Bapatla Engineering College

(Autonomous)

Department of Information Technology

18ITL32 - Data Structures Lab Manual

(w.e.f. 2018-2019).

Bapatla Engineering College :: Bapatla

(Autonomous under Acharya Nagarjuna University)
(Sponsored by Bapatla Education Society)
BAPATLA - 522102 Guntur District, A.P., INDIA
www.becbapatla.ac.in.

Contents

1. Lab Course Description

1. Course code :18ITL32
2. Course Title : **Data Structures Lab**
3. Core/ Elective : Core
4. Pre-requisites (if any) : Basic C-Programming Knowledge.
5. Semester / year in which offered : **II year I Semester**
6. No. of weeks of Instruction : 15 weeks
7. No. of hours per week : 3
8. **Course objectives:**

   i. Understand and remember algorithms and its analysis procedure.
   ii. Learn about various linear data structures and their basic operations.
   iii. Understand and remember algorithms for sorting techniques.
   iv. Understand the concept of Binary tree, binary search tree, AVL tree and their basic operatios.

9. **Course outcomes:**

   The students will be able to
   i. Implement stack, queue using array and linked lists.
   ii. Implement list data structures and their basic operations.
   iii. Implement sorting techniques.
   iv. Implement various tree data structures and their traversals.

10. **List of programs:** Separate sheet has been attached.
11. **Evaluation procedure**

INDEX	*EVALUATION*	*TOTAL MARKS*
A	*Marks allotted for day-to-day lab work .*	*20 M*
B	*Marks allotted for record*	*15 M*
C	*Marks Awarded for Lab Exam*	*15 M*
D	*Continuous Internal Evaluation (CIE) (A+B+C)*	*50 M*
E	*Semester End Examination (SEE)*	*50 M*

2. List of Experiments

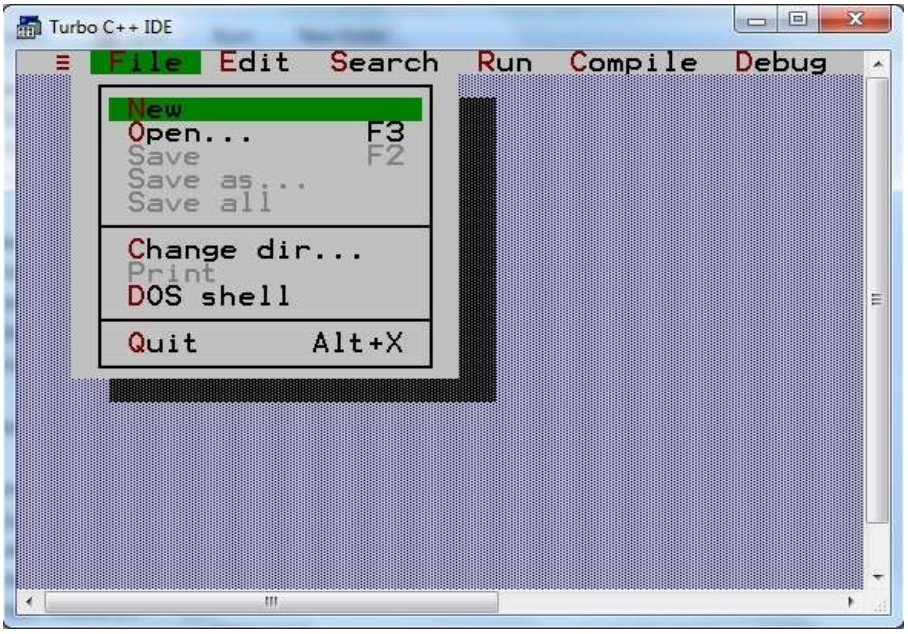
3. Compile and Run C Program

To compile and run a C language program, you need a C compiler. To setup a C language compiler in your Computer/laptop, there are two ways:

