   temp = max->data;
   max->data = min->data;
   min->data = temp;
   printf("After Swap : ");
   Display(head);
```

```
}
void create(){
   struct Node *ptr, *ptr1;
   int n;
   printf("Enter the Size of Linked list : ");
   scanf("%d", &n);
   head = (struct Node *)malloc(sizeof(struct Node));
   head->data = getdata();
   head->next = NULL;
   ptr = head;
   for (int i = 1; i <= n; i++) {
      if (i == n)
         ptr->next = NULL;
      else
      {
         ptr1 = (struct Node *)malloc(sizeof(struct Node));
         ptr->next = ptr1;
         ptr1->data = getdata();
         ptr1->next = NULL;
         ptr = ptr1; // ptr = ptr->next;
      }
   }
   printf("Before Swap : ");
   Display(head);
}


int main(){
   create();
   max_min_swap();
}
```

OUTPUT :

Enter the Size of Linked list : 5
Enter Element : 10
Enter Element : 20
Enter Element : 30
Enter Element : 45
Enter Element : 35
Before Swap : 10 -> 20 -> 30 -> 45 -> 35 ->
After Swap : 45 -> 20 -> 30 -> 10 -> 35 ->

5) Write a c program to remove duplicate values from singly Linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node *next;
} *first = NULL;
void Create(int A[], int n)
{
   int i;
   struct Node *t, *last;
   first = (struct Node *) malloc (sizeof (struct Node));
   first->data = A[0];
   first->next = NULL;
   last = first;
   for (i = 1; i < n; i++)
   {
      t = (struct Node *) malloc (sizeof (struct Node));
      t->data = A[i];
      t->next = NULL;
      last->next = t;
      last = t;
   }
}
void Display(struct Node *p)
{
   while (p != NULL)
   {
      printf ("%d ", p->data);
      p = p->next;
   }
}
void RemoveDuplicate(struct Node *p)
{
   struct Node *q = p->next;
   while (q != NULL)
   {
      if (p->data != q->data)
      {
         p = q;
         q = q->next;
```

```
        }
        else
       {
           p->next = q->next;
           free(q);
           q = p->next;
       }
    }
}
int main()
{
    int A[] = {1,2,2,5,5};
    Create (A, 5);
    printf ("Linked List with Duplicates: \n");
    Display (first);
    RemoveDuplicate (first);
    printf ("\nLinked List without Duplicates: \n");
    Display (first);
    return 0;
}
```

OUTPUT :

Linked List with Duplicates:

1 2 2 5 5

Linked List without Duplicates:

1 2 5

Question 1: Why and When should I use Stack or Queue data structures instead of Arrays/ Lists?

- Because they assist you in managing your data in a more specific manner than arrays and lists.
- We use stack or queue instead of arrays/lists when we want the elements in a specific order i.e. in the order we put them (queue) or in the reverse order (stack).
- Queues and stacks are dynamic while arrays are static. So when we require dynamic memory we use queue or stack over arrays.
- Stacks and queues are used over arrays when sequential access is required.
- To efficiently remove any data from the start (queue) or the end (stack) of a data structure.
- When you want to get items out in the same order that you put them in, use a queue (FIFO)
- When you need to bring things out in the opposite order that they were put in, use a stack (LIFO).

Question 2: Compare Array based vs Linked List stack implementations.

S.No.	Array based stack	Linked list based stack
1.	A singly-linked list supports O(1) time prepend and delete-first, the cost to push or pop into a linked-list-backed stack is also O(1) worst-case.	Enqueuing into the linked list can be implemented by appending to the back of the singly-linked list, which takes worst-case time O(1).
2.	However, each new element added requires a new allocation, and allocations can be expensive compared to other operations.	Dequeuing can be implemented by removing the first element, which also takes worst-case time O(1).
3.	In the case of an array, memory size is fixed, and it is not possible to change it during the run time.However, the constant factors in these O(1) terms may be high due to the expense of dynamic allocations.	There are techniques to try to reduce this by allocating extra space and lazily copying the elements over.
4.	The memory overhead of the linked list is usually O(n) total extra memory due to the storage of an extra pointer in each element.	Allocations are infrequent and accesses are fast, dynamic arrays are usually faster than linked lists.

