

Program-1: (Stack Implementation)

Code:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<ctype.h>
4  #define SIZE 5
5  int TOP=-1;
6
7  void operation(int[]);
8  void PUSH(int [],int);
9  int POP (int[]);
10 void DISPLAY(int[]);
11
12 int main()
13 {
14     int stack[SIZE];
15     operation(stack);
16     return 0;
17 }
18
19 void operation(int stack[])
20 {
21     int choice,ele,popele;
22     char flag ;
23
24     do{
25         printf("\nEnter The choice: \n\t1.PUSH\t2.POP\t3.DISPLAY\t4.EXIT\n\nChoice : ");
26         scanf("%d",&choice);
27         switch(choice)
28         {
29             case 1:if(TOP==SIZE-1)
30             {
31                 printf("\nThe Stack is full: OVERFLOW! Try other choices\n");
32                 operation(stack);
33             }
34             else
35             {
36                 printf("\nEnter The Element to be pushed: ");
37                 scanf("%d",&ele);
38                 PUSH(stack,ele);
39                 printf("\nThe Element %d was pushed successfully",ele);
40             }
41             break;
```

```
41             break;
42             case 2: if(TOP==1)
43             {
44                 printf("\nThe Stack is Empty: Underflow! Try other choices\n ");
45                 operation(stack);
46             }
47             else
48             {
49                 popele=stack[TOP];
50                 POP(stack);
51                 printf("\nThe Element %d was popped successfully ",popele);
52             }
53             break;
54             case 3:
55                 printf("\nThe Stack Elements are : \n");
56                 DISPLAY(stack);
57                 break;
58             case 4:return;
59             default:
60             {
61                 printf("\nInput choice ERROR! Try again! ");
62                 operation(stack);
63             }
64         }
65     }
66     printf("\n\nDo you wish to continue operations ? Press Y to continue : ");
67     fflush(stdin);
68     flag=getchar();
69     while(toupper(flag)!='Y');
70 }
71
72 void PUSH(int stack[],int ele)
73 {
74     TOP++;
75     stack[TOP]=ele;
76 }
77
78 int POP(int stack[])
79 {
80     int popele=stack[TOP];
```

```

80     int popele=stack[TOP];
81     TOP--;
82     return popele;
83 }
84 void DISPLAY(int stack[])
85 {
86     for(int i=TOP; i>=0; i--)
87         printf("\n\t%d", stack[i]);
88 }
89

```

Output:

```

Enter The choice:
    1.PUSH  2.POP  3.DISPLAY  4.EXIT
Choice : 1
Enter The Element to be pushed: 1
The Element 1 was pushed successfully
Do you wish to continue operations ? Press Y to continue : y
Enter The choice:
    1.PUSH  2.POP  3.DISPLAY  4.EXIT
Choice : 1
Enter The Element to be pushed: 2
The Element 2 was pushed successfully
Do you wish to continue operations ? Press Y to continue : y
Enter The choice:
    1.PUSH  2.POP  3.DISPLAY  4.EXIT
Choice : 1
The Stack is full: OVERFLOW! Try other choices
Enter The choice:
    1.PUSH  2.POP  3.DISPLAY  4.EXIT
Choice : 3
The Stack Elements are :
    3

```

```

The Stack Elements are :
    3
    2
    1
Do you wish to continue operations ? Press Y to continue : y
Enter The choice:
    1.PUSH  2.POP  3.DISPLAY  4.EXIT
Choice : 2
The Element 3 was popped successfully
Do you wish to continue operations ? Press Y to continue : y
Enter The choice:
    1.PUSH  2.POP  3.DISPLAY  4.EXIT
Choice : 2
The Element 2 was popped successfully
Do you wish to continue operations ? Press Y to continue : y
Enter The choice:
    1.PUSH  2.POP  3.DISPLAY  4.EXIT
Choice : 2
The Element 1 was popped successfully
Do you wish to continue operations ? Press Y to continue : y
Enter The choice:
    1.PUSH  2.POP  3.DISPLAY  4.EXIT
Choice : 2
The Stack is Empty: Underflow! Try other choices
Enter The choice:
    1.PUSH  2.POP  3.DISPLAY  4.EXIT
Choice : 4

```

Program-2: (Infix to PostFix)

Code:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #define size 50
4  int top=-1;
5  char stack[size];
6
7  //Expression Intake
8  void getData(char *infix)
9  {
10     printf("\nEnter The Infix Expression: ");
11     scanf("%s",infix);
12 }
13 //PUSH and POP operation
14 void PUSH(int item)
15 {
16     if(top==size-1)
17         printf("\nStack Overflow, the expression exceeds the stack limits!");
18     else
19         stack[++top]=item;
20 }
21 char POP()
22 {
23     if(top==-1)
24         printf("\nStack Underflow, Stack is Empty!");
25     else
26     {
27         int popItem=stack[top--];
28         return popItem;
29     }
30 }
31 //checking priority
32 int check(char ch)
33 {
34     switch(ch)
35     {
36         case '+':
37         case '-': return 1;
38         case '*':
39         case '/': return 2;
40         case '^': return 3;
41         default: return 0;
```

```
41     default: return 0;
42     }
43 }
44
45 void inputChecking(char *infix)
46 {
47     int operators=0,operand=0,brackets=0;
48     for(int i=0;infix[i]!='\0';i++)
49     {
50         switch(infix[i])
51         {
52             case '(': brackets++;
53             break;
54             case ')': brackets--;
55             break;
56             case '+':
57             case '-':
58             case '*':
59             case '^':
60             case '/': operators++;
61             break;
62             default:operand++;
63         }
64     }
65
66     if((++operators)==operand && brackets==0)
67         printf("\nThe given expression is valid.\n");
68     else
69     {
70         printf("\nThe given expression is Invalid, retry! \n");
71         main();
72     }
73 }
74
75
76
77
78
79
80 }
```

```
81
82 int main()
83 {
84     char infix[50],item,postFix[50];
85     int k=0;
86     getData(infix);
87     inputChecking(infix);
88     for(int i=0;infix[i]!='\0';i++)
89     {
90         switch(infix[i])
91         {
92             case '(': PUSH(infix[i]);
93             break;
94             case ')': while((item=POP())!='(')
95                     postFix[k++]=item;
96                     break;
97             case '+':
98             case '-':
99             case '*':
100             case '/':
101             case '^':
102                 while(check(stack[top])>=check(infix[i]))
103                 {
104                     item=POP();
105                     postFix[k++]=item;
106                 }
107                 PUSH(infix[i]);
108                 break;
109             default : postFix[k++]=infix[i];
110                     break;
111         }
112     }
113
114
115
116
117
118
119 }
```

```

120 while(top!=-1)
121 {
122     item=POP();
123     postfix[k++]=item;
124 }
125 postfix[k++]='\0';
126 printf("\n\nThe PostFix Expression is : %s",postfix);
127 return 0;
128 }
129

```

Output:

Enter The Infix Expression: 2*(3/(4-1))

The PostFix Expression is : 2341-/*

Process returned 0 (0x0) execution time : 8.314 s

Press any key to continue.