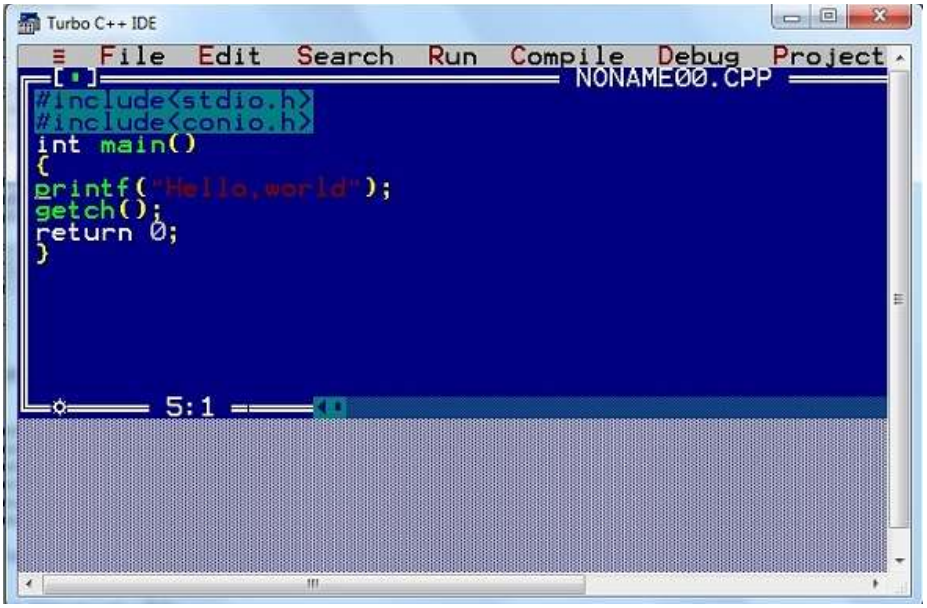
1. Download a full-fledged IDE like Turbo C or Microsoft Visual C++, which comes along with a C language compiler.
2. Or, you use any text editor to edit the program files and download the C compiler separately.

Using an IDE - Turbo C

Turbo C IDE is the oldest IDE for C programming. It is freely available over internet and is good for a beginner.
Step 1 : Open turbo C IDE(Integrated Development Environment), click on **File** and then click on New

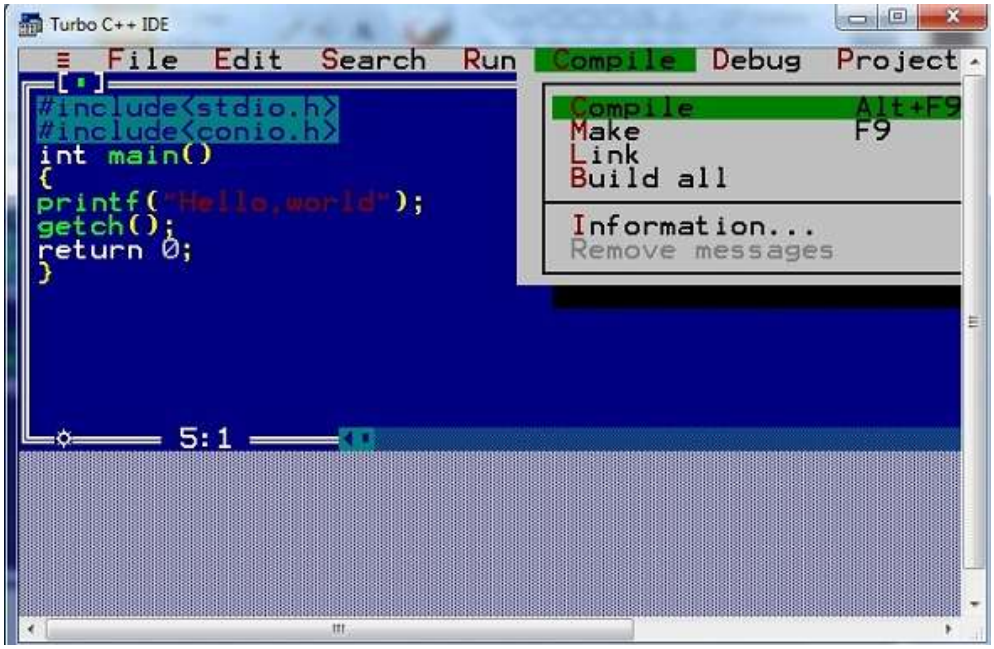


Step 2 : Write the simple C-program in the new window.