1) Write a C program to create a Binary Search Tree (Insertion, deletion and traversal).

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

struct node
{
   int info;
   struct node *lchild;
   struct node *rchild;
}*root;
void 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(item<root->info)
      ptr=root->lchild;
   else
      ptr=root->rchild;
   ptrsave=root;

   while(ptr!=NULL)
   {
      if(item==ptr->info)
      {      *loc=ptr;
         *par=ptrsave;
         return;
      }
      ptrsave=ptr;
      if(item<ptr->info)
         ptr=ptr->lchild;
      else
         ptr=ptr->rchild;
    }/*End of while */
```

```
    *loc=NULL;   /*item not found*/
    *par=ptrsave;
}/*End of find()*/

void 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(item<parent->info)
         parent->lchild=tmp;
      else
         parent->rchild=tmp;
}/*End of insert()*/


void 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()*/

void 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()*/

void 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()*/
int del(int item)
{
    struct node *parent,*location;
    if(root==NULL)
    {
```

```
        printf("Tree empty");
        return 0;
    }

    find(item,&parent,&location);
    if(location==NULL)
    {
        printf("Item not present in tree");
        return 0;
    }

    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()*/
void 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()*/
void main()
{
    int choice,num;
    root=NULL;
    printf("\n");
    printf("1.Insert\n");
    printf("2.Delete\n");
    printf("3.Display\n");
    printf("4.Quit\n");
    while(1)
    {

