DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE
(AUTONOMOUS)
(Approved by AICTE & Affiliated to Anna University, Chennai)
Re-Accredited by NAAC with 'A' Grade
Accredited by NBA for AERO, BME, CSE, ECE, EEE, IT & MECH.
PERAMBALUR-621212, TAMILNADU, INDIA.
Website: www.dsengg.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

LAB MANUAL

U23ITP32 - DATA STRUCTURE LABORATORY

REGUALTION: 2023

YEAR/SEM: II/III **BATCH**: 2023 - 2027

Prepared By **Approved By**

LIST OF EXERCISES

1. Array implementation of Stack and Queue ADTs
2. Array implementation of List ADT
3. Linked list implementation of List, Stack and Queue ADTs
4. Applications of List, Stack and Queue ADTs
5. Implementation of Binary Trees and operations of Binary Trees
6. Implementation of Binary Search Trees
7. Implementation of AVL Trees
8. Implementation of Heaps using Priority Queues.
9. Graph representation and Traversal algorithms
10. Applications of Graphs
11. Implementation of searching and sorting algorithms
12. Hashing – any two collision techniques

INDEX

Ex.No.	Date	Title	Marks	Staff Sign.
1a		Stack-Array Implementation		
1b		Queue-Array Implementation		
2		List-Array Implementation		
3a		List-Linked List Implementation		
3b		Stack-Linked List Implementation		
3c		Queue-Linked List Implementation		
4a		Polynomial Addition		
4b		Infix to Postfix Conversion		
4c		Evaluation Postfix Expression		
5		Binary Tree		
6		Binary Search Tree		
7		AVL Tree		
8		Heap using Priority Queue		
9		Graph Traversal		
10		Topological Sort		
11a		Linear & Binary Search		
11b		Sorting		
12		Linear Probing		

Ex.No.-1a STACK – ARRAY IMPLEMENTATION

AIM:

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

ALGORITHM:

(i) Push Operation:

- 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.
- If so, then return stack is full and element cannot be pushed into the stack.
- Else, increment the top by one and push the new element in the new position of top.

(ii) Pop Operation:

- To pop an element from the stack, check whether the top of the stack is equal to -1.
- If so, then return stack is empty and element cannot be popped from the stack. o 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

RESULT:

Thus a C program to implement the stack using arrays is written and executed successfully.

Ex.No.-1b QUEUE – ARRAY IMPLEMENTATION

AIM:

To write a C program to implement the 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 queue is not empty , 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

RESULT:

Thus a C program to implement the queue using arrays is written and executed successfully.

Ex.No.-2 LIST – ARRAY IMPLEMENTATION

AIM:

To write a C program to implement the List using arrays.

ALGORITHM:

1. Start the program.
2. Read the number of elements in the list and create the list.
3. Read the position and element to be inserted.
4. Adjust the position and insert the element.
5. Read the position and element to be deleted.
6. Remove the element from the list and adjust the position.
7. Read the element to be searched.
8. Compare the elements in the list with searching element.
9. If element is found, display it else display element is not found.
10. Display all the elements in the list.
11. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define MAX 10

void create( );
void insert( );
void deletion( );
void search( );
void display ();
int a,b[20], n, p, e, f, i, pos;

void main( )
{
        int ch;
        char g='y';
        do
        {
```

```
printf("\n Main Menu");
printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit \n");
printf("\n Enter your Choice:");
scanf("%d", &ch);
switch(ch)
{
        case 1:
                create();
                break;
        case 2:
                deletion();
                break;
        case 3:
                search();
                break;
        case 4:
                insert();
                break;
        case 5:
                display();
                break;
        case 6:
        exit();
        break;

        default:
        printf("\n Enter the correct choice:");
        }
        printf("\n Do u want to continue:");
        scanf("\n%c", &g);
        }
        while(g=='y'||g=='Y');
        getch();
        }


        void create( )
        {
        printf("\n Enter the number of nodes:");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
        printf("\n Enter the Element:",i+1);
        scanf("%d", &b[i]);
        }

        }
```

```
void deletion( )
{
printf("\n Enter the position u want to delete:");
scanf("%d", &pos);
if(pos>=n)
{
printf("\n Invalid Location:");
}
else
{
for(i=pos+1;i<n;i++)
{
b[i-1]=b[i];
}
n--;
}
printf("\n The Elements after deletion:");
for(i=0;i<n;i++)
{
printf("\t%d", b[i]);
}
}


void search()
{
printf("\n Enter the Element to be searched:");
scanf("%d", &e);

for(i=0;i<n;i++)
{
if(b[i]==e)
{
printf("Value is in the %d Position", i);
}
else
{
printf("Value %d is not in the list:", e);
continue;
}
}
}


void insert()
{
```

```
printf("\n Enter the position u need to insert:");
scanf("%d", &pos);

if(pos>=n)
 {
 printf("\n invalid Location:");
 }
 else
 {
 for(i=MAX-1;i>=pos-1;i--)
 {
 b[i+1]=b[i];
 }
 printf("\n Enter the element to insert:");
 scanf("%d",&p);
 b[pos]=p;
 n++;
 }
 printf("\n The list after insertion:");

display();
}


void display()
{
printf("\n The Elements of The list ADT are:");
for(i=0;i<n;i++)
{
printf("\n\n%d", b[i]);
}
}
```

OUTPUT:

```
 Main Menu
1. Create
2. Delete
3. Search
4. Insert
5. Display
6. Exit
Enter your Choice:1

