Bachelor of Engineering

First Year

II SEMESTER

CS6212 – PROGRAMMING AND DATA STRUCTURES LABORATORY – I

LAB MANUAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS6212 – PROGRAMMING AND DATA STRUCTURES LABORATORY - I

L T P C 0 0 3 2

OBJECTIVES:

• To introduce the concepts of structured Programming language.
• To introduce the concepts of pointers and files
• To introduce the concepts of primitive Data Structures.

1. C Programs using Conditional and Control Statements
2. C Programs using Arrays, Strings and Pointers and Functions
3. Representation of records using Structures in C – Creation of Linked List – Manipulation of records in a Linked List
4. File Handling in C – Sequential access – Random Access
5. Operations on a Stack and Queue – infix to postfix – simple expression evaluation using stacks - Linked Stack Implementation – Linked Queue Implementation
6. Implementation of Sorting algorithms
7. Implementation of Linear search and Binary Search.

TOTAL: 45 PERIODS

Ex. No.: 1.a	ROOTS OF QUADRATIC EQUATIONS

Aim:

To write a C program for finding the roots of a given quadratic equation using conditional control statements.

Algorithm:

1. Start the program.
2. Declare the variables and read the coefficients and constants.
3. Find the determinant.
4. If it is greater than zero, print the roots are real and distinct and find the roots.
5. If it is less than zero, print the roots are imaginary.
6. If it is equal to zero, print the roots are real and equal and find the roots.
7. Stop the program.

Program:

```
/*Roots of a Quadratic Equation*/
#include <stdio.h>
#include <math.h>
int main()
{
 float a, b, c, determinant, r1,r2, real, imag;
 printf("Enter coefficients a, b and c: ");
 scanf("%f%f%f",&a,&b,&c);
 determinant=b*b-4*a*c;
 if (determinant>0)
 {
    r1= (-b+sqrt(determinant))/(2*a);
    r2= (-b-sqrt(determinant))/(2*a);
    printf("Roots are: %.2f and %.2f",r1 , r2);
 }
 else if (determinant==0)
 {
  r1 = r2 = -b/(2*a);
  printf("Roots are: %.2f and %.2f", r1, r2);
 }
 else
 {
  real= -b/(2*a);
  imag = sqrt(-determinant)/(2*a);
  printf("Roots are: %.2f+%.2fi and %.2f-%.2fi", real, imag, real, imag);
 }
 return 0;
}
```

Output:

```
Enter coefficients a, b and c:
2.3
4
5.6
Roots are: -0.87+1.30i and -0.87-1.30i
```

Ex. No.: 1.b	ARMSTRONG NUMBER

Aim:

To write a C program for finding whether a given number is Armstrong number or not using loop control statements.

Algorithm:

1. Start the program.
2. Take number input from user store it in 'number'.
3. Store that number into 'temp'
   temp:=number;
4. while temp!=0 do,
   remainder:=temp%10;
   sum:=sum+remainder*remainder*remainder;
   temp:=temp/10;
   End while
5. if number==sum then,
   print number is armstrong
   else
   print number is not armstrong
6. Stop the program

Program:

```
/*Armstrong Number*/
#include <stdio.h>
int main()
{
 int n, n1, rem, num=0;
 printf("Enter a positive  integer: ");
 scanf("%d", &n);
 n1=n;
 while(n1!=0)
 {
    rem=n1%10;
    num+=rem*rem*rem;
    n1/=10;
 }
 if(num==n)
   printf("%d is an Armstrong number.",n);
 else
   printf("%d is not an Armstrong number.",n);
}
```

Output:

```
Enter a positive  integer: 371
371 is an Armstrong number.
```

Ex. No.: 2.a	MATRIX MULTIPLICATION

Aim:

To write a C program to compute matrix multiplication using the concept of arrays.

Algorithm:

1. Start the program.
2. To multiply two matrixes sufficient and necessary condition is "number of columns in matrix A = number of rows in matrix B".
3. Loop for each row in matrix A.
4. Loop for each columns in matrix B and initialize output matrix C to 0.
5. This loop will run for each rows of matrix A.
6. Loop for each columns in matrix A.
7. Multiply A[i,k] to B[k,j] and add this value to C[i,j]
8. Return output matrix C.
9. Stop the program.

Program:

```
/*Matrix Multiplication*/
#include<stdio.h>
#include<conio.h>
int main()
{
    int a[10][10], b[10][10], c[10][10],m,n, i, j, k, l,g;
    printf("Enter the number of rows and column of first matrix\n");
    scanf("%d%d",&m,&n);
    printf("\nEnter the elements of first %dx%d matrix\n",m,n);
    for (i=0; i< m; i++)
    {
       for(j=0; j<n; j++)
       {
              scanf("%d",&a[i][j]);
       }
    }
    s1 : printf("\nEnter the number of rows and column of second matrix\n");
    scanf("%d%d",&g,&l);
    if(n!=g)
    {
          printf("\nIn matrix multiplication first column and second row number should be the
same \nRenter ");
          getch();
          goto s1;
    }
    printf("\nEnter the elements of second %dx%d matrix",n,l);
    for(i = 0; i <n; i++)
    {
        for (j = 0; j < l; j++)
        {
        scanf("%d", &b[i][j]);
        }
    }
    printf("\nThe first matrix is :-\n");
    for (i = 0; i < m; i++)
    {
         for (j = 0; j < n; j++)
         {
         printf("\t%d", a[i][j]);
         }
    printf("\n");
    }
    printf("\nThe second matrix is :-\n");
    for (i = 0; i < n; i++)
    {
```