Enter The Infix Expression: a*(b+c^d-p/r+(t-s))

The PostFix Expression is : abcd^+pr/-ts-+*

Process returned 0 (0x0) execution time : 37.303 s

Press any key to continue.

Program-3:

(Queue Implementation)

Code:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #define MAX 5
4
5
6
7  void operation(int *,int *,int *);
8  void enqueue(int *,int *,int *);
9  int dequeue(int *,int *,int *);
10 void displayQueue(int *,int *,int *);
11
12 int main()
13 {
14     int front=0,rear=-1;
15     int queue[MAX];
16     operation(queue,&rear,&front);
17     return 0;
18 }
19
20 void operation(int *queue,int *rear,int *front)
21 {
22     int choice,element;
23     do
24     {
25         element=0;
26         choice=0;
27         printf("\n\n Enter the Choice : \n\t\t 1.Enqueue\n\t\t 2.Dequeue\n\t\t 3.Display\n\t\t 4.Quit\n Choice: ");
28         scanf("%d",&choice);
29         switch(choice)
30         {
31             case 1:
32                 if(*rear==MAX-1)
33                     printf("\n Queue Overflow!,Can't Enqueue, try other option!\n");
34                 else
35                 {
36                     printf("\n Enter The element to Enqueue: ");
37                     scanf("%d",&element);
38                     enqueue(element,queue,rear);
39                 }
40                 break;
41             case 2:
```

```
42                 if(*rear ==-1)
43                     printf("\n Queue Underflow!,Can't Dequeue, try other option!\n");
44                 else
45                     printf("\n %d was Dequeued from Queue \n",dequeue(queue,rear,front));
46                 break;
47             case 3:
48                 displayQueue(queue,rear,front);
49                 break;
50             case 4:
51                 exit(0);
52             default:
53                 {
54                     element=0;
55                     printf("\n Input Error!, press 1 to try again ! :");
56                     fflush(stdin);
57                     scanf("%d",&element);
58                     if(element==1)
59                         operation(queue,rear,front);
60                     else
61                         exit(0);
62                 }
63             }
64         }
65     }while(1);
66 }
67
68 void enqueue(int ele,int *queue,int *rear)
69 {
70     queue[++(*rear)]=ele;
71 }
72
73 int dequeue(int *queue,int *rear,int *front)
74 {
75     int ele=queue[*front];
76     if((*rear)==(*front))
```

```
77     {
78         if((*rear)==(*front))
79         {
80             *rear=-1;
81             *front=0;
82         }
83         else
84         {
85             *front++;
86             return ele;
87         }
88     }
89
90 void displayQueue(int *queue,int *rear,int *front)
91 {
92     if(*rear== -1)
93         printf("\n The Queue is Empty! \n");
94     else
95     {
96         printf("\n The Queue is : ");
97         for(int i=front;i<=rear;i++)
98             printf(" %d ",queue[i]);
99     }
100 }
101
102
103
```

Output:

```
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 1

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 2

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 3

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 4

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
```

(a)

```
3.Display
4.Quit
Choice: 1
Enter The element to Enqueue: 5

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 1
Queue Overflow,Can't Enqueue, try other option!

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 1 2 3 4 5

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
1 was Dequeued from Queue

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 2 3 4 5

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
```

(b)

```
The Queue is : 2 3 4 5
Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
2 was Dequeued from Queue

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 3 4 5

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
3 was Dequeued from Queue

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 4 5

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
4 was Dequeued from Queue
```

(c)

```
4.Quit
Choice: 2
4 was Dequeued from Queue

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is : 5

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
5 was Dequeued from Queue

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is Empty!

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
Queue Underflow,Can't Dequeue, try other option!

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
```

(d)

```
Choice: 2
Queue Underflow,Can't Dequeue, try other option!

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 3
The Queue is Empty!

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 2
Queue Underflow,Can't Dequeue, try other option!

Enter the Choice :
1.Enqueue
2.Dequeue
3.Display
4.Quit
Choice: 4

Process returned 0 (0x0)   execution time : 26.285 s
Press any key to continue.
```

(e)

Program-4:

(CircularQueue Implementation)

Code:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX 3
4
5  int front=-1;
6  int rear=-1;
7
8  int queue[MAX];
9
10 void Enque(int);
11 int Deque();
12 void display();
13 void operation();
14
15 int main()
16 {
17     printf("\n <----Circular Queue---->\n");
18     operation();
19     return 0;
20 }
21
22 void operation()
23 {
24     int option=0;
25     int item;
26     do;
27     {
28         printf("\n 1. Insert to Queue (EnQueue)");
29         printf("\n 2. Delete from the Queue (DeQueue)");
30         printf("\n 3. Display the content ");
31         printf("\n 4. Exit \n");
32         printf("\n Enter the option :");
33         scanf("%d",&option);
34         switch(option)
35         {
36             case 1: if(front==(rear+1)%MAX)
37                     {
38                         printf("\n Queue Overflow! Try other Options");
39                         operation();
40                     }
41             else
```

```

42             {
43                 printf("\n Enter the element: ");
44                 scanf("%d",&item);
45                 Enque(item);
46                 break;
47             }
48             case 2: if(front==0 && rear==0)
49                     {
50                         printf("\n Queue Underflow, It is empty! Try other Options");
51                         operation();
52                     }
53             else
54             {
55                 item=Deque();
56                 printf("\n Removed element from the queue is %d \n",item);
57                 break;
58             }
59             case 3: display();
60                     break;
61             case 4: exit(0);
62             default: printf("\n Input Error! Try Again \n");
63                     operation();
64             }
65         } while (1);
66     }
67 }
68
69 void Enque(int ele)
70 {
71     rear=(rear+1)%MAX;
72     queue[rear]=ele;
73     if(front ==-1)
74         front=-1;
75 }
76
77 int Deque()
78 {
79     int item;
80     item=queue[front];
```

```

81     if(front==rear)
82     {
83         front=-1;
84         rear=-1;
85     }
86     else
87     {
88         front=(front+1)%MAX;
89         return item;
90     }
91 }
92
93 void display()
94 {
95     int i;
96     if (front==0 && rear==0)
97     {
98         printf("\n Queue is empty! Try other Options ");
99         operation();
100     }
101     else
102     {
103         printf("\n Queue contents are: ");
104         if(rear<front)
105         {
106             for(i=front;i<rear;i=(i+1)%MAX)
107                 printf("%d ", queue[i]);
108             printf("%d ", queue[i]);
109         }
110         else
111         {
112             if(rear<front)
113             {
114                 for(i=0;i<rear;i++)
115                     printf("%d ", queue[i]);
116                 for(i=front;i<MAX;i++)
117                     printf("%d ", queue[i]);
118             }
119         }
120     }
121 }
```