        printf("\nEnter 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:
           break;
         default:
           printf("Wrong choice\n");
        }/*End of switch */
      }/*End of while */
}/*End of main()*/
```

OUTPUT :

1.Insert

2.Delete

3.Display

4.Quit

Enter your choice : 1

Enter the number to be inserted : 30

Enter your choice : 1

Enter the number to be inserted : 20

Enter your choice : 1

Enter the number to be inserted : 10

Enter your choice : 3

30

20

10

Enter your choice : 2

Enter the number to be deleted : 20

Enter your choice : 3

30

10

Enter your choice : 4

2) Write a C program to implement tree traversing methods inorder, preorder and post-order traversal.

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

// structure of a node
struct node
{
   int data;
   struct node *left;
   struct node *right;
};

// globally initialized root pointer
struct node *root = NULL;

// function prototyping
struct node *create_node(int);
void insert(int);
struct node *delete (struct node *, int);
int search(int);
void inorder(struct node *);
void postorder();
void preorder();
struct node *smallest_node(struct node *);
struct node *largest_node(struct node *);
int get_data();

int main()
{
   int userChoice;
   int userActive = 'Y';
   int data;
   struct node* result = NULL;

   while (userActive == 'Y' || userActive == 'y')
   {
      printf("\n\n------- Binary Search Tree ------\n");
      printf("\n1. Insert");
      printf("\n2. Delete");
      printf("\n3. Search");
      printf("\n4. Get Larger Node Data");
      printf("\n5. Get smaller Node data");
      printf("\n\n-- Traversals --");
      printf("\n\n6. Inorder ");
      printf("\n7. Post Order ");
```

```
printf("\n8. Pre Oder ");
printf("\n9. Exit");

printf("\n\nEnter Your Choice: ");
scanf("%d", &userChoice);
printf("\n");

switch(userChoice)
{
   case 1:
      data = get_data();
      insert(data);
      break;

   case 2:
      data = get_data();
      root = delete(root, data);
      break;

   case 3:
      data = get_data();
      if (search(data) == 1)
      {
         printf("\nData was found!\n");
      }
      else
      {
         printf("\nData does not found!\n");
      }
      break;

   case 4:
      result = largest_node(root);
      if (result != NULL)
      {
         printf("\nLargest Data: %d\n", result->data);
      }
      break;

   case 5:
      result = smallest_node(root);
      if (result != NULL)
      {
         printf("\nSmallest Data: %d\n", result->data);
      }
      break;
```

```
            case 6:
                inorder(root);
                break;

            case 7:
                postorder(root);
                break;

            case 8:
                preorder(root);
                break;

            case 9:
                printf("\n\nProgram was terminated\n");
                break;

            default:
                printf("\n\tInvalid Choice\n");
                break;
        }

        printf("\n__________\nDo you want to continue?Press Y or y: ");
        fflush(stdin);
        scanf(" %c", &userActive);
    }

    return 0;
}

// creates a new node
struct node *create_node(int data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));

    if (new_node == NULL)
    {
        printf("\nMemory for new node can't be allocated");
        return NULL;
    }

    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;

    return new_node;
}
```

```
// inserts the data in the BST
void insert(int data)
{
   struct node *new_node = create_node(data);

   if (new_node != NULL)
   {
      // if the root is empty then make a new node as the root node
      if (root == NULL)
      {
         root = new_node;
         printf("\n* node having data %d was inserted\n", data);
         return;
      }

      struct node *temp = root;
      struct node *prev = NULL;

      // traverse through the BST to get the correct position for insertion
      while (temp != NULL)
      {
         prev = temp;
         if (data > temp->data)
         {
            temp = temp->right;
         }
         else
         {
            temp = temp->left;
         }
      }

      // found the last node where the new node should insert
      if (data > prev->data)
      {
         prev->right = new_node;
      }
      else
      {
         prev->left = new_node;
      }

      printf("\n* node having data %d was inserted\n", data);
   }
}

// deletes the given key node from the BST
```

```
struct node *delete (struct node *root, int key)
{
   if (root == NULL)
   {
      return root;
   }
   if (key < root->data)
   {
      root->left = delete (root->left, key);
   }
   else if (key > root->data)
   {
      root->right = delete (root->right, key);
   }
   else
   {
      if (root->left == NULL)
      {
         struct node *temp = root->right;
         free(root);
         return temp;
      }
      else if (root->right == NULL)
      {
         struct node *temp = root->left;
         free(root);
         return temp;
      }
      struct node *temp = smallest_node(root->right);
      root->data = temp->data;
      root->right = delete (root->right, temp->data);
   }
   return root;

}

// search the given key node in BST
int search(int key)
{
   struct node *temp = root;

   while (temp != NULL)
   {
      if (key == temp->data)
      {
         return 1;
      }
```

```
        else if (key > temp->data)
        {
            temp = temp->right;
        }
        else
        {
            temp = temp->left;
        }
    }
    return 0;
}

// finds the node with the smallest value in BST
struct node *smallest_node(struct node *root)
{
    struct node *curr = root;
    while (curr != NULL && curr->left != NULL)
    {
        curr = curr->left;
    }
    return curr;
}

// finds the node with the largest value in BST
struct node *largest_node(struct node *root)
{
    struct node *curr = root;
    while (curr != NULL && curr->right != NULL)
    {
        curr = curr->right;
    }
    return curr;
}

// inorder traversal of the BST
void inorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    inorder(root->left);
    printf("%d ",  root->data);
    inorder(root->right);
}

// preorder traversal of the BST
```

```
void preorder(struct node *root)
{
   if (root == NULL)
   {
      return;
   }
   printf("%d ", root->data);
   preorder(root->left);
   preorder(root->right);
}

// postorder travsersal of the BST
void postorder(struct node *root)
{
   if (root == NULL)
   {
      return;
   }
   postorder(root->left);
   postorder(root->right);
   printf("%d ", root->data);
}

// getting data from the user
int get_data()
{
   int data;
   printf("\nEnter Data: ");
   scanf("%d", &data);
   return data;
}
```

OUTPUT :