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

    printf("\nMultiplication of the two matrices is as follows:\n");
    for (i = 0;i < m; i++)
    {
        printf("\n");
        for (j = 0; j < l; j++)
        {
            c[i][j]=0;
            for(k=0;k<n;k++)
                c[i][j] = c[i][j]+a[i][k] * b[k][j];
            printf("\t%d", c[i][j]);
        }
    }
    getch();
}
```

Output:

```
Enter the number of rows and column of first matrix
2
2

Enter the elements of first 2x2 matrix
3
2
3
2

Enter the number of rows and column of second matrix
2
2

Enter the elements of second 2x2 matrix4
1
4
1

The first matrix is :-
    3    2
    3    2

The second matrix is :-
    4    1
    4    1

Multiplication of the two matrices is as follows:

    20    5
    20    5
```

Ex. No.: 2.b	STRING

Aim:

To write a C program to illustrate the concept of string handling functions.

Algorithm:

1. Start the program.
2. Get two strings as input.
3. Use the string compare function for checking the two strings are equal.
4. Use the string concatenation function for joining first string with second string.
5. Find the length of the strings using the strlen function.
6. Print the reverse of a string using strrev function.
7. Show the upper case and lower case of a string using strupr and strlwr functions.
8. Stop the program.

Program:

```
/*String Handling*/
 #include    <string.h>
   void main()
   {   char  s1[20], s2[20], s3[20];
       int   x, l1, l2, l3;
       clrscr();
       printf("\n\nEnter two string constants: \n");
       scanf("%s %s", s1, s2);
       x = strcmp(s1, s2);
       if(x != 0)
       {  printf("\n\nStrings are not equal \n");
          strcat(s1, s2);
       }
       else
          printf("\n\nStrings are equal \n");
       strcpy(s3, s1);
       l1 = strlen(s1);
       l2 = strlen(s2);
       l3 = strlen(s3);
       printf("\ns1 = %s\t length = %d characters\n", s1, l1);
       printf("s2 = %s\t length = %d characters\n", s2, l2);
       printf("s3 = %s\t length = %d characters\n", s3, l3);
       printf("Reverse of S1 is %s\n",strrev(s1));
       printf("Uppercase of S2 is %s\n",strupr(s2));
       printf("Lowercase of S2 is %s\n",strlwr(s2));
   }
```

Output:

```
Enter two string constants:
Sachein
Ashwath

Strings are not equal

s1 = SacheinAshwath      length = 14 characters
s2 = Ashwath     length = 7 characters
s3 = SacheinAshwath      length = 14 characters
Reverse of S1 is htawhsAniehcaS
Uppercase of S2 is ASHWATH
Lowercase of S2 is ashwath
```

Ex. No.: 3.a	LARGEST ELEMENT IN ARRAY

Aim:

To write a C program to find the largest element in an array using pointers.

Algorithm:

1. Create an initial array with some elements in it.
2. Allocate memory for the elements using calloc function.
3. Check first element is greater than the second element of the array using pointers and so on.
4. Print the largest element.
5. Stop the program.

Program:

```
/*Largest Element in Array*/
#include <stdio.h>
#include <stdlib.h>
int main(){
   int i,n;
   float *data;
   printf("Enter total number of elements(1 to 100): ");
   scanf("%d",&n);
   data=(float*)calloc(n,sizeof(float));
   if(data==NULL)
   {
      printf("Error!!! memory not allocated.");
      exit(0);
   }
   printf("\n");
   for(i=0;i<n;++i)
   {
     printf("Enter Number %d: ",i+1);
     scanf("%f",data+i);
   }
   for(i=1;i<n;++i)
   {
     if(*data<*(data+i))
        *data=*(data+i);
   }
   printf("Largest element = %.2f",*data);
   return 0;
}
```

Output:

```
Enter the total number of elements(1 to 100): 5

Enter Number 1: 10.5
Enter Number 2: 12.6
Enter Number 3: 39.78
Enter Number 4: 9.6
Enter Number 5: 42.5
Largest element = 42.50
```

Ex. No.: 3.b	NUMBER CONVERSION

Aim:

To write a C program to a binary number to decimal or decimal to binary using functions.

Algorithm:

1. Start the program.
2. Accept Number from User
3. Divide Number by 10 and Store Remainder in variable rem
4. Divide Original Number by 10.
5. Inside the First Iteration power = 0.
6. Power is Incremented in each iteration.
7. Calculate sum using the above formula, calculated sum is nothing but the decimal representation of the given binary number.
8. Stop the program.

Program:

```
/*Number Conversion*/
#include <stdio.h>
#include <math.h>
int binary_decimal(int n);
int decimal_binary(int n);
int main()
{
  int n;
  char c;
  printf("Instructions:\n");
  printf("1. Enter alphabet 'd' to convert binary to decimal.\n");
  printf("2. Enter alphabet 'b' to convert decimal to binary.\n");
  scanf("%c",&c);
  if (c =='d' || c == 'D')
  {
     printf("Enter a binary number: ");
     scanf("%d", &n);
     printf("%d in binary = %d in decimal", n, binary_decimal(n));
  }
  if (c =='b' || c == 'B')
  {
     printf("Enter a decimal number: ");
     scanf("%d", &n);
     printf("%d in decimal = %d in binary", n, decimal_binary(n));
  }
  return 0;
}
int decimal_binary(int n)
{
   int rem, i=1, binary=0;
   while (n!=0)
   {
      rem=n%2;
      n/=2;
      binary+=rem*i;
      i*=10;
   }
   return binary;
}
int binary_decimal(int n)
{
   int decimal=0, i=0, rem;
   while (n!=0)
   {
      rem = n%10;
```

```
        n/=10;
        decimal += rem*pow(2,i);
        ++i;
    }
    return decimal;
}
```

Output:

```
Instructions:
1. Enter alphabet 'd' to convert binary to decimal.
2. Enter alphabet 'b' to convert decimal to binary.
b
Enter a decimal number: 8
8 in decimal = 1000 in binary
```

Ex. No.: 4	FILE HANDLING

Aim:

To write a C program to read data from keyboard, write it to a file named student again read the same data from student file and write it into data file.

Algorithm:

1. Start the program.
2. Declare the file pointers.
3. Allocate the memory dynamically.
4. Store the roll number and name in a file.
5. Display them from the file.
6. Stop the program.

Program:

```
/*File Handling*/
#include<stdio.h>
#include<conio.h>
#include<process.h>
struct stud
{
 int rno;
 char *nm;
};
void main()
{
 struct stud *s;
 int n,i;
 FILE *fp,*fp1;
 clrscr();
 printf("Enter how many record you want to input : ");
 scanf("%d",&n);
 s=(struct stud *)malloc(n*sizeof(struct stud));
 fp=fopen("SACHEIN.TXT","w");
 for(i=0;i<n;i++)
 {
  printf("\n\tInformation for student : %d\n",i+1);
  printf("Enter Roll No : ");
  scanf("%d",&s[i].rno);
  printf("Enter Name : ");
  fflush(stdin);
  gets(s[i].nm);
  fprintf(fp,"%5d  %-20s\n",s[i].rno,s[i].nm);
 }
 fclose(fp);
 fp=fopen("SACHEIN.TXT","r");
 fp1=fopen("ASHWATH.TXT","w");
 printf("\nContent of the SACHEIN.TXT file is\n");
 printf("Roll No   Name\n");
 printf("-------------------------\n");
 while(!feof(fp))
 {
  fscanf(fp,"%5d  %20s\n",&s[i].rno,s[i].nm);
  fprintf(fp1,"%5d  %-20s\n",s[i].rno,s[i].nm);
  printf("%7d   %-20s\n",s[i].rno,s[i].nm);       }
 fcloseall();
 getch();
}
```

Output:

Enter how many record you want to input :

2

Information for student : 1

Enter Roll No : 7

Enter Name : Ashwath

Information for student : 2

Enter Roll No : 10

Enter Name : Sachein

Content of the SACHEIN.TXT file is

Roll No Name

---

7 Ashwath

10 Sachein

Ex. No.: 5	EMPLOYEE DETAILS

Aim:

To write a C program to store employee details using the concept of structures.

Algorithm:

1. Start the program.
2. Create a structure details and declare its members.
3. Get the values for the structure members as input.
4. Display them in output.
5. Stop the program.

Program:

```
#include <stdio.h>
#include <conio.h>
struct details
{
 char name[30];
 int age;
 char address[500];
 float salary;
};
int main()
{
 struct details detail;
 clrscr();
 printf("\nEnter name:\n");
 scanf("%s",detail.name);
 printf("\nEnter age:\n");
 scanf("%d",&detail.age);
 printf("\nEnter Address:\n");
 scanf("%s",detail.address);
 printf("\nEnter Salary:\n");
 scanf("%f",&detail.salary);
printf("\n\n\n");
 printf("Name of the Employee : %s \n",detail.name);
 printf("Age of the Employee : %d \n",detail.age);
 printf("Address of the Employee : %s \n",detail.address);
 printf("Salary of the Employee : %f \n",detail.salary);
 getch();
}
```

Output:

```
Enter name:
Sachein
Enter age:
21
Enter Address:
Perambalur
Enter Salary:
28000