Enter the number of nodes:4
Enter the Element: 11
Enter the Element: 22
```

Enter the Element: 33
Enter the Element: 44

Do u want to continue:y

Main Menu
1.Create
2.Delete
3.Search
4.Insert
5.Display
6.Exi
Enter your Choice:2
Enter the position u want to delete:2
Elements after deletion: 11 33 44

Do u want to continue:y
Main Menu
1.Create
2.Delete
3.Search
4.Inse rt
5.Display
6.Exit
Enter your Choice:3

Enter the Element to be searched:44
Value is in the 3 Position

Do u want to continue:y

Main Menu
1.Create
2.Delete
3.Search
4.Inse rt
5.Display
6.Exi
Enter your Choice:4
Enter the position u need to insert:2
Enter the element to insert:25
The l st after insertion: 11 25 33 44

Do u want to continue:y

Main Menu
1. Create

2. Delete
3. Search
4. Insert
5. Display
6. Exit
Enter your Choice:5

The Elements of The list ADT are: 11 25 33 44

Do u want to continue:y

Main Menu
1. Create
2. Delete
3. Search
4. Insert
5. Display
6. Exit
Enter your Choice:6

RESULT:

Thus a C program to implement the List using array is written and executed successfully.

Ex.No.-3a LIST – LINKED LIST IMPLEMENTATION

AIM:

To write a C program to implement the List ADT using linked list.

ALGORITHM:

1. Start the program.
2. Create a node using structure
3. Dynamically allocate memory to node
4. Create and add nodes to linked list.
5. Read the element to be inserted.
6. Insert the elements into the list.
7. Read the element to be deleted.
8. Remove that element and node from the list.
9. Adjust the pointers.
10. Display the elements in the list.
11. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
/*declaring a structure to create a node*/
struct node
{
        int data;
        struct node *next;
};
struct node *start;
/* inserting nodes into the list*/
/*function to insert values from beginning of the the single linked
list*/ void insertbeg(void)
{
        struct node *nn;
        int a;
        /*allocating implicit memory to the node*/
        nn=(struct node *)malloc(sizeof(struct node));
```

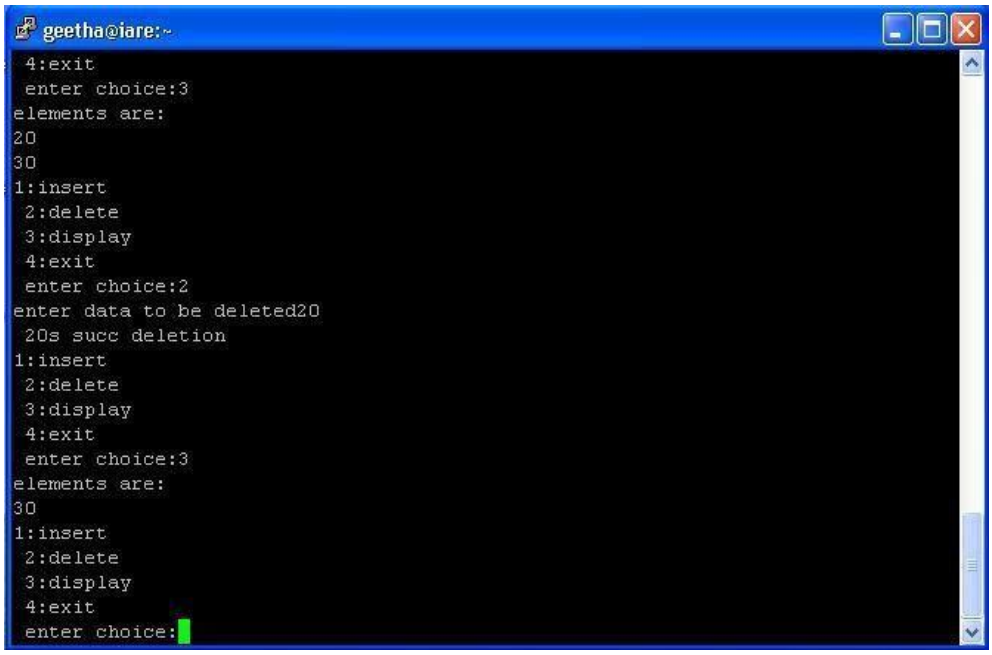
```
        printf("enter data:");
        scanf("%d",&nn->data);
        a=nn->data;
        if(start==NULL)
        /*checking if List is empty*/
        {
                nn->next=NULL;
               start=nn;
        }
        else
        {
                nn->next=start;
               start=nn;
        }
         printf("%d succ. inserted\n",a);
        return;
}
/*function to insert values from the end of the linked list*/
void insertend(void)
{
        struct node *nn,*lp;
        int b;
        nn=(struct node *)malloc(sizeof(struct node));
        printf("enter data:");
        scanf("%d",&nn->data);
        b=nn->data;
        if(start==NULL)
        {
                nn->next=NULL;
                start=nn;
        }
         else
        {
                lp=start;
                while(lp->next!=NULL)
                {
                        lp=lp->next;
                }
                lp->next=nn;
                nn->next=NULL;
        }
         printf("%d is succ. inserted\n",b);
         return;
 }
 /*function to insert values from the middle of the linked list*/
void insertmid(void)
 {
```

```
        struct node *nn,*temp,*ptemp;
        int x,v;
        nn=(struct node *)malloc(sizeof(struct node));
        if(start==NULL)
        {
                printf("sll is empty\n");
                return;
        }
        printf("enter data before which no. is to be inserted:\n");
        scanf("%d",&x);
        if(x==start->data)
        {
                insertbeg();
                return;
        }
        ptemp=start;
        temp=start->next;
        while(temp!=NULL&&temp->data!=x)
        {
                ptemp=temp;
                temp=temp->next;
        }
        if(temp==NULL)
        {
                printf("%d data does not exist\n",x);
        }
        else
        {
                printf("enter data:");
                scanf("%d",&nn->data);
                v=nn->data;
                ptemp->next=nn;
                nn->next=temp;
                printf("%d succ. inserted\n",v);
        }
        return;
}
void deletion(void)
{
        struct node *pt,*t;
        int x;
        if(start==NULL)
        {
                printf("sll is empty\n");
                return;
        }
        printf("enter data to be deleted:");
```

```
        scanf("%d",&x);
        if(x==start->data)
        {
                t=start;
                /* assigning first node pointer to next node pointer to delete a
                data from the starting of the node*/
                start=start->next;
                free(t);
                printf("%d is succ. deleted\n",x);
                return;
        }
        pt=start;
        t=start->next;
        while(t!=NULL&&t->data!=x)
        {
                 pt=t;t=t->next;
        }
        if(t==NULL)
        {
                printf("%d does not exist\n",x);
                return;
         }
        else
        {
                pt->next=t->next;
        }
        printf("%d is succ. deleted\n",x);
        free(t);
        return;
}
void display(void)
{
        struct node *temp;
        if(start==NULL)
        {
                printf("sll is empty\n");
                return;
        }
        printf("elements are:\n");
        temp=start;
        while(temp!=NULL)
        {
                 printf("%d\n",temp->data);
                 temp=temp->next;
        }
        return;
 }
```

```
/* main program*/
int main( )
{
        int c,a;
        start=NULL;
        do
        {
                printf("1:insert\n2:delete\n3:display\n4:exit\nenter choice:");
                scanf("%d",&c);
                switch(c)
                {
                        case 1:
                                printf("1:insertbeg\n2:insert end\n3:insert mid\nenter choice:");
                                scanf("%d",&a);
                                switch(a)
                                {
                                        case 1:insertbeg(); break;
                                        case 2:insertend(); break;
                                        case 3:insertmid(); break;
                                } break;
                        case 2:deletion(); break;
                        case 3:display(); break;
                        case 4:printf("program ends\n");break;
                        default:printf("wrong choice\n"); break;
                 }
         }while(c!=4);
        return 0;
 }
```