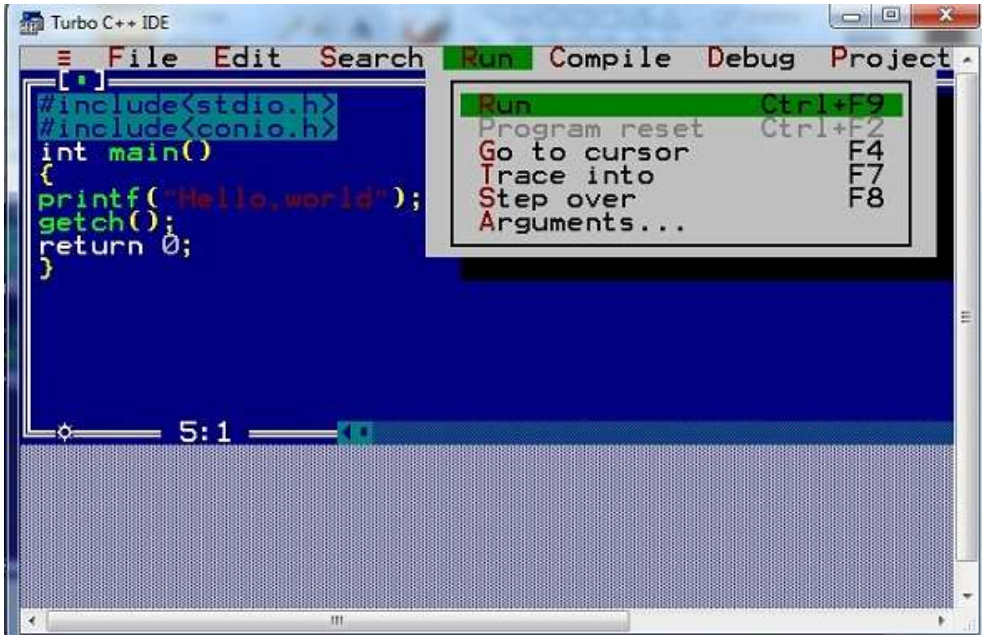


Step3: Save the program with .c extension (for e.g: simple.c).To save the program press F2 key or Ctrl+S.

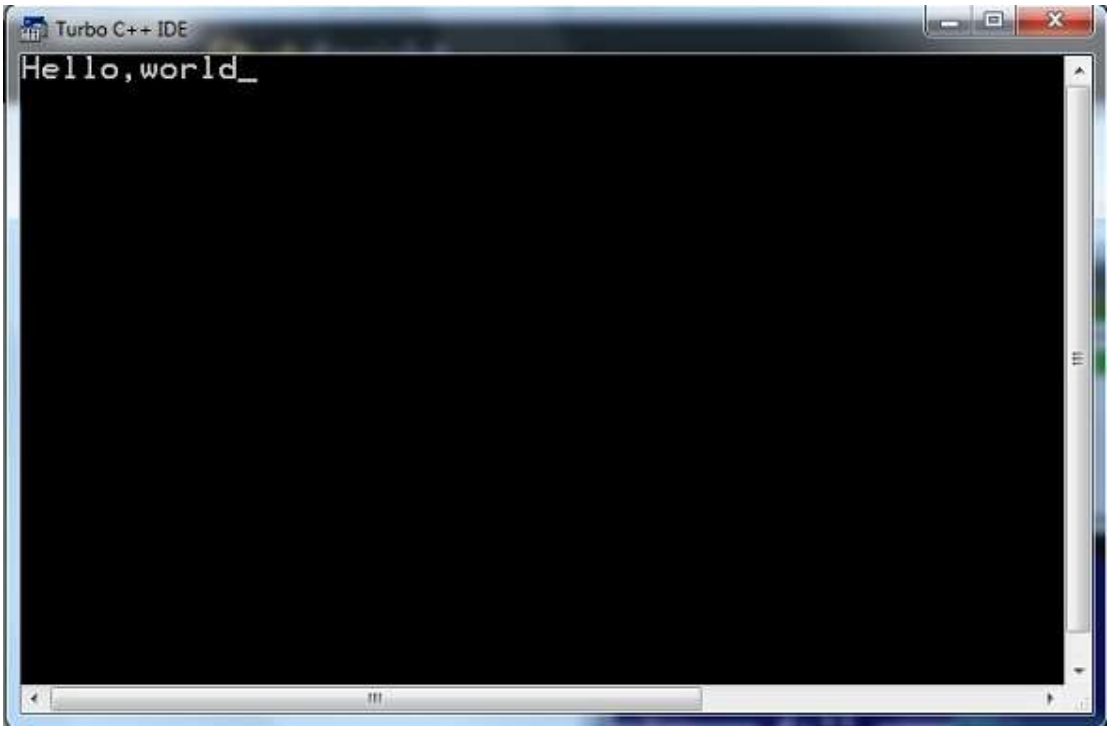
Step 4 : Click on compile or press Alt+f9 to compile the code