Name of the Employee : Sachein
Age of the Employee : 21
Address of the Employee : Ariyalur
Salary of the Employee : 28000.00
```

Ex. No.: 6	LINKED LIST IMPLEMENTATION

Aim:

To write a C program to illustrate the concept of linked list.

Algorithm:

A) Insertion:

1. Declare a new node and allocate memory for that node.
2. Get the data to be inserted and make the reference field of the node to NULL.
3. If first node is NULL, assign new node to first node; else, assign first to old node and make the reference field of old node to point to new node.

B) Deletion:

1. Get the element to be deleted and assign first node to old node.
2. If old node is NULL, then return element not found; else, assign the value of reference field of old to the reference field of temporary node and free the old node.

Program:

```
/* Linked List Implementation*/
#include<stdio.h>
#include<stdlib.h>
typedef struct Node
{
        int data;
        struct Node *next;
}node;
void insert(node *p, int data)
{
       while(p->next!=NULL)
        {
                p = p -> next;
        }
        p->next = (node *)malloc(sizeof(node));
        p = p->next;
        p->data = data;
        p->next = NULL;
}
int find(node *p, int key)
{
           p =  p -> next;
           while(p!=NULL)
           {
                    if(p->data == key)
                    {
                             return 1;
                    }
                    p = p -> next;
           }
           return 0;
}
void delete(node *p, int data)
{
           node *temp;
           while(p->next!=NULL && (p->next)->data != data)
           {
                    p = p -> next;
           }
           if(p->next==NULL)
           {
                    printf("Element %d is not present in the list\n",data);
                    return;
           }
           temp = p -> next;
        p->next = temp->next;
```

```
        free(temp);
        return;
}
void print(node *p)
{
        if(p==NULL)
        {
                return;
        }
        printf("%d ",p->data);
           print(p->next);
}
int main()
{
           node *start,*temp;
           int ch,data,status;
           clrscr();
           start = (node *)malloc(sizeof(node));
           temp = start;
           temp -> next = NULL;
           printf("<-----Linked List Implementation----->\n");
           while(1)
           {
                   printf("1. Insert\n");
                   printf("2. Delete\n");
                   printf("3. Print\n");
                   printf("4. Find\n");
                   printf("5. Exit\n");
                   printf("Enter Your Choice:\n");
              scanf("%d",&ch);
              switch(ch)
                   {
                   case 1:
                                   printf("\nEnter Data to Insert:\n");
                           scanf("%d",&data);
                           insert(start,data);
                           break;
                           case 2:
                                   printf("\nEnter Data to Delete:\n");
                                   scanf("%d",&data);
                                   delete(start,data);
                                   break;
                           case 3:
                                   printf("The list is ");
                                   print(start->next);
                                   printf("\n");
                                   break;
```

```
                case 4:
                        printf("\nEnter Data to Find:\n");
                        scanf("%d",&data);
                        status = find(start,data);
                        if(status)
                            printf("Element Found\n");
                        else
                            printf("Element Not Found\n");
                        break;
                case 5:
                        exit(0);
                default:
                        printf("Wrong Choice!\n");

                }
        }
}
```

Output:

<-----Linked List Implementation----->

1. Insert

2. Delete
3. Print
4. Find
5. Exit
Enter Your Choice:
1
Enter Data to Insert:
10
1. Insert
2. Delete
3. Print
4. Find
5. Exit
Enter Your Choice:
1
Enter Data to Insert:
20
1. Insert
2. Delete
3. Print
4. Find
5. Exit
Enter Your Choice:
3
The list is 10 20
1. Insert
2. Delete
3. Print
4. Find
5. Exit
Enter Your Choice:
2
Enter Data to Delete:
20
1. Insert
2. Delete
3. Print
4. Find
5. Exit
Enter Your Choice:
4
Enter Data to Find:
20
Element Not Found
1. Insert
2. Delete
3. Print

4. Find
5. Exit
Enter Your Choice:
5

Ex. No.: 7	STACK – ARRAY IMPLEMENTATION

Aim:

To write a C program to implement stack using the concept of arrays.

Algorithm:

A) Push Operation:

1. To push an element into the stack, check whether the top of the stack is greater than or equal to the maximum size of the stack.
2. If so, then return stack is full and element cannot be pushed into the stack.
3. Else, increment the top by one and push the new element in the new position of top.

B) Pop Operation:

1. To pop an element from the stack, check whether the top of the stack is equal to -1.
2. If so, then return stack is empty and element cannot be popped from the stack.
3. Else, decrement the top by one.

Program:

```
/*Stack Implementation using arrays*/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 5
struct stack  {
        int s[size];
        int top;
         }st;
int stfull()
{
 if(st.top>=size-1)
    return 1;
 else
    return 0;
}
void push(int item)
{
 st.top++;
 st.s[st.top] =item;
}
int stempty()
{
 if(st.top==-1)
    return 1;
 else
    return 0;
}
int pop()
 {
 int item;
 item=st.s[st.top];
 st.top--;
 return(item);
 }
void display()
{
 int i;
 if(stempty())
    printf("Stack Is Empty!\n");
 else
  {
  for(i=st.top;i>=0;i--)
     printf("\n%d",st.s[i]);
  }
}
```