OUTPUT:



RESULT:

Thus a C program to implement the List ADT using linked list is written and executed successfully.

Ex.No.-3b STACK – LINKED LIST IMPLEMENTATION

AIM:

To write a C program to implement the stack using 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\n 2.POP\n 3.DISPLAY\n 4.
                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

RESULT:

Thus a C program to implement the stack using linked list is written and executed successfully.

Ex.No.-3c QUEUE – LINKED LIST IMPLEMENTATION

AIM:

To write a C program to implement the 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

RESULT:

Thus a C program to implement the queue using linked list is written and executed successfully.

Ex.No.-4a POLYNOMIAL ADDITION

AIM:

To write a C program to perform polynomial addition using linked list.

ALGORITHM:

1. Start the program.
2. Declare the node as link.
3. Allocate memory space for pol1, pol2 and poly.
4. Read pol1 and pol2.
5. Call the function polyadd.
6. Add the co-efficients of poly1 and poly2 having the equal power and store it in poly.
7. If there is no co-efficient having equal power in poly1 and poly2, then add it to poly.
8. Display the poly1, poly2 and poly.
9. Stop the program.

PROGRAM:

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

struct link
{
        int coeff,pow;
        struct link *next;
}*poly1=NULL, *poly2=NULL,*poly=NULL;

void create(struct link *node)
{
        char ch;
        do
        {
                        printf("\n Enter Coeff:");
                        scanf("%d",&node->coeff);
                        printf("\n Enter Power:");
                        scanf("%d",&node->pow);
                        node->next=(struct link *) malloc(sizeof(struct link));
                        node=node->next;
                        node->next=NULL;
```

```
                printf("\n\n Continue(y/n):");
                ch=getch();
        }while(ch=="y" || ch =="Y");
}

void display(struct link *node)
{
                while(node->next!=NULL)
                {
                    printf("%dx^%d",node->coeff,node->pow);
                    node=node->next;
                    if(node->next!=NULL)
                            printf("+");
                }
}

void polyadd(struct link *poly1, struct link *poly2, struct link *poly)
{
                while(poly1->next && poly2->next)
                {
                    if(poly1->pow > poly2->pow)
                    {
                            poly->pow=poly1->pow;
                            poly->coeff=poly1->coeff;
                            poly1=poly1->next;
                            }
                    else if(poly1->pow < poly2->pow)
                    {
                            poly->pow=poly1->pow;
                            poly->coeff=poly1->coeff;
                            poly2=poly2->next;
                    }
                    else
                    {
                            poly->pow=poly1->pow;
                            poly->coeff=poly1->coeff + poly2->coeff;
                            poly1=poly1->next;
                            poly2=poly2->next;

                    }
                    poly->next=(struct link *) malloc(sizeof(struct link));
                    poly=poly->next;
                    poly->next=NULL;
}
                while(poly1->next || poly2->next)
                {
                    if(poly1->next)
```

```
                {
                        poly->pow=poly1->pow;
                        poly->coeff=poly1->coeff;
                        poly1=poly1->next;
                }
                if(poly2->next)
                {
                        poly->pow=poly2->pow;
                        poly->coeff=poly2->coeff;
                        poly2=poly2->next;
                }
                poly->next=(struct link *) malloc(sizeof(struct link));
                poly=poly->next;
                poly->next=NULL;
}
}
void main( )
{
                poly1=(struct link *)malloc(sizeof(struct link));
                poly2=(struct link *)malloc(sizeof(struct link));
                poly=(struct link *)malloc(sizeof(struct link));
                clrscr();
                printf("\n Enter the First Polynomial:");
                create(poly1);
                printf("\n Enter the Second Polynomial:");
                create(poly2);
                polyadd(poly1,poly2,poly);
                printf("\n\t First Polynomial:");
                display(poly1);
                printf("\n\t Second Polynomial:");
                display(poly2);
                printf("\n\t Addition of two Polynomials:");
                display(poly);
                getch();
}
```

OUTPUT:

```
Enter the First Polynomial:
Enter Coeff:3
Enter Power:3
Continue(y/n):y

Enter Coeff:6
Enter Power:2
Continue(y/n):y

Enter Coeff:9
```

Enter Power:1
Continue(y/n):y

Enter Coeff:8
Enter Power:0
Continue(y/n):n

Enter the Second Polynomial:
Enter Coeff:76
Enter Power:3
Continue(y/n):y

Enter Coeff:43
Enter Power:2
Continue(y/n):y

Enter Coeff:23
Enter Power:1
Continue(y/n):y

Enter Coeff:24
Enter Power:0
Continue(y/n):n

First Polynomial: 3x^3+6x^2+9x^1+8x^0
Second Polynomial: 76^3+43x^2+23x^1+24x^0
Addition of two Polynomials: 79^3+49x^2+32x^1+32x^0

RESULT:

Thus a C program to perform polynomial addition using linked list is written and executed successfully.

Ex.No.-4b 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-

RESULT:

Thus a C program to convert the expression in infix to postfix is written and executed successfully.

Ex.No.-4c 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

RESULT:

Thus a C program to evaluate the postfix expression is written and executed successfully.

Ex.No.-5 BINARY TREE

AIM:

To write a C program to implement binary tree and its traversals.

ALGORITHM:

1. Start the program.
2. Declare the node.
3. Create the binary tree by inserting elements into it.
4. Traverse the binary tree by inorder and display the nodes.
5. Traverse the binary tree by preorder and display the nodes.
6. Traverse the binary tree by postorder and display the nodes.
7. Stop the program.

PROGRAM:

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

typedef struct bin
{
        int data;
        struct bin *left;
        struct bin *right;
}node;

void insert(node *,node *);
void inorder(node *);
void preorder(node *);
void postorder(node *);
node *getnode();

void main( )
{
        int choice;
        char ans='n';
        node *newnode,*root;
        root=NULL;
        clrscr ( ) ;
        do
```

```
{
        printf("\n\t Program for Binary Tree Travesal");
        printf("\n\t 1.Create");
        printf("\n\t 2.Inorder");
        printf("\n\t 3.Preorder");
        printf("\n\t 4.Postorder");
        printf("\n\t 5.Exit");
        printf("\n\t Enter your Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:
                        root=NULL;
                        do
                        {
                                newnode=getnode();
                                printf("\n\tEnter the element:");
                                scanf("%d",&newnode->data);
                                if(root==NULL)
                                root=newnode;
                                else
                                insert(root,newnode);
                                printf("\n\tDo u want to enter more elements?(y/n):
                                "); ans=getche();
                        }while(ans=='y' || ans=='Y');
                        clrscr();
                        break;

                case 2:
                        if(root==NULL)
                                printf("\n\t Tree is not created.");
                        else
                                inorder(root);
                        break;

                case 3:
                        if(root==NULL)
                                printf("\n\t Tree is not created.");
                        else
                                preorder(root);
                        break;
                case 4:
                        if(root==NULL)
                                printf("\n\t Tree is not created.");
                        else
                                postorder(root);
                        break;
```

```
                }
        }while(choice!=5);
}

node *getnode()
{
        node *temp;
        temp=(node *)malloc(sizeof(node));
        temp->left=NULL;
        temp->right=NULL;
        return temp;
}

void insert(node *root, node *newnode)
{
        char ch;
        printf("\n\t Where to insert LEFT/RIGHT of %d: ",root->data);
        ch=getche();
        if((ch=='r')||(ch=='R'))
        {
                if(root->right==NULL)
                {
                        root->right=newnode;
                }
                else
                        insert(root->right,newnode);;
        }
        else
        {
                if(root->left==NULL)
                {
                        root->left=newnode;
                }
                else
                        insert(root->left,newnode);
        }
}

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

```
}

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

void postorder(node *temp)
{
        if(temp!=NULL)
        {
                postorder(temp->left);
                postorder(temp->right);
                printf(" %d",temp->data);

        }
}
```

OUTPUT:

```
Program for Binary Tree Travesal
1. Create
2. Inorder
3. Preorder
4. Postorder
5. Exit
Enter your Choice:1
Enter the element:10
Do u want to enter more elements?(y/n):y

Enter the element:12
Where to insert LEFT/RIGHT of 10: l
Do u want to enter more elements?(y/n):y

Enter the element:17
Where to insert LEFT/RIGHT of 10: r
Do u want to enter more elements?(y/n):y