Output:

```
<-----Circular Queue----->
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :1

Enter the element: 1

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :1

Enter the element: 2

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :1

Enter the element: 3

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :1

Queue Overflow! Try other Options
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :3

Queue contents are: 1 2 3
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
```

(a)

```
Enter the option :3

Queue contents are: 1 2 3
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :2

Removed element from the queue is 1

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :3

Queue contents are: 2 3
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :2

Removed element from the queue is 2

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :3

Queue contents are: 3
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :2

Removed element from the queue is 3

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
```

(b)

```
Enter the option :2

Removed element from the queue is 3

1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :3

Queue is empty! Try other Options
1. Insert to Queue (EnQueue)
2. Delete from the Queue (DeQueue)
3. Display the content
4. Exit

Enter the option :4

Process returned 0 (0x0)   execution time : 17.612 s
Press any key to continue.
```

(c)

Program-5:

(LinkedList Implementation Insertion)

Code:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct node
5  {
6      int id;
7      char name[20];
8      int sem;
9      struct node *next;
10 };
11
12 struct node *head=NULL;
13
14 void linkedList();
15 void insertNodeAtBegin();
16 void insertNodeAtEnd();
17 void insertNodeAtAny();
18 void displayList();
19
20 int size=0;
21
22 int main()
23 {
24     linkedList();
25     return 0;
26 }
27
28 void linkedList()
29 {
30     int choice1,choice2;
31     printf("\n <--Enter the Operation--->");
32     printf("\n 1.Insert Node. \n 2.Display List \n 3.Exit.\n Choice: ");
33     scanf("%d",&choice1);
34     switch(choice1)
35     {
36         case 1: printf("\n 1.At the First Position \t 2.At End of list\t 3.At Any Specified Location\n Choice:");
37                 scanf("%d",&choice2);
38                 switch(choice2)
39                 {
40                     case 1: insertNodeAtBegin();
41                         break;
```

```
42                     case 2: insertNodeAtEnd();
43                         break;
44                     case 3: insertNodeAtAny();
45                         break;
46                     default: printf("\n Input Error, try Again!\n ");
47                             linkedList();
48                 }
49             }
50             break;
51         case 2: displayList();
52             break;
53         case 3: exit(0);
54             break;
55         default: printf("\n Input error, Try again!!\n");
56                 linkedList();
57             }
58     }
59
60 void insertNodeAtBegin()
61 {
62     struct node *newnode;
63     newnode=(struct node*)malloc(sizeof(struct node));
64     printf("\n <--Enter the Details--> ");
65     printf("\n ID: "); scanf("%d",&(newnode->id));
66     printf(" Name: "); scanf("%s",&(newnode->name));
67     printf(" Sem: "); scanf("%d",&(newnode->sem));
68     newnode->next=head;
69     head=newnode;
70     size++;
71     printf("\n Node created \n");
72     linkedList();
73 }
74
75 void insertNodeAtEnd()
76 {
77     struct node *newnode,*temp;
78     newnode =(struct node *) malloc (sizeof(struct node));
79     printf("\n <--Enter the Details--> ");
80     printf("\n ID: "); scanf("%d",&(newnode->id));
81     printf(" Name: "); scanf("%s",&(newnode->name));
82     printf(" Sem: "); scanf("%d",&(newnode->sem));
```

```
83     if (head==NULL)
84     {
85         newnode->next=NULL;
86         head=newnode;
87         printf("Node created\n");
88         linkedList();
89     }
90     for(temp=head;(temp->next)!=NULL;temp=(temp->next));
91     temp->next=newnode;
92     size++;
93     printf("\n Node created \n");
94     linkedList();
95 }
96
97 void insertNodeAtAny()
98 {
99     struct node *newnode,*temp,*head;
100     newnode=(struct node*)malloc(sizeof(struct node));
101     printf("\n <--Enter the Details--> ");
102     printf("\n ID: "); scanf("%d",&(newnode->id));
103     printf(" Name: "); scanf("%s",&(newnode->name));
104     printf(" Sem: "); scanf("%d",&(newnode->sem));
105     int pos=0;
106     printf("\n Enter the position(pos=1 and pos <id) : ",size);
107     scanf("%d",&pos);
108     if(pos==0)
109     {
110         printf("\n Error position, check the operation menu!");
111         linkedList();
112     }
113     for(temp=head;temp->next!=NULL;temp=temp->next)
114     {
115         if(s==pos-1)
116         {
```

```

120         newnode->next=(temp->next);
121         temp->next=newnode;
122         size++;
123         printf("\n Node created \n");
124         linkedList();
125     }
126     s++;
127     temp=temp->next;
128 }
129
130 void displayList()
131 {
132     if(head==NULL)
133     {
134         printf("\n Empty List!\n");
135         linkedList();
136     }
137     printf("\n The List is :");
138     for(struct node *temp=head;temp!=NULL;temp=temp->next)
139     {
140         printf("\n ---Student Details---");
141         printf("\n ID: %d ",temp->id);
142         printf("\n Name: %s ",temp->name);
143         printf("\n Sem: %d",temp->sem);
144     }
145     linkedList();
146 }
147
148

```

Output:

```

<---Enter the Operation---
1.Insert Node.
2.Display List
3.Exit.
Choice: 1

1.At the First Position      2.At End of list      3.At Any Specified Location
Choice:1

<--Enter the Details-->
ID: 1
Name: aaa
Sem: 1

Node created

<---Enter the Operation---
1.Insert Node.
2.Display List
3.Exit.
Choice: 1

1.At the First Position      2.At End of list      3.At Any Specified Location
Choice:1

<--Enter the Details-->
ID: 2
Name: bbb
Sem: 2

Node created

<---Enter the Operation---
1.Insert Node.
2.Display List
3.Exit.
Choice: 1

1.At the First Position      2.At End of list      3.At Any Specified Location
Choice:2

<--Enter the Details-->
ID: 3
Name: ccc
Sem: 3

Node created

```

(a)

```

<---Enter the Operation---
1.Insert Node.
2.Display List
3.Exit.
Choice: 1

1.At the First Position      2.At End of list      3.At Any Specified Location
Choice:1