```
void main(void)
{
int item,ch;
char ans;
st.top=-1;
clrscr();
printf("<-----Stack using Array----->\n");
while(1)
{
 printf("\n1.Push\n2.Pop\n3.Display\n4.exit\n");
 printf("Enter Your Choice:\n");
 scanf("%d",&ch);
 switch(ch)
 {
  case 1:
       printf("Enter The item to be pushed:\n");
       scanf("%d",&item);
       if(stfull())
          printf("Stack is Full!\n");
       else
          push(item);
       break;
  case 2:
       if(stempty())
          printf("Empty stack!\n");
       else
         {
         item=pop();
         printf("The popped element is %d\n",item);
         }
       break;
  case 3:
       display();
       break;
  case 4:
       exit(0);
 }
}
getch();
}
```

Output:

<-----Stack using Array----->

1.Push
2.Pop
3.Display
4.exit
Enter Your Choice:
1
Enter The item to be pushed:
10

1.Push
2.Pop
3.Display
4.exit
Enter Your Choice:
1
Enter The item to be pushed:
20

1.Push
2.Pop
3.Display
4.exit
Enter Your Choice:
3

20
10

1.Push
2.Pop
3.Display
4.exit
Enter Your Choice:
2
The popped element is 20

1.Push
2.Pop
3.Display
4.exit
Enter Your Choice:
4

Ex. No.: 8	STACK – LINKED LIST IMPLEMENTATION

Aim:

To write a C program to implement stack using the concept of linked list.

Algorithm:

A) Push Operation:

1. To push an element into the stack, copy the element to be inserted in the data field of the new node.
2. Assign the reference field of the new node as NULL.
3. Check if top is not NULL, if so, then assign the value of top in the reference field of new node.
4. Assign the address of the new node to the top.

B) Pop Operation:

1. To pop an element from the stack, check whether the top of the stack is NULL.
2. If so, then return stack is empty and element cannot be popped from the stack.
3. Else, assign the top value to a temporary node.
4. Now assign the value in the reference field of the node pointed by top to the top value.
5. Return the value in the data field of the temporary node as the element deleted and delete the temporary node.

Program:

```
/*Stack Implementation using Linked List*/
#include<stdio.h>
void push();
void pop();
void display();
main()
{
int n;
printf("STACK USING LINKED LIST\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
do
{
printf("\nEnter your choice\n");
scanf("%d",&n);
switch(n)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
break;
default:
printf("Invalid choice\n");
break;
}
}
while(n!=4);
}

typedef struct node
{
int data;
struct node *link;
}n;
n *top=NULL;

void push()
{
int item;
n *temp;
printf("Enter the item\n");
```

```
scanf("%d",&item);
temp=(n*)malloc(sizeof(n));
temp->data=item;
temp->link=top;
top=temp;
}

void pop()
{
n *temp;
if(top==NULL)
printf("Stack is empty\n");
else
{
temp=top;
printf("The element deleted = %d\n",temp->data);
free(temp);
top=top->link;
}
}

void display()
{
n *save;
if(top==NULL)
printf("Stack is empty\n");
else
{
save=top;
printf("The elements of the stack are :");
while(save!=NULL)
{
printf("%d\t",save->data);
save=save->link;
}
printf("\nTopmost element = %d\n",top->data);
}
}
```

Output:

```
 STACK USING LINKED LIST
1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter your choice
1
Enter the item
10

Enter your choice
1
Enter the item
20

Enter your choice
3
The elements of the stack are :20       10
Topmost element = 20

Enter your choice
2
The element deleted = 20

Enter your choice
4
```

Ex. No.: 9	STACK APPLICATION – INFIX TO POSTFIX CONVERSION

Aim:

To Write a C Program to perform infix to postfix conversion using stack.

Algorithm:

1. Define a stack
2. Go through each character in the string
3. If it is between 0 to 9, append it to output string.
4. If it is left brace push to stack
5. If it is operator *+-/ then
    a. If the stack is empty push it to the stack
    b. If the stack is not empty then start a loop:
        i. If the top of the stack has higher precedence
        ii. Then pop and append to output string
        iii. Else break
        iv. Push to the stack
6. If it is right brace then
    a. While stack not empty and top not equal to left brace
    b. Pop from stack and append to output string
    c. Finally pop out the left brace.
7. If there is any input in the stack pop and append to the output string.

Program:

```
/*Stack application -  Infix to Postfix Conversion*/
#define SIZE 50
#include <ctype.h>
char s[SIZE];
int top = -1;
push(char elem)
{
 s[++top] = elem;
}
char pop()
{
 return (s[top--]);
}
int pr(char elem)
{
 switch (elem)
{
 case '#':
  return 0;
 case '(':
  return 1;
 case '+':
 case '-':
  return 2;
 case '*':
 case '/':
  return 3;
 }
}

main()
{
 char infx[50], pofx[50], ch, elem;
 int i = 0, k = 0;
 printf("<-----Stack Application: Infix to Postfix Conversion----->\n");
 printf("\n\nRead the Infix Expression ? ");
 scanf("%s", infx);
 push('#');
 while ((ch = infx[i++]) != '\0')
{
  if (ch == '(')
   push(ch);
  else if (isalnum(ch))
   pofx[k++] = ch;
  else if (ch == ' )' )
{
```