Enter the element:8
Where to insert LEFT/RIGHT of 10: l
Where to insert LEFT/RIGHT of 12: r
```

Do u want to enter more elements?(y/n):n

Program for Binary Tree Travesal
1. Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your Choice:2
12 8 10 17

Program for Binary Tree Travesal
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your Choice:3
10 12 8 17

Program for Binary Tree Travesal
1.Create
2.Inorder
3.Preorder
4.Postorder
5.Exit
Enter your Choice:4
8 12 17 10

Program for Binary Tree Travesal
1. Create
2. Inorder
3. Preorder
4. Postorder
5. Exit
Enter your Choice:5

RESULT:

Thus a C program to implement the tree and tree traversals is written and executed successfully.

Ex.No.-6 BINARY SEARCH TREE

AIM:

To write a C program to implement binary search tree.

ALGORITHM:

1. Start the program.
2. Declare the node.
3. Read the elements to be inserted.
4. Create the binary search tree.
5. Read the element to be searched.
6. Visit the nodes by inorder.
7. Find the searching node and display if it is present with parent node.
8. Read the element to be removed from BST.
9. Delete that node from BST.
10. Display the binary search tree by inorder.
11. Stop the program.

PROGRAM:

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

typedef struct bst
{
        int data;
        struct bst *left,*right;
}node;

void insert(node *,node *);
void inorder(node *);
node *search(node *,int,node **);
void del(node *,int);

void main()
{
        int choice;
        char ans='N';
```

```
int key;
node *newnode,*root,*temp,*parent;
node *getnode();
root=NULL;
clrscr();
do
{
        printf("\n\t Program for Binary Search Tree");
        printf("\n\t 1.Create");
        printf("\n\t 2.Search");
        printf("\n\t 3.Delete");
        printf("\n\t 4.Display");
        printf("\n\t 5.Exit");
        printf("\n\t Enter your Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
                case 1:
                        do
                        {
                                newnode=getnode();
                                printf("\n\tEnter the element:");
                                scanf("%d",&newnode->data);
                                if(root==NULL)
                                root=newnode;
                                else
                                insert(root,newnode);
                                printf("\n\tDo u want to enter more elements?(y/n):
                                "); ans=getche();
                        }while(ans=='y' || ans=='Y');
                        break;
                case 2:
                        printf("\n\tEnter the element to be searched:");
                        scanf("%d",&key);
                        temp=search(root,key,&parent);
                        printf("\n\tParent of node %d is %d",temp->data,parent-
                        >data); break;
                case 3:
                        printf("\n\tEnter the element to be deleted:");
                        scanf("%d",&key);
                        del(root,key);
                        break;
                case 4:
                        if(root==NULL)
                                printf("\n\t Tree is not created.");
                        else
                        {
```

```
                                        printf("\n The Tree is:");
                                        inorder(root);
                                }
                                break;
                        }
        }while(choice!=5);
}
node *getnode( )
{
        node *temp;
        temp=(node *)malloc(sizeof(node));
        temp->left=NULL;
        temp->right=NULL;
        return temp;
}
void insert(node *root, node *newnode)
{
        if(newnode->data > root->data)
        {
                if(root->right==NULL)
                {
                        root->right=newnode;
                }
                else
                        insert(root->right,newnode);;
        }
        if(newnode->data < root->data)
        {
                if(root->left==NULL)
                {
                        root->left=newnode;
                }
                else
                        insert(root->left,newnode);
        }
}
void inorder(node *temp)
{
        if(temp!=NULL)
        {
                inorder(temp->left);
                printf(" %d",temp->data);
                inorder(temp->right);
        }
}
node *search(node *root,int key,node **parent)
{
```

```
        node *temp;
        temp=root;
        while(temp!=NULL)
        {
            if(temp->data==key)
            {
                    printf("\n\tThe %d element is present",temp->data);
                    return(temp);
            }
            *parent=temp;
            if(temp->data > key)
                temp=temp->left;
            else
                temp=temp->right;
        }
        return NULL;
}
void del(node *root, int key)
{
        node *temp,*parent,*tempsucc;
        temp=search(root,key,&parent);
        if(temp->left!=NULL&&temp->right!=NULL)
        {
                parent=temp;
                tempsucc=temp->right;
                while(tempsucc->left!=NULL)
                {
                        parent=tempsucc;
                        tempsucc=tempsucc->left;
                }
                temp->data=tempsucc->data;
                parent->right=NULL;
                printf("Now Deleted it!");
                return;
        }
        if(temp->left!=NULL&&temp->right!=NULL)
        {
                if(parent->left==temp)
                        parent->left=temp->left;
                else
                        parent->right=temp->left;
                        temp=NULL;
                        free(temp);
                        printf("Now deleted it!");
                        return;
        }
        if(temp->left!=NULL&&temp->right!=NULL)
```

```
        {
                if(parent->left==temp)
                        parent->left=temp->right;
                else
                        parent->right=temp->right;
                temp=NULL;
                free(temp);
                printf("Now deleted it!");
                return;
        }
        if(temp->left!=NULL&&temp->right!=NULL)
        {
                if(parent->left==temp)
                        parent->left=NULL;
                else
                        parent->right=NULL;
                        printf("Now deleted it!");
                        return;
        }
}
```

OUTPUT:

```
Program for Binary Search Tree
1. Create
2. Search
3. Delete
4. Display
5. Exit
Enter your Choice:1

Enter the element:10
Do u want to enter more elements?(y/n):y

Enter the element:8
Do u want to enter more elements?(y/n):y

Enter the element:9
Do u want to enter more elements?(y/n):y

Enter the element:7
Do u want to enter more elements?(y/n):y

Enter the element:15
Do u want to enter more elements?(y/n):y

Enter the element:13
```

Do u want to enter more elements?(y/n):y

Enter the element:14
Do u want to enter more elements?(y/n):y

Enter the element:12
Do u want to enter more elements?(y/n):y

Enter the element:16
Do u want to enter more elements?(y/n):n

1. Create
2. Search
3. Delete
4. Display
5. Exit

Enter your Choice:4

The Tree is: 7 8 9 10 12 13 14 15 16

1. Create
2. Search
3. Delete
4. Display
5. Exit

Enter your Choice:2

Enter the element to be searched:16
The 16 element is present
Parent of node 16 is 15

1. Create
2. Search
3. Delete
4. Display
5. Exit

Enter your Choice:5

RESULT:

Thus a C program to implement the binary search tree is written and executed successfully.