<--Enter the Details-->
ID: 4
Name: ddd
Sem: 4

Node created

<---Enter the Operation---
1.Insert Node.
2.Display List
3.Exit.
Choice: 2

The List is :
<--Student Details--->
ID: 4
Name: ddd
Sem: 4
<--Student Details--->
ID: 2
Name: bbb
Sem: 2
<--Student Details--->
ID: 1
Name: aaa
Sem: 1
<--Student Details--->
ID: 3
Name: ccc
Sem: 3
<---Enter the Operation---
1.Insert Node.
2.Display List
3.Exit.
Choice: 1

```

(b)

```

1.At the First Position      2.At End of list      3.At Any Specified Location
Choice:3

<--Enter the Details-->
ID: 5
Name: eee
Sem: 5

Enter the position(pos)=1 and pos <4) : 2

Node created

<---Enter the Operation---
1.Insert Node.
2.Display list
3.Exit.
Choice: 2

The List is :
<--Student Details--->
ID: 4
Name: ddd
Sem: 4
<--Student Details--->
ID: 2
Name: bbb
Sem: 2
<--Student Details--->
ID: 1
Name: aaa
Sem: 1
<--Student Details--->
ID: 5
Name: eee
Sem: 5
<--Student Details--->
ID: 3
Name: ccc
Sem: 3
<---Enter the Operation---
1.Insert Node.
2.Display list
3.Exit.
Choice: 3

Process returned 0 (0x0)   execution time : 156.461 s
Press any key to continue.

```

(c)

Program-6:

(LinkedList Implementation Deletion)

Code:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct node
5  {
6      int id;
7      char name[20];
8      int sem;
9      struct node *next;
10 }
11
12 struct node *head=NULL;
13
14 void linkedList();
15 void insertNode();
16 void deleteNode(int);
17 void deleteNodeAtBegin();
18 void deleteNodeAtEnd();
19 void deleteNodeOfGiven();
20 void displayList();
21
22 int size=0;
23
24 int main()
25 {
26     linkedList();
27     return 0;
28 }
29
30 void linkedList()
31 {
32     int choice1,choice2;
33     printf("\n\n <--Enter the Operation-->");
34     printf("\n 1.Insert Node. \n 2.Delete Node \n 3.Display List\n 4.Exit.\n Choice: ");
35     scanf("%d",&choice1);
36     switch(choice1)
37     {
38         case 1:
39             insertNode();
40             break;
41         case 2:
```

```
42         case 2: printf("\n 1.At the First Position \t 2.At End of list\t 3.At Given Element Position\n Choice:");
43                 scanf("%d",&choice2);
44                 switch(choice2)
45                 {
46                     case 1: deleteNode(1);
47                             break;
48                     case 2: deleteNode(2);
49                             break;
50                     case 3: deleteNode(3);
51                             break;
52                     default: printf("\n Input Error, try Again!\n ");
53                             linkedList();
54                 }
55                 break;
56
57         case 3: displayList();
58                 break;
59
60         case 4: exit(0);
61
62         default: printf("\n Input error, Try again!\n");
63                 linkedList();
64     }
65 }
66
67 void insertNode()
68 {
69     struct node *newnode,*temp;
70     newnode =(struct node *) malloc (sizeof(struct node));
71     printf("\n <--Enter the Details--> ");
72     printf("\n ID: "); scanf("%d",&(newnode->id));
73     printf("\n Name: "); scanf("%s",(newnode->name));
74     printf("\n Sem: "); scanf("%d",&(newnode->sem));
75     if (head==NULL)
76     {
77         newnode->next=NULL;
78         head=newnode;
79         printf("\n Node created\n");
80     }
81     else
```

```
81     linkedList();
82     size++;
83     for(temp=head;(temp->next)!=NULL;temp=(temp->next));
84     newnode->next=NULL;
85     {
86         temp->next=newnode;
87         size++;
88         printf("\n Node created \n");
89         linkedList();
90     }
91 }
92
93 void deleteNode(int flag)
94 {
95     if(head==NULL)
96     {
97         printf("\n The List is Empty! \n");
98         linkedList();
99     }
100     else
101     if(head->next==NULL)
102     {
103         printf("\n Node Deleted. \n Now List Is empty! ");
104         free(head);
105         head=NULL;
106         linkedList();
107     }
108     else
109     {
110         switch(flag)
111         {
112             case 1: deleteNodeAtBegin();
113                     break;
114             case 2: deleteNodeAtEnd();
115                     break;
116             case 3: deleteNodeOfGiven();
```

```

119         case 3: deleteNodeOfGiven();
120             break;
121     }
122     }
123     linkedList();
124 }
125
126
127 void deleteNodeAtBegin()
128 {
129     struct node* temp=head;
130     head=head->next;
131     free(temp);
132     printf("\n Node Deleted.");
133 }
134 void deleteNodeAtEnd()
135 {
136     struct node* temp=head;
137     for(temp;(temp->next)->next!=NULL;temp=temp->next);
138     free(temp->next);
139     temp->next=NULL;
140     printf("\n Node Deleted.");
141 }
142 void deleteNodeOfGiven()
143 {
144     struct node* temp1=head,*temp2=temp1;
145     int ele;
146     printf("\n Enter the Element :");
147     scanf("%d",&ele);
148     for(temp1,temp1->next!=NULL;temp1=temp1->next)
149     {
150         temp2=temp1->next;
151         if(temp2->id==ele)
152         {
153             temp1->next=temp2->next;
154             free(temp2);
155             printf("\n Node Deleted.");
156             return;
157         }
158     }
    if(temp1->id==ele)

```

```

158     {
159         head=temp2;
160         free(temp1);
161         printf("\n Node Deleted.");
162         return;
163     }
164 }
165 printf("\n Element Not Present In List! ");
166 linkedList();
167
168
169 void displayList()
170 {
171     if(head==NULL)
172     {
173         printf("\n Empty List!\n");
174         linkedList();
175     }
176     printf("\n The List is :");
177     for(struct node *temp=head;temp!=NULL;temp=temp->next)
178     {
179         printf("\n\n <---Student Details--->");
180         printf("\n ID: %d ",temp->id);
181         printf("\n Name: %s ",temp->name);
182         printf("\n Sem: %d",temp->sem);
183         printf("\n <-----X----->");
184     }
185     linkedList();
186 }
187

```

Output:

```

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 1

<--Enter the Details-->
ID: 1
Name: 1
Sem: 1

Node created

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 1

<--Enter the Details-->
ID: 2
Name: 2
Sem: 2

Node created

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 1

<--Enter the Details-->
ID: 3
Name: 3
Sem: 3

Node created

<---Enter the Operation--->
1.Insert Node.