```
   while (s[top] != ' ( ' )
    pofx[k++] = pop();
   elem = pop();
  }
else
{
   while (pr(s[top]) >= pr(ch))
    pofx[k++] = pop();
   push(ch);
  }
 }
 while (s[top] != '#')
  pofx[k++] = pop();
 pofx[k] = '\0';
 printf("\n\nGiven Infix Expn: %s  Postfix Expn: %s\n", infx, pofx);
}
```

Output:

<-----Stack Application: Infix to Postfix Conversion----->

Read the Infix Expression ? a+b*c-d

Given Infix Expn: a+b*c-d Postfix Expn: abc*+d-

Ex. No.: 10	STACK APPLICATION – EVALUATING POSTFIX EXPRESSION

Aim:

To Write a C Program to evaluate postfix expression using stack.

Algorithm:

1. Start the program.
2. Scan the Postfix string from left to right.
3. Initialise an empty stack.
4. If the scannned character is an operand, add it to the stack. If the scanned character is an operator, there will be atleast two operands in the stack.
5. If the scanned character is an Operator, then we store the top most element of the stack(topStack) in a variable temp. Pop the stack. Now evaluate topStack(Operator)temp. Pop the stack and Push result into the stack.
6. Repeat this step till all the characters are scanned.
7. After all characters are scanned, we will have only one element in the stack. Return topStack.
8. Stop the program.

Program:

```
#define SIZE 50
#include <ctype.h>
int s[SIZE];
int top=-1;
push(int elem)
{
 s[++top]=elem;
}

int pop()
{
 return(s[top--]);
}
 main()
{
 char pofx[50],ch;
 int i=0,op1,op2;
 printf("<-----Stack Application: Evaluating Postfix Expression----->\n");
 printf("\n\nRead the Postfix Expression ? ");
 scanf("%s",pofx);
 while( (ch=pofx[i++]) != '\0')
 {
  if(isdigit(ch)) push(ch-'0');
  else
  {
   op2=pop();
   op1=pop();
   switch(ch)
   {
   case '+':push(op1+op2);break;
   case '-':push(op1-op2);break;
   case '*':push(op1*op2);break;
   case '/':push(op1/op2);break;
   }
  }
 }
 printf("\n Given Postfix Expn: %s\n",pofx);
 printf("\n Result after Evaluation: %d\n",s[top]);
}
```

Output:

<-----Stack Application: Evaluating Postfix Expression----->

Read the Postfix Expression ? 456*+7-

Given Postfix Expn: 456*+7-

Result after Evaluation: 27

Ex. No.:11	QUEUE – ARRAY IMPLEMENTATION

Aim:

To write a C program to implement the concept of queue using arrays.

Algorithm:

Enqueue:

1. Check if queue is not full.
2. If it is full then return queue overflow and item cannot be inserted.
3. If not, check if rear value is -1, if so then increment rear and by 1; if not increment front by 1.
4. Store the item in the new value of front.

Dequeue:

1. Check if queue is not empty.
2. If it is empty, return queue underflow and dequeue operation cannot be done.
3. If not, check if rear and front are equal, if so assign -1 to front and rear; if not decrement front by 1.

Program:

```
/*Queue using Array*/
#include<stdio.h>
#include<conio.h>
#define MAX 10
int queue[MAX],front=-1,rear=-1;
void insert_element();
void delete_element();
void display_queue();
 int main()
{
  int option;
  printf(">>> c program to implement queue operations <<<");
  do
  {
   printf("\n\n 1.Enqueue an element");
   printf("\n 2.Dequeue an element");
   printf("\n 3.Display queue");
   printf("\n 4.Exit");
   printf("\n Enter your choice: ");
   scanf("%d",&option);
   switch(option)
   {
     case 1: insert_element();
              break;
     case 2: delete_element();
             break;
     case 3: display_queue();
             break;
     case 4: return 0;
   }
   }while(option!=4);
}
void insert_element()
{
  int num;
  printf("\n Enter the number to be Enqueued: ");
  scanf("%d",&num);
  if(front==0 && rear==MAX-1)
    printf("\n Queue OverFlow Occured");
  else if(front==-1&&rear==-1)
  {
     front=rear=0;
     queue[rear]=num;
  }
  else if(rear==MAX-1 && front!=0)
```

```
  {
   rear=0;
   queue[rear]=num;
  }
  else
  {
     rear++;
     queue[rear]=num;
  }
}
void delete_element()
{
 int element;
 if(front==-1)
 {
     printf("\n Underflow");
 }
 element=queue[front];
 if(front==rear)
    front=rear=-1;
 else
 {
  if(front==MAX-1)
     front=0;
  else
     front++;
      printf("\n The dequeued element is: %d",element);
 }

}
void display_queue()
{
   int i;
   if(front==-1)
     printf("\n No elements to display");
   else
   {
     printf("\n The queue elements are:\n ");
     for(i=front;i<=rear;i++)
     {
        printf("\t %d",queue[i]);
     }
   }
}
```

Output:

```
>>> c program to implement queue operations <<<

 1.Enqueue an element
2.Dequeue an element
3.Display queue
4.Exit
Enter your choice: 1

Enter the number to be Enqueued: 10


1.Enqueue an element
2.Dequeue an element
3.Display queue
4.Exit
Enter your choice: 1

Enter the number to be Enqueued: 20


1.Enqueue an element
2.Dequeue an element
3.Display queue
4.Exit
Enter your choice: 3

The queue elements are:
        10      20

1.Enqueue an element
2.Dequeue an element
3.Display queue
4.Exit
Enter your choice: 2

The dequeued element is: 10

1.Enqueue an element
2.Dequeue an element
3.Display queue
4.Exit
Enter your choice: 4
```

Ex. No.:12	QUEUE – LINKED LIST IMPLEMENTATION

Aim:

To write a C program to implement the concept of queue using linked list.

Algorithm:

Enqueue:

1. Create a new node and allocate memory space for the new node.
2. Assign the element to be inserted in the data field of the new node.
3. Assign NULL to the address field of the new node.
4. Check if rear and front pointers are NULL.
5. If so, then make the front and rear pointers to point to new node.
6. If not, then assign address of the new node as the rear pointer value.

Dequeue:

1. Check if queue is not empty.
2. If it is empty, return queue underflow and dequeue operation cannot be done.
3. If not, assign the front->next value as the new front pointer and free the deleted node.

Program:

//Queue using linked list

```
#include<stdio.h>
 #include<conio.h>
 #include<stdlib.h>
 struct node
 {
       int data;
       struct node* next;
 }*rear, *front;

void dequeue()
{
      struct node *temp, *var=rear;
     if(var==rear)
     {
          rear = rear->next;
          free(var);
     }
     else
     printf("\nQueue Empty");
}

void enqueue(int value)
{
    struct node *temp;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=value;
    if (front == NULL)
    {
        front=temp;
        front->next=NULL;
        rear=front;
    }
    else
    {
        front->next=temp;
        front=temp;
        front->next=NULL;
    }
}

void display()
{
    struct node *var=rear;
    if(var!=NULL)
```

```
    {
        printf("\nElements in Queue:  ");
        while(var!=NULL)
        {
            printf("\t%d",var->data);
            var=var->next;
        }
    printf("\n");
    }
    else
    printf("\nQueue is Empty");
}

int main()
{
    int ch;
    clrscr();
    front=NULL;
    printf("<-----Queue using Linked List----->\n");
    printf(" \n1. Enqueue an element");
    printf(" \n2. Dequeue an element");
    printf(" \n3. Display Queue");
    printf(" \n4. Exit\n");
    while(1)
    {
        printf(" \nEnter your choice: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
            {
                int value;
                printf("\nEnter a value to Enqueue: ");
                scanf("%d",&value);
                enqueue(value);
                display();
                break;
            }
            case 2:
            {
                dequeue();
                display();
                break;
            }
            case 3:
            {
                display();
```

```
            break;
        }
        case 4:
        {
            exit(0);
        }
        default:
        {
            printf("\nwrong choice for operation");
        }
    }
  }
}
```

OUTPUT:

<-----Queue using Linked List----->

1. Enqueue an element
2. Dequeue an element
3. Display Queue
4. Exit

Enter your choice: 1

Enter a value to Enqueue: 10

Elements in Queue: 10

Enter your choice: 1

Enter a value to Enqueue: 20

Elements in Queue: 10 20

Enter your choice: 1

Enter a value to Enqueue: 30

Elements in Queue: 10 20 30

Enter your choice: 2

Elements in Queue: 20 30

Enter your choice: 4

Ex. No.: 13	SEARCHING

Aim:

To Write a C Program to perform linear search and binary search.

Algorithm:

Linear Search

1. Read n numbers and search value.
2. If search value is equal to first element then print value is found.
3. Else search with the second element and so on.

Binary Search

1. Read n numbers and search value.
2. If search value is equal to middle element then print value is found.
3. If search value is less than middle element then search left half of list with the same method.
4. Else search right half of list with the same method.

Program:

```
/*Searching*/
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
int a[100],i,n,item,s=0,ch,beg,end,mid;
clrscr();
printf("Enter No. of Elements:");
scanf("%d",&n);
printf("\nEnter Elements:\n");
for(i=1;i<=n;i++)
{
scanf("%d",&a[i]);
}
while(1)
{
printf("\n1.Linear Search\n2.Binary Search\n3.Exit\n");
printf("Enter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("<-----LINEAR SEARCH----->\n");
printf("\nEnter Element you want to Search:");
scanf("%d",&item);
for(i=1;i<=n;i++)
{
if(a[i]==item)
{
printf("\nData is Found at Location : %d",i);
s=1;
break;
}
}
if(s==0)
{
printf("Data is Not Found");
}
break;
case 2:
printf("<-----BINARY SEARCH----->\n");
printf("\nEnter Item you want to Search:");
scanf("%d",&item);
beg=1;
end=n;
```