Ex.No.-7 AVL TREE

AIM:

To write a C program to implement an AVL tree.

ALGORITHM:

1. Start the program.
2. Declare the node.
3. Read the elements and create a tree.

Insert:

4. Insert a new node as new leaf node just as in ordinary binary search tree.
5. Now trace the path from inserted node towards root. For each node, „n‟ encountered , check if heights of left(n) and right(n) differ by atmost 1.
    a) if yes, move towards parent(n).
    b) Otherwise restructure the by doing either a single rotation or a double rotation.

Delete:

6. Search the node to be deleted.
7. If the node to be deleted is a leaf node, then simply make it NULL to remove.
8. If the node to be deleted is not a leaf node, then the node must be swapped with its inorder successor. Once the node is swapped, then remove the node.
9. Traverse back up the path towards root, check the balance factor of every node along the path.
10. If there is unbalanced in some subtree then balance the subtree using appropriate single or double rotation.

PROGRAM:

```
#include<stdio.h>

typedef struct node
{
   int data;
   struct node *left,*right;
   int ht;
}node;
```

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

int main( )
{
   node *root=NULL;
   int x,n,i,op;

   do
   {
      printf("\n1)Create:");
      printf("\n2)Insert:");
      printf("\n3)Delete:");
      printf("\n4)Print:");
      printf("\n5)Quit:");
      printf("\n\nEnter Your Choice:");
      scanf("%d",&op);

      switch(op)
      {
         case 1: printf("\nEnter no. of elements:");
                 scanf("%d",&n);
                 printf("\nEnter tree data:");
                 root=NULL;
                 for(i=0;i<n;i++)
                 {
                    scanf("%d",&x);
                    root=insert(root,x);
                 }
                 break;

         case 2: printf("\nEnter a data:");
                 scanf("%d",&x);
                 root=insert(root,x);
                 break;
```

```
            case 3: printf("\nEnter a data:");
                    scanf("%d",&x);
                    root=Delete(root,x);
                    break;

            case 4: printf("\nPreorder sequence:\n");
                    preorder(root);
                    printf("\n\nInorder sequence:\n");
                    inorder(root);
                    printf("\n");
                    break;
        }
    }while(op!=5);

    return 0;
}

node * insert(node *T,int x)
{
    if(T==NULL)
    {
        T=(node*)malloc(sizeof(node));
        T->data=x;
        T->left=NULL;
        T->right=NULL;
    }
    else
        if(x > T->data)        // insert in right subtree
        {
            T->right=insert(T->right,x);
            if(BF(T)==-2)
                if(x>T->right->data)
                    T=RR(T);
                else
                    T=RL(T);
        }
        else
            if(x<T->data)
            {
                T->left=insert(T->left,x);
                if(BF(T)==2)
                    if(x < T->left->data)
                        T=LL(T);
                    else
                        T=LR(T);
            }
```

```
        T->ht=height(T);

        return(T);
}

node * Delete(node *T,int x)
{
    node *p;

    if(T==NULL)
    {
        return NULL;
    }
    else
        if(x > T->data)        // insert in right subtree
        {
            T->right=Delete(T->right,x);
            if(BF(T)==2)
                if(BF(T->left)>=0)
                    T=LL(T);
                else
                    T=LR(T);
        }
        else
            if(x<T->data)
            {
                T->left=Delete(T->left,x);
                if(BF(T)==-2)    //Rebalance during windup
                    if(BF(T->right)<=0)
                        T=RR(T);
                    else
                        T=RL(T);
            }
            else
            {
                //data to be deleted is found
                if(T->right!=NULL)
                { //delete its inorder succesor
                    p=T->right;

                    while(p->left!= NULL)
                        p=p->left;

                    T->data=p->data;
                    T->right=Delete(T->right,p->data);

                    if(BF(T)==2)//Rebalance during windup
```

```
                    if(BF(T->left)>=0)
                        T=LL(T);
                    else
                        T=LR(T);\
                }
                else
                    return(T->left);
            }
    T->ht=height(T);
    return(T);
}

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

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

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

    if(lh>rh)
        return(lh);

    return(rh);
}

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

node * rotateleft(node *x)
{
```

```
    node *y;
    y=x->right;
    x->right=y->left;
    y->left=x;
    x->ht=height(x);
    y->ht=height(y);