Step 5 : Click on Run or press Ctrl+f9 to run the code



Step 6 : Output



4. Lab Programs

Program 1

Aim: To write a C-program to implement Single linked list and perform the following operations

(i)	**Create()**	**(ii) Traverse()**	**(iii) insert_at_begin()**
(iv)	**Insert_at_end()**	**(v) insert_at_specific()**	**(vi) delete_at_begin()**
(vii)	**Delete_at_end()**	**(viii) delete_at_specific()**	**(ix) delete_at_specific()**
(x)	**Count_nodes()**	**(xi) Exit()**	

Source Code:- { sll.c }

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
void create();
void traverse();
void insertatbegin();
void insertatend();
void insertatspecific();
void deleteatbegin();
void deleteatend();
void deleteatspecific();
void countnode();
void reverse();
struct node *ptr,*cptr;
char ch;
struct node
{
        int data;
        struct node *next;
};
struct node *head=NULL;
void main()
{
  int ch;
  clrscr();
```

```
printf("\n1.create\n2.traverse\n3.insertatbegin\n4.insertatend\n5.insertatspecific\n6.deleteatbegin\n7
.deleteatend\n8.deleteatspecific\n9.countnode\n10.reverse\n11.exit");
do
 {
  printf("\nenter your choice");
  scanf("%d",&ch);
  switch(ch)
    {
     case 1:create();
            break;
     case 2:traverse();
            break;
     case 3:insertatbegin();
            break;
     case 4:insertatend();
            break;
     case 5:insertatspecific();
            break;
     case 6:deleteatbegin();
            break;
     case 7:deleteatend();
            break;
     case 8:deleteatspecific();
            break;
     case 9:countnode();
            break;
     case 10:reverse();
            break;
     case 11:exit(0);
            break;
     default:printf("\n invalid choice");
  }
}while(ch<=11);
```

```
  getch();
}
void create()
{
    printf("\nenter first node data");
    ptr=(struct node*)malloc(sizeof(struct node));
    scanf("%d",&ptr->data);
    head=ptr;
    do
       {
         printf("\nenter next node data");
        cptr=(struct node*)malloc(sizeof(struct node));
        scanf("%d",&cptr->data);
        ptr->next=cptr;
        ptr=cptr;
        printf("\ndo you want continue y/n");
        ch=getch();
      }while(ch=='Y'||ch=='y');
     ptr->next=NULL;
}
void traverse()
{
  struct node *ptr1;
  ptr1=head;
  printf("\n the list is:");
  while(ptr1!=NULL)
   {
        printf("\t %d->",ptr1->data);
        ptr1=ptr1->next;
    }
}
void insertatbegin()
{
```

```
    struct node *cptr;
    cptr=(struct node*)malloc(sizeof(struct node));
    printf("\nenter new node data");
    scanf("%d",&cptr->data);
    if(head==NULL)
     {
          cptr->next=NULL;
          head=cptr;
     }
   else
     {
          cptr->next=head;
          head=cptr;
     }
}
void insertatend()
  {
    struct node *cptr;
    cptr=(struct node*)malloc(sizeof(struct node));
    printf("\n enter new node data");
    scanf("%d",&cptr->data);
    cptr->next=NULL;
    if(head==NULL)
      {
       head=cptr;
      }
    else
    {
      struct node *temp;
      temp=head;
       while(temp->next!=NULL)
          {
          temp=temp->next;
```