```

(a)

```

Sem: 3
Node created

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 1

<--Enter the Details-->
ID: 4
Name: 4
Sem: 4

Node created

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 3

The List is :

<---Student Details--->
ID: 1
Name: 1
Sem: 1
<-----X----->

<---Student Details--->
ID: 2
Name: 2
Sem: 2
<-----X----->

<---Student Details--->
ID: 3
Name: 3
Sem: 3
<-----X----->

<---Student Details--->
ID: 4

```

(b)

```

ID: 4
Name: 4
Sem: 4
<-----X----->

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 2

1.At the First Position      2.At End of list      3.At Given Element Position
Choice:1

Node Deleted.

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 3

The List is :

<---Student Details--->
ID: 2
Name: 2
Sem: 2
<-----X----->

<---Student Details--->
ID: 3
Name: 3
Sem: 3
<-----X----->

<---Student Details--->
ID: 4
Name: 4
Sem: 4
<-----X----->

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.

```

(c)

```

4.Exit.
Choice: 2

1.At the First Position      2.At End of list      3.At Given Element Position
Choice:2

Node Deleted.

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 3

The List is :

<---Student Details--->
ID: 2
Name: 2
Sem: 2
<-----X----->

<---Student Details--->
ID: 3
Name: 3
Sem: 3
<-----X----->

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 2

1.At the First Position      2.At End of list      3.At Given Element Position
Choice:3

Enter the Element :4

Element Not Present In List!

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 2

```

(d)

```

4.Exit.
Choice: 2

1.At the First Position      2.At End of list      3.At Given Element Position
Choice:3

Enter the Element :3

Node Deleted.

<---Enter the Operation--->
1.Insert Node.
2.Delete Node
3.Display List
4.Exit.
Choice: 3

The List is :

<---Student Details--->
ID: 2
Name: 2
Sem: 2
<-----X----->

```

(e)

Program-7:

(LinkedList Operations)

Code:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct node
5  {
6      int data;
7      struct node *next;
8  };
9
10 struct node *head1=NULL,*head2=NULL;
11
12 void linkedList();
13 int choiceFlag();
14 void insert(int);
15 void display(int);
16 void reverse(int);
17 void sorting(int);
18 void swap(struct node*,struct node*);
19 void concatenate();
20
21 int main()
22 {
23     linkedList();
24     return 1;
25 }
26
27 void linkedList()
28 {
29     int choice1,choice2;
30     printf("\n 1. Insert \n 2. Display \n 3. Reverse \n 4. Sort \n 5. Concatenate and display \n 6. Exit");
31     printf("\n Enter your choice : ");
32     scanf("%d",&choice1);
33     switch(choice1)
34     {
35         case 1: choice2=choiceFlag();insert(choice2); break;
36         case 2: choice2=choiceFlag();display(choice2);break;
37         case 3: choice2=choiceFlag();reverse(choice2);break;
38         case 4: choice2=choiceFlag();sorting(choice2);break;
39         case 5: concatenate();break;
40         case 6: break;
41         case 0:exit(0);
42     }
```

```
42     default:printf("\n Try Again!\n");
43 }
44 linkedList();
45 }
46
47 int choiceFlag()
48 {
49     int flag=0;
50     do:
51     {
52         printf("\n 1.First List\n 2.Second List\n Choice: ");
53         scanf("%d",&flag);
54         if(flag!=1 && flag!=2)
55             printf("\n Error choice, Try again! ");
56     } while(flag!=1 && flag!=2);
57     return flag;
58 }
59
60 void insert(int flag)
61 {
62     struct node *newnode,*temp,**head;
63     int item;
64     newnode =(struct node *) malloc (sizeof(struct node));
65     printf("\n Enter the data : ");
66     scanf("%d",&item);
67     newnode->data=item;
68     newnode->next=NULL;
69
70     if(flag==1)
71         head=&head1;
72     else
73         head=&head2;
74
75     if (*head==NULL)
76     {
77         *head=newnode;
78         printf("\n First Node created\n");
79     }
```

```
80     printf("\n First Node created\n");
81 }
82 else
83 {
84     temp=*head;
85     while(temp->next!=NULL)
86     {
87         temp=temp->next;
88     }
89     temp->next=newnode;
90     printf("\n Next Node created\n");
91 }
92
93 void reverse(int flag )
94 {
95     struct node **head;
96
97     if(flag==1)
98         head=&head1;
99     else
100        head=&head2;
101
102     if(*head==NULL)
103     {
104         printf("\n List Empty! \n");
105         return;
106     }
107
108     struct node *pre=NULL,*forw=NULL,*cur=*head;
109     while(cur!=NULL)
110     {
111         forw=cur->next;
112         cur->next=pre;
113         pre=cur;
114         cur=forw;
115     }
```

```

119     cur=cur->next;
120 }
121 *head-pre;
122 printf("\n Reversed! \n");
123 }
124 void sorting(int flag)
125 {
126     struct node **head;
127     int swapflag;
128     if(flag==1)
129         head=&head1;
130     else
131         head=&head2;
132     if(*head==NULL)
133     {
134         printf("\n List Empty! \n");
135         return;
136     }
137     struct node *temp1=*head,*temp2=NULL;
138     do
139     {
140         temp1 = *head;
141         swapflag = 0;
142         while (temp1->next!=temp2)
143         {
144             if (temp1->data > (temp1->next->data))
145             {
146                 swap(temp1, temp1->next);
147                 swapflag=1;
148             }
149             temp1=temp1->next;
150         }
151         temp2=temp1;
152     } while (swapflag);
153     printf("\n Sorted! \n");
154 }
155 }
156 }
157 }
158 }

```

```

159 void swap(struct node *t1, struct node *t2)
160 {
161     int temp = t1->data;
162     t1->data = t2->data;
163     t2->data = temp;
164 }
165 void concat()
166 {
167     struct node *temp1=head1,*temp2=head2;
168     while(temp1->next!=NULL)
169     {
170         temp1=temp1->next;
171         temp2=temp2->next;
172         display(0);
173     }
174 }
175 void display(int flag)
176 {
177     struct node *ptr;
178     if(flag==1)
179         ptr=head1;
180     else
181         ptr=head2;
182     if(ptr==NULL)
183     {
184         printf("\n List Empty! \n");
185     }
186     else
187     {
188         printf("\n");
189         while(ptr!=NULL)
190         {
191             printf(" %d ",ptr->data);
192             ptr=ptr->next;
193         }
194     }
195 }
196 }
197 }
198 }
199 }

```

Output:

```

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 1
Enter the data : 1

First Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 1
Enter the data : 2

Next Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 1
Enter the data : 3

Next Node created

1. Insert
2. Display
3. Reverse

```

(a)

```

3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 2
Enter the data : 9

Next Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 2

1.First List
2.Second List
Choice: 1

1 2 3 4
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 2

1.First List
2.Second List
Choice: 2

6 2 9
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 3

1.First List
2.Second List

```

(b)

```

3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 1
Enter the data : 4

Next Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 2
Enter the data : 6

First Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 1

1.First List
2.Second List
Choice: 2
Enter the data : 2

Next Node created

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit

```

(c)

```

2.Second List
Choice: 1

Reversed!