------- Binary Search Tree ------

1. Insert

2. Delete

3. Search

4. Get Larger Node Data

5. Get smaller Node data

-- Traversals --

6. Inorder

7. Post Order

8. Pre Oder

9. Exit

Enter Your Choice: 1

Enter Data: 40

*** node having data 40 was inserted**

_____

Do you want to continue?Press Y or y: y

Enter Your Choice: 1

Enter Data: 30

*** node having data 30 was inserted**

_____

Do you want to continue?Press Y or y: y

Enter Your Choice: 1

Enter Data: 50

*** node having data 50 was inserted**

_____

Do you want to continue?Press Y or y: y

Enter Your Choice: 1

Enter Data: 20

*** node having data 20 was inserted**

_____

Do you want to continue?Press Y or y: y

Enter Your Choice: 1

Enter Data: 5

*** node having data 5 was inserted**

_____

Do you want to continue?Press Y or y: y

Enter Your Choice: 1

Enter Data: 3

*** node having data 3 was inserted**

_____

Do you want to continue?Press Y or y: y

Enter Your Choice: 1

Enter Data: 70

*** node having data 70 was inserted**

_____

Do you want to continue?Press Y or y: y

Enter Your Choice: 6

3 5 20 30 40 50 70

_____

Do you want to continue?Press Y or y: y

Enter Your Choice: 7

3 5 20 30 70 50 40

_____

Do you want to continue?Press Y or y: y

Enter Your Choice: 8

40 30 20 5 3 50 70

_____

Do you want to continue?Press Y or y: n

Question 1: Which traversal method is most preferred in terms of performance? Explain with reason.

- Inorder Traversal is the one the most used variant of DFS(Depth First Search) Traversal of the tree. Because In-order traversal is very commonly used in binary search trees because it returns values from the underlying set in order, according to the comparator that set up the binary search tree (hence the name). Post-order traversal while deleting or freeing nodes and values can delete or free an entire binary tree.

Question 2: What is the use of In-order, Pre-order and Post-order tree traversal?

- Pre-order traversal while duplicating nodes and values can make a complete duplicate of a binary tree. It can also be used to make a prefix expression (Polish notation) from expression trees: traverse the expression tree pre-orderly.
- In-order traversal is very commonly used on binary search trees because it returns values from the underlying set in order, according to the comparator that set up the binary search tree (hence the name).
- Post-order traversal while deleting or freeing nodes and values can delete or free an entire binary tree. It can also generate a postfix representation of a binary tree.]

Question 3: How can we implement preorder, inorder and post order tree traversals without using recursion?

Using stack is the obvious way to traverse tree without recursion. Below is an algorithm for traversing binary tree using stack.

1) Create an empty stack S.
2) Initialize current node as root
3) Push the current node to S and set current = current->left until current is NULL
4) If current is NULL and stack is not empty then
    a. Pop the top item from stack.
    b. Print the popped item, set current = popped_item->right
    c. Go to step 3.
5) If current is NULL and stack is empty then we are done.

1) Write a c program to implement DFS graph traversal.

```
#include <stdio.h>
#include <stdlib.h>
int vis[100];
struct Graph {
   int V;
   int E;
   int** Adj;
};
struct Graph* adjMatrix()
{
   struct Graph* G = (struct Graph*)
      malloc(sizeof(struct Graph));
   if (!G) {
      printf("Memory Error\n");
      return NULL;
   }
   G->V = 7;
   G->E = 7;

   G->Adj = (int**)malloc((G->V) * sizeof(int*));
   for (int k = 0; k < G->V; k++) {
      G->Adj[k] = (int*)malloc((G->V) * sizeof(int));
   }

   for (int u = 0; u < G->V; u++) {
      for (int v = 0; v < G->V; v++) {
         G->Adj[u][v] = 0;
      }
   }
   G->Adj[0][1] = G->Adj[1][0] = 1;
   G->Adj[0][2] = G->Adj[2][0] = 1;
   G->Adj[1][3] = G->Adj[3][1] = 1;
   G->Adj[1][4] = G->Adj[4][1] = 1;
   G->Adj[1][5] = G->Adj[5][1] = 1;
   G->Adj[1][6] = G->Adj[6][1] = 1;
   G->Adj[6][2] = G->Adj[2][6] = 1;

   return G;
}
void DFS(struct Graph* G, int u)
{
   vis[u] = 1;
   printf("%d ", u);
   for (int v = 0; v < G->V; v++) {
      if (!vis[v] && G->Adj[u][v]) {
```

```
            DFS(G, v);
        }
    }
}

// Function for DFS traversal
void DFStraversal(struct Graph* G)
{
    for (int i = 0; i < 100; i++) {
        vis[i] = 0;
    }
    for (int i = 0; i < G->V; i++) {
        if (!vis[i]) {
            DFS(G, i);
        }
    }
}
void main()
{
    struct Graph* G;
    printf("\n------------Simple Depth First Traversal----------\n");
    G = adjMatrix();
    DFStraversal(G);
}
```