```
        }
       temp->next=cptr;
      }
    }
  void insertatspecific()
    {
      int n1;
      struct node *cptr;
      cptr=(struct node*)malloc(sizeof(struct node));
      printf("\n where you want insert ");
      scanf("%d",&n1);
      printf("\n enter new node data");
      scanf("%d",&cptr->data);
      if(head==NULL)
        {
          cptr->next=NULL;
          head=cptr;
        }
      else
        {
          struct node *temp;
          temp=head;
          while(temp->data!=n1)
           {
             temp=temp->next;
           }
          cptr->next=temp->next;
          temp->next=cptr;
        }
      }
  void deleteatbegin()
    {
      struct node *temp1;
```

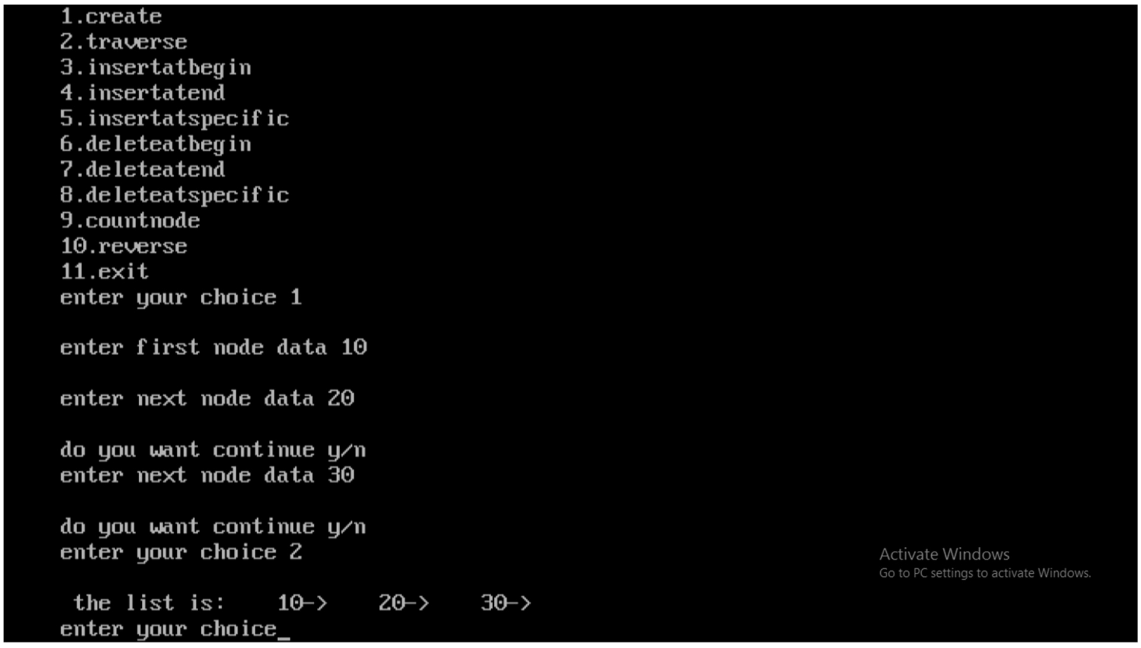
```
    temp1=head;
    if(head==NULL)
      {
        printf("\n the list is empty and deletion is not possible");
      }
    else
      {
        if(temp1->next==NULL)
         {
          head=NULL;
           free(temp1);
         }
        else
          {
           head=temp1->next;
           free(temp1);
          }
        }
    }
void deleteatend()
{
    struct node *temp1,*temp2;
    temp1=head;
    if(head==NULL)
      {
        printf("\n the list is empty and deletion is not possible");
      }
    else
      {
       if(temp1->next==NULL)
         {
            head=NULL;
            free(temp1);
```

```
        }
      else
        {
          while(temp1->next!=NULL)
            {
              temp2=temp1;
              temp1=temp1->next;
            }
          temp2->next=NULL;
          free(temp1);
        }
    }
  }
void deleteatspecific()
 {
   int n1;
   struct node *temp1,*temp2;
   temp1=head;
   if(head==NULL)
     {
       printf("|n list is empty and deletion is not possible");
     }
   else
      {
         Printf(“which node u want to delete\n”);
         Scanf(“%d”,&n1);
         if(temp1->next==NULL)
           {
             head=NULL;
             free(temp1);
           }
         else
           {
```

```
            while(temp1->data!=n1)
              {
                temp2=temp1;
                temp1=temp1->next;
              }
             temp2->next=temp1->next;
            free(temp1);
           }
        }
  }
void countnode()
   {
     int count=0;
     struct node *temp1;
     temp1=head;
       while(temp1->next!=NULL)
         {
            count++;
            temp1=temp1->next;
         }
      printf("\n no.of nodes in the list is %d",count+1);
   }
void reverse()
  {
    struct node *temp1,*temp2;
    temp1=head;
      if(temp1->next==NULL)
        {
          printf("%d->",temp1->data);
        }
      else
        {
           while(temp1->next!=NULL)
```

```
        {
          temp2=temp1;
          temp1=temp1->next;
         }
      printf("%d->",temp1->data);
      temp2->next=NULL;
       reverse();
    }
}
```

Output:-