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 2

1.First List
2.Second List
Choice: 1

4 3 2 1
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 4

1.First List
2.Second List
Choice: 1

Sorted!

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 5

1 2 3 4 6 2 9
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 4

```

(d)

```

5. Concatenate and display
6. Exit
Enter your choice : 5

1 2 3 4 6 2 9
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 4

1.First List
2.Second List
Choice: 1

Sorted!

1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 2

1.First List
2.Second List
Choice: 1

1 2 2 3 4 6 9
1. Insert
2. Display
3. Reverse
4. Sort
5. Concatenate and display
6. Exit
Enter your choice : 6

Process returned 0 (0x0)   execution time : 69.901
Press any key to continue.

```

(d)

Program-8:

(LinkedList_Stack-Queue_Implementation)

Code:

Stack:

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 struct node *top=NULL;
4
5 struct node
6 {
7     int data;
8     struct node *next;
9 };
10
11 void stack();
12 void push(int);
13 int pop();
14 void display();
15
16 int main()
17 {
18     stack();
19     return 1;
20 }
21
22 void stack()
23 {
24     int choice=0,ele=0;
25     do
26     {
27         printf("\n Enter the choice:\n 1.Push,\n 2.Pop,\n 3.Display,\n 4.Exit\n Choice:");
28         scanf("%d",&choice);
29         switch(choice)
30         {
31             case 1:
32                 printf("\n Enter the element to Push: ");
33                 scanf("%d",&ele);
34                 push(ele);
35                 break;
36             case 2:
37                 ele=pop();
38                 printf("\n %d was deleted from Queue ",ele);
39                 break;
40             case 3:
41                 display();
42                 break;
43             case 4:
44                 exit(0);
45             default:
46                 printf("\n Input Error Try Again! ");
47                 stack();
48         }
49     }while(1);
50 }
51
52 void push(int ele)
53 {
54     struct node *newnode;
55     newnode =(struct node *) malloc (sizeof(struct node));
56     newnode->data=ele;
57     newnode->next=NULL;
58     if(top==NULL)
59     {
60         top=newnode;
61     }
62     else
63     {
64         newnode->next=top;
65         top=newnode;
66     }
67     printf("\n Element %d was inserted!\n ",ele);
68 }
69
70 int pop()
71 {
72     int ele;
73     struct node *temp;
74     if(top==NULL)
75     {
76         printf("\n Stack Underflow, The Stack is empty!\n ");
77         stack();
78     }
79 }
80 }
```

```
41
42     display();
43     break;
44 case 4:
45     exit(0);
46 default:
47     printf("\n Input Error Try Again! ");
48     stack();
49 }
50 }while(1);
51 }
52
53 void push(int ele)
54 {
55     struct node *newnode;
56     newnode =(struct node *) malloc (sizeof(struct node));
57     newnode->data=ele;
58     newnode->next=NULL;
59     if(top==NULL)
60     {
61         top=newnode;
62     }
63     else
64     {
65         newnode->next=top;
66         top=newnode;
67     }
68     printf("\n Element %d was inserted!\n ",ele);
69 }
70
71 int pop()
72 {
73     int ele;
74     struct node *temp;
75     if(top==NULL)
76     {
77         printf("\n Stack Underflow, The Stack is empty!\n ");
78         stack();
79     }
80 }
```

```
81     ele=top->data;
82     temp=top;
83     if(top->next==NULL)
84     {
85         top=NULL;
86     }
87     else
88     {
89         top=top->next;
90         free(temp);
91         return ele;
92     }
93 }
94 void display()
95 {
96     struct node *i;
97     if(top==NULL)
98     {
99         printf("\n The Stack is empty!\n ");
100         stack();
101     }
102     printf("\n The Stack Contains:\n TOP->");
103     for(i=top;i!=NULL;i=i->next)
104         printf("%d ",i->data);
105 }
```

Queue:

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 struct node *front=NULL, *rear=NULL;
4
5 struct node
6 {
7     int data;
8     struct node *next;
9 };
10
11 void queue();
12 void insNode(int);
13 int delNode();
14 void display();
15
16 int main()
17 {
18     queue();
19     return 1;
20 }
21
22 void queue()
23 {
24     int choice=0,ele=0;
25     do
26     {
27         printf("\n Enter the choice:\n 1.Insert,\n 2.Delete,\n 3.Display,\n 4.Exit\n Choice:");
28         scanf("%d",&choice);
29         switch(choice)
30         {
31             case 1:
32                 printf("\n Enter the element to Insert: ");
33                 scanf("%d",&ele);
34                 insNode(ele);
35                 break;
36             case 2:
37                 ele=delNode();
38                 printf("\n %d was deleted from Queue ",ele);
39                 break;
40             case 3:
41                 display();
42                 break;
43             case 4:
44                 exit(0);
45             default:
46                 printf("\n Input Error Try Again! ");
47                 queue();
48         }
49     }while(1);
50 }
51
52 void insNode(int ele)
53 {
54     struct node *newnode;
55     newnode =(struct node *) malloc (sizeof(struct node));
56     newnode->data=ele;
57     newnode->next=NULL;
58     if(rear==NULL)
59     {
60         front=newnode;
61         rear=newnode;
62     }
63     else
64     {
65         rear->next=newnode;
66         rear=newnode;
67     }
68     printf("\n Element %d was inserted!\n ",ele);
69 }
70
71 int delNode()
72 {
73     int ele;
74     struct node *temp;
75     if(front==NULL)
76     {
77         printf("\n Queue Underflow, The queue is empty!\n ");
78         queue();
79     }
80 }
```

```
41
42     display();
43     break;
44 case 4:
45     exit(0);
46 default:
47     printf("\n Input Error Try Again! ");
48     queue();
49 }
50 }while(1);
51 }
52
53 void insNode(int ele)
54 {
55     struct node *newnode;
56     newnode =(struct node *) malloc (sizeof(struct node));
57     newnode->data=ele;
58     newnode->next=NULL;
59     if(rear==NULL)
60     {
61         front=newnode;
62         rear=newnode;
63     }
64     else
65     {
66         rear->next=newnode;
67         rear=newnode;
68     }
69     printf("\n Element %d was inserted!\n ",ele);
70 }
71
72 int delNode()
73 {
74     int ele;
75     struct node *temp;
76     if(front==NULL)
77     {
78         printf("\n Queue Underflow, The queue is empty!\n ");
79         queue();
80 }
```

```
81
82     ele=front->data;
83     temp=front;
84     if(front==rear)
85     {
86         front=NULL;
87         rear=NULL;
88     }
89     else
90     {
91         front=front->next;
92         free(temp);
93         return ele;
94     }
95 }
96 void display()
97 {
98     struct node *i;
99     if(front==NULL && rear==NULL)
100     {
101         printf("\n The Queue is empty!\n ");
102         queue();
103     }
104     printf("\n The Queue Contains:");
105     for(i=front;i!=NULL;i=i->next)
106         printf("%d ",i->data);
107 }
```

Output:

Stack:

```
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 1

Element 1 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 2

Element 2 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 3

Element 3 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 4

Element 4 was inserted!