OUTPUT:

------------Simple Depth First Traversal----------

0 1 3 4 5 6 2

2) Write a c program to implement BFS graph traversal.

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
   for (i=1;i<=n;i++)
    if(a[v][i] && !visited[i])
     q[++r]=i;
   if(f<=r) {
      visited[q[f]]=1;
      bfs(q[f++]);
   }
}
void main()
{
   int v;
   printf("\n Enter the number of vertices:");
   scanf("%d",&n);
   for (i=1;i<=n;i++)
   {
      q[i]=0;
      visited[i]=0;
   }
   printf("\n Enter graph data in matrix form:\n");
   for (i=1;i<=n;i++)
    for (j=1;j<=n;j++)
     scanf("%d",&a[i][j]);
   printf("\n Enter the starting vertex:");
   scanf("%d",&v);
   bfs(v);
   printf("\n The node which are reachable are:\n");
   for (i=1;i<=n;i++)
    if(visited[i])
     printf("%d\t",i); else
     printf("\n Bfs is not possible");
   getch();
}
```

Output:

Enter the number of vertices:3

Enter graph data in matrix form:

0 1 1

1 0 1

1 1 0

Enter the starting vertex:1

The node which are reachable are:

1 2 3

Question 1: Real time applications of DFS and BFS (At least 5 applications of each one).

Applications of Breadth First Traversal

1. **Shortest Path and Minimum Spanning Tree for unweighted graph :** In an unweighted graph, the shortest path is the path with least number of edges.
2. **Peer to Peer Networks. I**n Peer to Peer Networks like BitTorrent, Breadth First Search is used to find all neighbor nodes.
3. **Social Networking Websites :** In social networks, we can find people within a given distance 'k' from a person using Breadth First Search till 'k' levels.
4. **GPS Navigation systems:** Breadth First Search is used to find all neighboring locations.
5. **Broadcasting in Network :** In networks, a broadcasted packet follows Breadth First Search to reach all nodes.

Applications of Depth First Search

1. **Detecting cycle in a graph :** A graph has cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.
2. **Topological Sorting :** Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when recomputing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in makefiles, data serialization, and resolving symbol dependencies in linkers.
3. **To test if a graph is bipartite :**
   We can augment either BFS or DFS when we first discover a new vertex, color it opposite its parents, and for each other edge, check it doesn't link two vertices of the same color. The first vertex in any connected component can be red or black.
4. **Finding Strongly Connected Components of a graph :** A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex.
5. **Solving puzzles with only one solution**, such as mazes. (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set.)

Question 2: Difference between DFS and BFS. (At least 5 points of each one)

BFS vs DFS

S.No.	Parameters	BFS	DFS
1.	Stands for	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
2.	Data Structure	BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
3.	Definition	BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to the next level.	DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes.
4.	Technique	BFS can be used to find a single source shortest path in an unweighted graph because, in BFS, we reach a vertex with a minimum number of edges from a source vertex.	In DFS, we might traverse through more edges to reach a destination vertex from a source.
5.	Conceptual Difference	BFS builds the tree level by level.	DFS builds the tree sub-tree by sub-tree.
6.	Approach used	It works on the concept of FIFO (First In First Out).	It works on the concept of LIFO (Last In First Out).
7.	Suitable for	BFS is more suitable for searching vertices closer to the given source.	DFS is more suitable when there are solutions away from source.