```
 enter new node data 25

enter your choice 2

 the list is:    5->     10->    20->    25->    30->    40->
enter your choice 6

enter your choice 2

 the list is:    10->    20->    25->    30->    40->
enter your choice 7

enter your choice 2

 the list is:    10->    20->    25->    30->
enter your choice 8

 which node do you want to delete 25

enter your choice 9

 no.of nodes in the list is 3
enter your choice 10
30->20->10->
enter your choice 11
 the list is:    10->    20->    30->
enter your choice 3

enter new node data 5

enter your choice 2

 the list is:    5->     10->    20->    30->
enter your choice 4

 enter new node data 40

enter your choice 2

 the list is:    5->     10->    20->    30->    40->
enter your choice 5

 where you want insert 20

 enter new node data 25

enter your choice 2

 the list is:    5->     10->    20->    25->    30->    40->
enter your choice
```

Program 2

Aim: To write a C-program to implement Double linked list and perform the following operations.

(ii)	**Create()**	**(ii) Traverse()**	**(iii) insert_at_begin()**
(v)	**Insert_at_end()**	**(v) insert_at_specific()**	**(vi) delete_at_begin()**
(vii)	**Delete_at_end()**	**(viii) delete_at_specific()**	**(xi) Count_nodes()**
(x)	**Reverse()**	**(xi) Exit()**	

Sorce Code:- { dll.c }

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
struct node
 {
      int data;
      struct node *prev,*next;
 };
struct node *head=NULL,*ptr,*cptr;
void create();
void traverse();
void insertbegin();
void insertend();
void insertspecific();
void deletebegin();
void deleteend();
void deletespecific();
void reverse();
void countnodes();
void main()
{
 int ch;
 clrscr();
 printf("\n1.create\n2.traverse\n3.insert_begin\n4.insert_end\n5.insert_specific");
 printf("\n6.delete_begin\n7.delete_end\n8.delete_specific\n9.count_nodes\n10.reverse\n 11.exit");
```

```
do
 {
   printf("\nenterur choice");
   scanf("%d",&ch);
   switch(ch)
      {
        case 1:create();
               break;
        case 2:traverse();
               break;
        case 3:insertbegin();
               break;
        case 4:insertend();
               break;
        case 5:insertspecific();
               break;
        case 6:deletebegin();
               break;
        case 7:deleteend();
               break;
        case 8:deletespecific();
               break;
        case 9:countnodes();
               break;
        case 10:reverse();
               break;
        case 11:exit(0);
         }
      }while(ch<=11);
   getch();
 }
void create()
{
```

```
  int ch;
  //struct node *ptr;
  ptr=(struct node *)malloc(sizeof(struct node));
  printf("enter ur first node data\n");
  scanf("%d",&ptr->data);
  head=ptr;
  ptr->prev=head;
  do
   {
    cptr=(struct node *)malloc(sizeof(struct node));
    printf("\nenter next node data");
    scanf("%d",&cptr->data);
    cptr->prev=ptr;
    ptr->next=cptr;
    ptr=cptr;
    printf("\ndo u want continueY/N");
    ch=getch();
   }while(ch=='y'||ch=='Y');
  ptr->next=NULL;
}
void traverse()
{
  struct node *tmp;
  tmp=head;
  while(tmp!=NULL)
    {
      printf("\t%d<->",tmp->data);
      tmp=tmp->next;
    }
}
void insertbegin()
{
  struct node *cptr;
```

```
  cptr=(struct node *)malloc(sizeof(struct node));
  printf("\nenter new node data");
  scanf("%d",&cptr->data);
  cptr->prev=NULL;
  if(head==NULL)
  {
      cptr->next=NULL;
      head=cptr;
  }
  else
  {
      cptr->next=head;
      head->prev=cptr;
      head=cptr;
  }
}
void insertend()
{
  cptr=(struct node *)malloc(sizeof(struct node));
  printf("\nenter new node data");
  scanf("%d",&cptr->data);
  if(head==NULL)
  {
      head=cptr;
      cptr->prevh=NULL;
   }
  else
  {
      struct node *tmp;
      tmp=head;
      while(tmp->next!=NULL)
      {
         tmp=tmp->next;
```

```
        }
       tmp->next=cptr;
      cptr->prev=tmp;
      cptr->next=NULL;
     }
   }
void insertspecific()
{
    int n1;
    struct node *tmp1;
    tmp1=head;
    cptr=(struct node *)malloc(sizeof(struct node));
    printf("\nwhere u want insert");
    scanf("%d",&n1);
    if(tmp1==NULL)
     {
       head=cptr;
       cptr->next=NULL;
     }
    else
     {
       printf("enter insert new node element");
       scanf("%d",&cptr->data);
       if(head->next==NULL)
         {
           cptr->next=head;
           head=cptr;
         }
       else
        {
          while(tmp1->data!=n1)
            {
               tmp1=tmp1->next;
```

```
            }
            cptr->next=tmp1->next;
            cptr->prev=tmp1;
            tmp1->next=cptr;
           }
       }
   }
void deletebegin()
 {
   if(head==NULL)
    {
      printf("\n list is  empty,deletion is not possible");
    }
   else
    {
      if(head->next==NULL)
        {
          head=NULL;
        }
      else
        {
          head=head->next;
          head->prev=NULL;
        }
    }
}

void deleteend()
{
  struct node *tmp1 ,*tmp2;
  tmp1=head;
  if(head==NULL)
   {
```

```
        printf("\n list is empty, deletion is not possible");
      }
     if(tmp1->next==NULL)
      {
        head=NULL;
        free(tmp1);
      }
     else
      {
       while(tmp1->next!=NULL)
        {
          tmp2=tmp1;
          tmp1=tmp1->next;
        }
       tmp2->next=NULL;
       free(tmp1);
      }
}
void deletespecific()
{
    int n1;
    struct node *tmp1,*tmp2;
    tmp1=head;
    if(head==NULL)
      {
        printf("\n list is empty,deletion is not possible");
      }
    else
      {
       if(tmp1->next==NULL)
         {
           head=NULL;
           free(tmp1);
```

```
            }
          else
            {
              printf("\n which node we want to delete");
              scanf("%d",&n1);
              while(tmp1->data!=n1)
                {
                  tmp2=tmp1;
                  tmp1=tmp1->next;
                }
              tmp2->next=tmp1->next;
              tmp1->next->prev=tmp1->prev;
              free(tmp1);
            }
         }
     }
void countnodes()
{
  intcnt=0;
  struct node *tmp1;
  tmp1=head;
  while(tmp1->next!=NULL)
       {
         cnt++;
         tmp1=tmp1->next;
       }
    printf("\n total number of nodes is%d",cnt+1);
  }

void reverse()
    {
        struct node *tmp1;
        tmp1=head;
```

```
        while(tmp1->next!=NULL)
          {
            tmp1=tmp1->next;
          }
        while(tmp1->prev!=NULL)
        {
         printf("%d<->",tmp1->data);
           tmp1=tmp1->prev;
        }
    printf("%d",tmp1->data);
}
```