Enter the choice:
1.Push.
2.Pop.
```

(a)

```
Element 4 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:1

Enter the element to Push: 5

Element 5 was inserted!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:3

The Stack Contains:
TOP-> 5 4 3 2 1
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

5 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

4 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

3 was deleted from Queue
Enter the choice:
```

(b)

Queue:

```
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:1

Enter the element to Insert: 1

Element 1 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:1

Enter the element to Insert: 2

Element 2 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:1

Enter the element to Insert: 3

Element 3 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:1

Enter the element to Insert: 4

Element 4 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
```

(a)

```
Element 4 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:1

Enter the element to Insert: 5

Element 5 was inserted!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:3

The Queue Contains:1 2 3 4 5
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

1 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

2 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:3

The Queue Contains:3 4 5
Enter the choice:
1.Insert.
2.Delete.
```

(b)

```
3 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

2 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

1 was deleted from Queue
Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:2

Stack UnderFlow, The Stack is empty!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:3

Stack is empty!

Enter the choice:
1.Push.
2.Pop.
3.Display.
4.Exit
Choice:4

Process returned 0 (0x0) execution time : 24.445 s
Press any key to continue.
```

(c)

```
2.Delete.
3.Display.
4.Exit
Choice:2

3 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

4 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

5 was deleted from Queue
Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:2

Queue UnderFlow, The queue is empty!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:3

Queue is empty!

Enter the choice:
1.Insert.
2.Delete.
3.Display.
4.Exit
Choice:4

Process returned 0 (0x0) execution time : 29.284 s
Press any key to continue.
```

(c)

Program-9:

(Queue Implementation)

Code:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  typedef struct node
5  {
6      int data;
7      struct node* prev;
8      struct node *next;
9  }Node;
10
11  Node *head=NULL;
12
13  void doublyLinkedList();
14  void insertNode(int);
15  void insertNodeToLeft();
16  void insertNodeToRight();
17  void deleteSpecifiedValue();
18  void displayList();
19
20  int main()
21  {
22      doublyLinkedList();
23      return 0;
24  }
25
26  void doublyLinkedList()
27  {
28      int choice=0;
29      printf("\n <--Doubly Linked List-->");
30      printf("\n 1.Enter Node 2.Enter Node to Left 3.Enter Node to Right 4.Delete A Node 5.DisplayList 6.Exit\n Choice: ");
31      scanf("%d",&choice);
32      switch(choice)
33      {
34          case 1: insertNode(0);
35                  break;
36          case 2: insertNode(1);
37                  break;
38          case 3: insertNode(2);
39                  break;
40          case 4: deleteSpecifiedValue();
41                  break;
```

```
41      break;
42      case 5: displayList();
43              break;
44      case 6: exit(0);
45
46      default: printf("\n Error choice, Try Again! ");
47              doublyLinkedList();
48      }
49      doublyLinkedList();
50  }
51
52  void insertNode(int flag)
53  {
54      Node *newnode;
55      newnode=(Node*)malloc(sizeof(Node));
56      printf("\n Enter the Element: ");
57      scanf("%d",&newnode->data);
58      if(head==NULL)
59      {
60          head=newnode;
61          newnode->next=NULL;
62          newnode->prev=NULL;
63          printf("\n First Node created \n");
64          doublyLinkedList();
65      }
66      if(flag==0)
67      {
68          Node *temp=head;
69          for(temp;(temp->next)!=NULL;temp=temp->next);
70          temp->next=newnode;
71          newnode->prev=temp;
72          newnode->next=NULL;
73      }
74      else if(flag==1)
75          insertNodeToLeft(newnode);
76      else
77          insertNodeToRight(newnode);
78  }
79
80  void insertNodeToRight(Node *tempNew)
```

```
80  void insertNodeToRight(Node *tempNew)
81  {
82      int ele;
83      char choice;
84      printf("\n Enter the Node element To who's right you want to Insert Node: ");
85      scanf("%d",&ele);
86      Node *temp=head;
87      for(temp;temp!=NULL;temp=temp->next)
88      {
89          if(temp->data==ele)
90          {
91              if(temp->next!=NULL)
92              {
93                  tempNew->next=temp->next;
94                  tempNew->prev=temp;
95                  (temp->next)->prev=tempNew;
96                  temp->next=tempNew;
97                  printf("\n Node created \n");
98                  doublyLinkedList();
99              }
100              else
101              {
102                  tempNew->next=NULL;
103                  tempNew->prev=temp;
104                  temp->next=tempNew;
105                  printf("\n Node created \n");
106                  doublyLinkedList();
107              }
108          }
109      }
110
111      printf("\n The given Element was not found! ,press Y to Try again! or press anything to exit: ");
112      fflush(stdin);
113      scanf("%c",&choice);
114      if(choice=='Y' || choice == 'y')
115          insertNodeToRight(tempNew);
116      else
117      {
118          free(tempNew);
119          printf("\n Node creation Failed \n");
```

```

119     printf("\n Node creation Failed \n");
120     doublyLinkedList();
121 }
122 }
123
124 void InsertNodeToLeft(Node *tempNew)
125 {
126     int ele;
127     char choice;
128     printf("\n Enter the Node element To who's left you want to Insert Node: ");
129     scanf("%d",&ele);
130     Node *temp=head;
131     if(head->data==ele)
132     {
133         tempNew->next=head;
134         tempNew->prev=NULL;
135         head=tempNew;
136         printf("\n Node created \n");
137         doublyLinkedList();
138     }
139     for(temp;temp!=NULL;temp=temp->next)
140     {
141         if(temp->data==ele)
142         {
143             tempNew->next=temp;
144             tempNew->prev=temp->prev;
145             (temp->prev)->next=tempNew;
146             temp->prev=tempNew;
147             printf("\n Node created \n");
148             doublyLinkedList();
149         }
150     }
151     printf("\n The given Element was not found! ,press Y to Try again! or press anything to exit: ");
152     fflush(stdin);
153     scanf("%c",&choice);
154     if(choice=='Y' || choice=='y')
155         InsertNodeToLeft(tempNew);
156     else
157     {
158         //printf("\n Exit\n");
159     }