    return(y);
}

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

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

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

    return(T);
}

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

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

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

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

    return(lh-rh);
}

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

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

OUTPUT:

```
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:1
Enter no. of elements:4
Enter tree data:7 12 4 9

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

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

1) Create:
2) Insert:
3) Delete:
4) Print:
5) Quit:
Enter Your Choice:3
Enter a data:7

1) Create:
2) Insert:
3) Delete:
4) Print:
5) Quit:
Enter Your Choice:4
Preorder sequence:
9(Bf=0)4(Bf=0)12(Bf=0)
Inorder sequence:
4(Bf=0)9(Bf=0)12(Bf=0)

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

RESULT:

Thus a C program to implement an AVL tree is written and executed successfully.

Ex.No.-8 HEAP USING PRIORITY QUEUE

AIM:

To write a C program to implement heap sort using priority queue.

ALGORITHM:

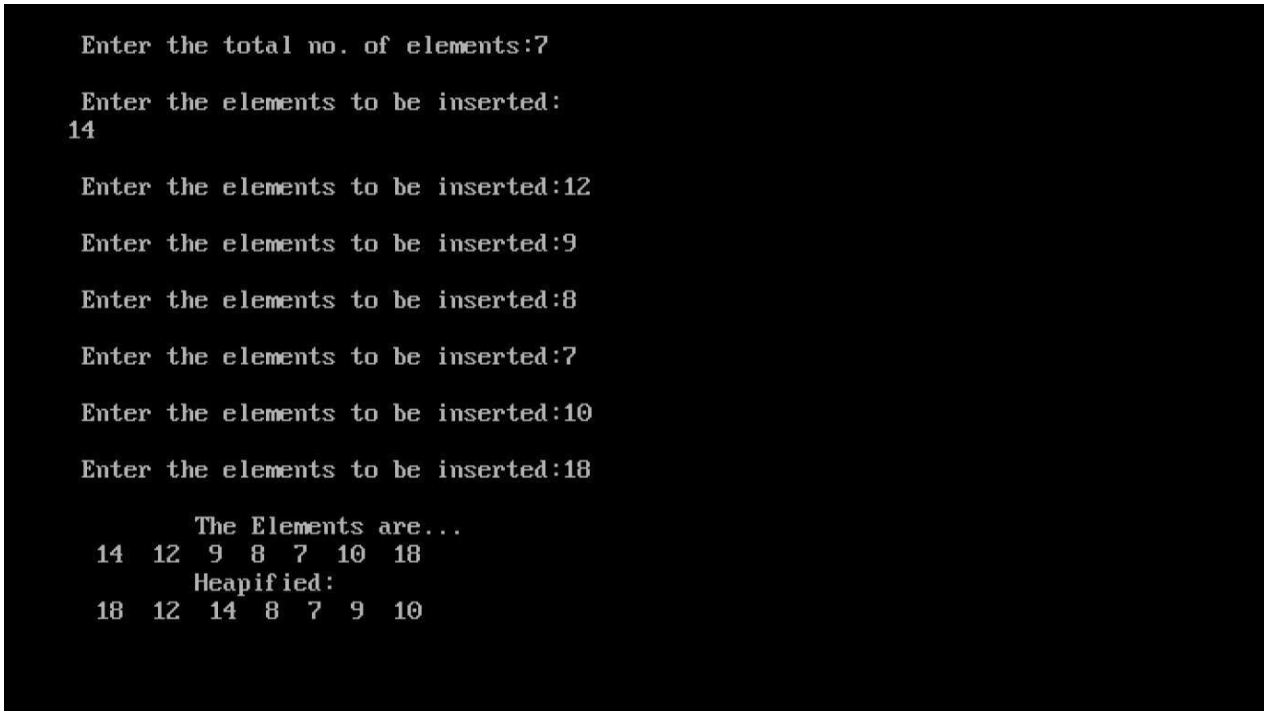
1. Start the program.
2. Read the elements to be inserted in heap.
3. Insert the element one by one.
4. Construct the heap structure to satisfy heap property of maxheap.
5. Display the heapified structure.
6. Stop the program.

PROGRAM:

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

# define MAX 10

int arr[MAX];
int i,item,n;

void insert(int num)
{
        if(i<MAX)
        {
                arr[i]=num;
        }
        else
                printf("\n Array is full");
}

void makeheap()
{
        for(i=0;i<n;i++)
        {
                int val=arr[i];
                int j=i;
                int f=(j-1)/2;
```

```
                while(j>0 && arr[f] < val)
                {
                        arr[j]=arr[f];
                        j=f;
                        f=(j-1)/2;
                }
                arr[j]=val;
        }
}

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

int main()
{
        clrscr();
        printf("\n Enter the total no. of elements:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
        printf("\n Enter the elements to be inserted:");
        scanf("%d",&item);
        insert(item);
        }
        printf("\n\t The Elements are...");
        display();
        makeheap();
        printf("\n\t Heapified:");
        display();
        getch();
        return 0;
}
```

OUTPUT:



RESULT:

Thus a C program to implement heap sort using priority queue is written and executed successfully.

Ex.No.-9 GRAPH TRAVERSAL

AIM:

To write a C program to implement graph traversals by Breadth First Search and Depth First Search.

ALGORITHM:

Breadth First Search:

1. Start the program.
2. Read the number of vertices and adjacency matrix.
3. Read the vertex from which to traverse the graph.
4. Initialize the visited array to 1 and insert the visited vertex in the queue.
5. Visit the vertex which is at the front of the queue.
6. Delete it from the queue and place its adjacent nodes in the queue.
7. Repeat the steps 5 & 6 , till the queue is not empty.
8. Display the traversal path.
9. Stop the program.

Depth First Search:

1. Start the program.
2. Read the number of vertices and adjacency matrix.
3. Initialize the visited array to 1.
4. Traverse the path one by one and push the visited vertex in the stack.
5. When there is no vertex further, we traverse back and search for unvisited vertex.
6. Display the traversal path.
7. Stop the program.

a) Breadth First Search

PROGRAM:

```
#include<stdio.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 nodes 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. Not all nodes are
reachable");
                        break;
                }
        }
}
```

OUTPUT:

Enter the number of vertices : 4

Enter graph data in matrix form:

1 1 1 1

0 1 0 0

0 0 1 0

0 0 0 1

Enter the starting vertex: 1

The nodes which are reachable are:

1 2 3 4

b) Depth First Search

PROGRAM:

```
#include<stdio.h>

void DFS(int);
int G[10][10],visited[10],n;     //n is no of vertices and graph is sorted in array G[10][10]

void main()
{
   int i,j;
   printf("Enter number of vertices:");

   scanf("%d",&n);

   //read the adjacency matrix
   printf("\nEnter adjecency matrix of the graph:");

   for(i=0;i<n;i++)
     for(j=0;j<n;j++)
         scanf("%d",&G[i][j]);

   //visited is initialized to zero
  for(i=0;i<n;i++)
      visited[i]=0;

   DFS(0);
}

void DFS(int i)
```

```
{
   int j;
   printf("\n%d",i);
   visited[i]=1;

   for(j=0;j<n;j++)
     if(!visited[j]&&G[i][j]==1)
         DFS(j);
}
```

OUTPUT:

```
Enter number of vertices : 8
Enter adjacency matrix of the graph :
0       1       1       1       1       0       0       0
1       0       0       0       0       1       0       0
1       0       0       0       0       1       0       0
1       0       0       0       0       0       1       0
1       0       0       0       0       0       1       0
0       1       1       0       0       0       0       1
0       0       0       1       1       0       0       1
0       0       0       0       0       1       1       0
0
1
5
2
7
6
3
4
```

RESULT:

Thus a C program to implement graph traversals by Breadth First Search and Depth First Search is written and executed successfully.