```
mid=(beg+end)/2;
while(beg<=end && a[mid]!=item)
{
if(a[mid]<item)
beg=mid+1;
else
end=mid-1;
mid=(beg+end)/2;
}
if(a[mid]==item)
{
printf("\nData is Found at Location : %d",mid);
}
else
{
printf("Data is Not Found");
}
break;
case 3:
default:
exit(0);
}
}
getch();
}
```

Output:

```
Enter No. of Elements:
5

Enter Elements:
2
4
3
5
1

1.Linear Search
2.Binary Search
3.Exit
Enter your choice:
1
<-----LINEAR SEARCH----->

Enter Element you want to Search:
1

Data is Found at Location : 5
1.Linear Search
2.Binary Search
3.Exit
Enter your choice:
2
<-----BINARY SEARCH----->

Enter Item you want to Search:
3

Data is Found at Location : 3
1.Linear Search
2.Binary Search
3.Exit
Enter your choice:
```

Ex. No.: 14	SORTING

Aim:

To Write a C Program to perform insertion sort, quick sort and bubble sort**.**

Algorithm:

Insertion Sort

1. Get the n elements to be sorted.
2. The ith element is compared from (i-1)th to 0th element and placed in proper position according to ascending value.
3. Repeat the above step until the last element.

Quick Sort

1. Pick an element, called a pivot, from the list.
2. Reorder the list so that all elements which are less than the pivot come before the pivot and so that all elements greater than the pivot come after it.
3. After this partitioning, the pivot is in its final position. This is called the partition operation.
4. Recursively sort the sub-list of lesser elements and the sub-list of greater elements

Bubble Sort

1. Get the n elements to be sorted.
2. Compare the first two elements of the array and swap if necessary.
3. Then, again second and third elements are compared and swapped if it is necessary and continue this process until last and second last element is compared and swapped.
4. Repeat the above two steps n-1 times and print the result.

Program:

```
/*Sorting*/
#include<conio.h>
#include<stdio.h>
#include<process.h>
void quickSort(int numbers[], int array_size);
void q_sort(int numbers[], int left, int right);
void bubble(int *array,int length);
void insertion(int a[], int n);
void insertion(int a[], int n)
{
 int i,j,temp;
 for(i=1;i<n;i++)
   {
     temp=a[i];
     j=i-1;
     while((temp<a[j])&&(j>=0))
          {
            a[j+1]=a[j];
        j=j-1;
       }
     a[j+1]=temp;
   }
}
void display(int a[],int n)
{
int i;
printf("\n\t\t\tSorted List\n");
for(i=0;i<n;++i)
printf("\t%d",a[i]);
}
void q_sort(int a[], int left, int right)
{
int pivot, l_hold, r_hold;
l_hold = left;
r_hold = right;
pivot = a[left];
while (left < right)
{
while ((a[right] >= pivot) && (left < right))
right--;
if (left != right)
{
a[left] = a[right];
left++;
}
while ((a[left] <= pivot) && (left < right))
```

```
left++;
if (left != right)
{
  a[right] = a[left];
  right--;
}
}
a[left] = pivot;
pivot = left;
left = l_hold;
right = r_hold;
if (left < pivot)
q_sort(a, left, pivot-1);
if (right > pivot)
q_sort(a, pivot+1, right);
}
void bubble(int *array,int length)
{
int i,j;
for(i=0;i<length;i++)
{
for(j=0;j<i;j++)
{
if(array[i]>array[j])
{
int temp=array[i];
array[i]=array[j];
array[j]=temp;
}
}
}
}
void main()
{
int a[100],n,i,ch;
clrscr( );
printf("\nEnter The Number Of Elements\t: ");
scanf("%d",&n);
printf("\nEnter Elements\n");
for(i=0;i< n;++i)
scanf("%d",&a[i]);
while(1)
{
printf("\n1.Insertion sort\n2.Quick sort\n3.Bubble sort\n4.Exit\n");
printf("Enter your choice:");
scanf("%d",&ch);
switch(ch)
```

```
{
case 1:
printf("<-----Insertion SORT----->\n");
insertion(a,n);
display(a,n);
break;
case 2:
printf("<-----Quick SORT----->\n");
q_sort(a,0,n-1);
display(a,n);
break;
case 3:
bubble(a,n);
printf("<-----Bubble SORT----->\n");
printf("\n\t\t\tSorted List\n");
for (i=n-1;i>=0;i--)
printf("\t%d",a[i]);
break;
case 4:
exit(0);
default:
printf("Enter a Valid Choice!");
}
}
getch();
}
```

Output:

```
Enter The Number Of Elements    :
5

Enter Elements
2
4
1
3
5

1.Insertion sort
2.Quick sort
3.Bubble sort
4.Exit
Enter your choice:
1
<-----Insertion SORT----->

                Sorted List
      1      2      3      4      5
1.Insertion sort
2.Quick sort
3.Bubble sort
4.Exit
Enter your choice:
2
<-----Quick SORT----->

                Sorted List
      1      2      3      4      5
1.Insertion sort
2.Quick sort
3.Bubble sort
4.Exit
Enter your choice:
3
<-----Bubble SORT----->

                Sorted List
      1      2      3      4      5
```