```

```

159     free(tempNew);
160     printf("\n Node creation Failed \n");
161     doublyLinkedList();
162 }
163 }
164
165 void deleteSpecifiedValue()
166 {
167     if(head==NULL)
168     {
169         printf("\n Empty List!\n");
170         doublyLinkedList();
171     }
172     int ele;
173     printf("\n Enter the Node element to delete: ");
174     scanf("%d",&ele);
175     Node *temp=head;
176     if(head->next==NULL)
177     {
178         if(head->data==ele)
179         {
180             free(temp);
181             head=NULL;
182             printf("\n Node Deleted. \n Now List Is empty! ");
183             doublyLinkedList();
184         }
185     }
186     else
187     {
188         for(temp;temp!=NULL;temp=temp->next)
189         {
190             if(temp->data==ele)
191             {
192                 if(temp->next==NULL)
193                 {
194                     (temp->prev)->next=NULL;
195                     free(temp);
196                     printf("\n Node Deleted \n");
197                     doublyLinkedList();
198                 }

```

```

197         doublyLinkedList();
198     }
199     if(temp->prev==NULL)
200     {
201         (temp->next)->prev=NULL;
202         head=head->next;
203         printf("\n Node Deleted \n");
204         doublyLinkedList();
205     }
206     else
207     {
208         (temp->prev)->next=temp->next;
209         (temp->next)->prev=temp->prev;
210         free(temp);
211         printf("\n Node Deleted \n");
212         doublyLinkedList();
213     }
214 }
215 }
216 printf("\n The given element %d is not present in list!\n ",ele);
217 }
218 }
219
220 void displayList()
221 {
222     if(head==NULL)
223     {
224         printf("\n Empty List!\n");
225         doublyLinkedList();
226     }
227     Node *temp=head;
228     printf("\n The List Contains :");
229     for(temp;temp!=NULL;temp=temp->next)
230     {
231         printf(" %d ",temp->data);
232     }
233     doublyLinkedList();
234 }

```

Output:

```

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 1

Enter the Element: 1

First Node created

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 1

Enter the Element: 2

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 1

Enter the Element: 3

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 5

The List Contains : 1 2 3
<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 2

```

(a)

```

Enter the Element: 10

Enter the Node element To who's left you want to Insert Node: 1

Node created

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 5

The List Contains : 10 1 2 3
<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 3

Enter the Element: 4

Enter the Node element To who's right you want to Insert Node: 2

Node created

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 5

The List Contains : 10 1 2 4 3
<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 4

Enter the Node element to delete: 10

Node Deleted

```

(b)

```

Enter the Node element to delete: 10

Node Deleted

<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 5

The List Contains : 1 2 4 3
<--Doubly Linked List-->
1.Enter Node
2.Enter Node to Left
3.Enter Node to Right
4.Delete A Node
5.DisplayList
6.Exit
Choice: 6

Process returned 0 (0x0)   execution time : 38.214 s
Press any key to continue.

```

(c)

Program-10:

(Binary Search Tree_Implementation)

Code:

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  typedef struct node
5  {
6      int data;
7      struct node *left;
8      struct node *right;
9  }Node;
10 void tree();
11 Node * create();
12 Node *insert(Node *,Node *);
13 void traverse();
14 void preOrder(Node *);
15 void inOrder(Node *);
16 void postOrder(Node *);
17 void display(Node *,int);
18
19 Node *root;
20
21 int main()
22 {
23     tree();
24     return 0;
25 }
26 void tree()
27 {
28     int choice;
29     printf("\n <- Binary Search Tree-->\n 1.Insert Element\n 2.Traverse-All methods\n 3.Display BST\n 4.Exit\n Choice: ");
30     scanf("%d",&choice);
31     switch(choice)
32     {
33         case 1: insert(root,create()); break;
34         case 2: traverse(); break;
35         case 3: if(root==NULL)
36                 printf("\n Tree is Empty!");
37                 else
38                     display(root,0);
39                 break;
40         case 4: exit(0);break;
41         default: printf("\n Error Choice !\n ");
```

```
42     }
43     tree();
44 }
45 Node * create()
46 {
47     Node* newnode=(Node *)malloc(sizeof(Node));
48     printf("\n Enter the Element: ");
49     scanf("%d",&newnode->data);
50     newnode->left=NULL;
51     newnode->right=NULL;
52     return newnode;
53 }
54 Node * insert(Node *Root,Node *newNode)
55 {
56     if(root==NULL)
57     {
58         root=newNode;
59         printf("\n Root Node Created ");
60     }
61     else
62     {
63         if(newNode->data>Root->data)
64         {
65             if(Root->right==NULL)
66             {
67                 Root->right=newNode;
68             }
69             else
70                 insert(Root->right,newNode);
71         }
72         else
73         {
74             if(newNode->data<Root->data)
75             {
76                 if(Root->left==NULL)
77                 {
78                     Root->left=newNode;
79                 }
80                 else
```

```
81         else
82             insert(Root->left,newNode);
83     }
84 }
85 void traverse()
86 {
87     if(root==NULL)
88     {
89         printf("\n The Tree Is Empty! ");
90         return;
91     }
92     printf("\n Pre-Order Traverse: ");
93     preOrder(root);
94     printf("\n In-Order Traverse: ");
95     inOrder(root);
96     printf("\n Post-Order Traverse: ");
97     postOrder(root);
98 }
99 void preOrder(Node *Root)
100 {
101     if(Root==NULL){
102         printf(" %d ",Root->data);
103         preOrder(Root->left);
104         preOrder(Root->right);
105     }
106 }
107 void inOrder(Node *Root)
108 {
109     if(Root==NULL){
110         inOrder(Root->left);
111         printf(" %d ",Root->data);
112         inOrder(Root->right);
113     }
114 }
115 void postOrder(Node *Root)
116 {
117     if(Root==NULL){
118         postOrder(Root->left);
119         postOrder(Root->right);
120         printf(" %d ",Root->data);
121     }
122 }
```

```

120     postOrder(Root->right);
121     printf(" %d ",Root->data);
122 }
123 }
124 void display(Node* root,int i)
125 {
126     int j;
127     if(root!=NULL)
128     {
129         display(root->right,i+1);
130         for(j=0;j<i;j++)
131             printf("----");
132         printf("%d\n",root->data);
133         display(root->left,i+1);
134     }
135 }

```

Output:

```

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

```

Enter the Element: 100

```

Root Node Created
<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

```

Enter the Element: 150

```

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

```

Enter the Element: 50

```

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

```

Enter the Element: 30

```

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST

```

(a)

```

1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

```

Enter the Element: 60

```

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

```

Enter the Element: 170

```

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 3
-----170
----150
100
-----60
----50
-----30

```

```

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 2

```

```

Pre-Order Traverse: 100 50 30 60 150 170
In-Order Traverse: 30 50 60 100 150 170
Post-Order Traverse: 30 60 50 170 150 100

```

(b)