Ex.No.-10 TOPOLOGICAL SORTING

AIM:

To write a C program to perform topological sorting (Application of a graph).

ALGORITHM:

1. Start the program.
2. Read the number of vertices and adjacency matrix of a graph.
3. Find a vertex with no incoming edges.
4. Delete it along with all the edges outgoing from it.
5. If there are more than one such vertices then break the tie randomly.
6. Store the vertices that are deleted.
7. Display these vertices that give topologically sorted list.
8. Stop the program.

PROGRAM:

```
#include <stdio.h>
int main( )
{
   int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;
   printf("Enter the no of vertices:\n");
   scanf("%d",&n);
   printf("Enter the adjacency matrix:\n");
   for(i=0;i<n;i++){
      printf("Enter row %d\n",i+1);
      for(j=0;j<n;j++)
         scanf("%d",&a[i][j]);
 }
for(i=0;i<n;i++)
{
      indeg[i]=0;
      flag[i]=0;
 }
 for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         indeg[i]=indeg[i]+a[j][i];
    printf("\nThe topological order is:");
    while(count<n)
```

```
   {
    for(k=0;k<n;k++)
   {
          if((indeg[k]==0) && (flag[k]==0))
{
              printf("%d ",(k+1));
              flag [k]=1;
    }
    for(i=0;i<n;i++)
       {
              if(a[i][k]==1)
                 indeg[k]--;
       }
   }
        count++;
    }
     return 0;
}
```

OUTPUT :

Enter the no of vertices:
4
Enter the adjacency matrix:

Enter row 1
0 1 1 0
Enter row 2
0 0 0 1
Enter row 3
0 0 0 1
Enter row 4
0 0 0 0

The topological order is:1 2 3 4

RESULT:

Thus a C program to perform topological sorting is written and executed successfully.

Ex.No.-11a LINEAR & BINARY SEARCH

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: 3
```

RESULT:

Thus a C program to implement the linear search and binary search is written and executed successfully.

Ex.No.-11b 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

RESULT:

Thus a C program to perform insertion sort, quick sort and bubble sort is written and executed successfully.

Ex.No.-12 LINEAR PROBING

AIM:

To write a c program to create hash table and collision handling by linear probing.

ALGORITHM:

1. Start the program.
2. Read the numbers to be stored in hash table.
3. Create the hash function by generating the hash key.
4. If the location indicated by hash key is empty, then place the number in the hash table.
5. If collision occurs, then search for empty location.
6. If found, place the number at that location.
7. Display the hash table.

PROGRAM:

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
# define MAX 10

void main()
{
        int a[MAX],num,key,i;
        char ans;
        int create(int);
        void linearprob(int[], int,int),display(int[]);
        clrscr();

        printf("\nCOLLISION HANDLING BY LINEAR PROBING");
        for(i=0;i<MAX;i++)
        a[i]=-1;
        do
        {
                printf("\nEnter the number:");
                scanf("%d",&num);
                key=create(num);
                linearprob(a,key,num);
                printf("\n Do U wish to continue? (y/n)");
                ans=getche();
        }
                while(ans=='y');
```

```
                display(a);
                getch();
}

int create(int num)
{
        int key;
        key=num%10;
        return key;
}

void linearprob(int a[MAX],int key,int num)
{
        int flag,i,count=0;
        void display(int a[]);
        flag=0;
        if(a[key]==-1)
                a[key]=num;
        else
        {
                i=0;
                while(i<MAX)
                {
                        if(a[i]!=-1)
                                count++;
                                i++;
                }
                if(count==MAX)
                {
                        printf("\n Hash Table is full");
                        display(a);
                        getch();
                        exit(1);
                }
                for(i=key+1;i<MAX;i++)
                        if(a[i]==-1)
                        {
                                a[i]=num;
                                flag=1;
                                break;
                        }
                for(i=0;i<key&&flag==0;i++)
                        if(a[i]==-1)
                        {
                                a[i]=num;
                                flag=1;
                                break;
                        }
```

```
        }
}

void display(int a[MAX])
{
        int i;
        printf("\n Hash Table is..\n");
        for(i=0;i<MAX;i++)
        printf("\n %d  %d  ",i,a[i]);
}
```

OUTPUT:

COLLISION HANDLING BY LINEAR PROBING

Enter the number:131

Do U wish to continue? (y/n)y

Enter the number:21

Do U wish to continue? (y/n)y

Enter the number:3

Do U wish to continue? (y/n)y

Enter the number:4

Do U wish to continue? (y/n)y

Enter the number:8

Do U wish to continue? (y/n)y

Enter the number:9

Do U wish to continue? (y/n)y

Enter the number:18

Do U wish to continue? (y/n)n

Hash Table is..

0 18

1 131

2 21

3 3

4 4

5 5

6 -1

7 -1

8 8

9 9

RESULT:

Thus a C program to create hash table and collision handling by linear probing is written and executed successfully.