Output:-

```
1.create
2.traverse
3.insert begin
4.insert end
5.insert specific
6.delete begin
7.delete end
8.delete specific
9.count nodes
10.reverse
11.exit
enter ur choice
1
enter ur first node data
10

enter next node data
20

do u want continueY/N
enter next node data
30

do u want continueY/N
```

```
do u want continueY/N
enter ur choice2
        10<->   20<->   30<->
enter ur choice 3

enter new node data 5

enter ur choice 2
        5<->    10<->   20<->   30<->
enter ur choice 4

enter new node data 40

enter ur choice 5

where u want insert 20
enter insert new node element 25

enter ur choice 2
        5<->    10<->   20<->   25<->   30<->   40<->
enter ur choice 6

enter ur choice 2
        10<->   20<->   25<->   30<->   40<->
enter ur choice_
```



```
where u want insert 20
enter insert new node element 25

enter ur choice 2
        5<->    10<->   20<->   25<->   30<->   40<->
enter ur choice 6

enter ur choice 2
        10<->   20<->   25<->   30<->   40<->
enter ur choice 7

enter ur choice 2
        10<->   20<->   25<->   30<->
enter ur choice 8

 which node we want to delete 25

enter ur choice 9

 total number of nodes is3
enter ur choice 10

30<->
20<->10
enter ur choice 11
```

Program 3

Aim: To write a C-program to implement Circular Single linked list and perform the following operations.

(i)	**Create()**	**(ii) Traverse()**	**(iii) insert_at_begin()**
(iv)	**Insert_at_end()**	**(v) insert_at_specific()**	**(vi) delete_at_begin()**
(vii)	**Delete_at_end()**	**(viii) delete_at_specific()**	**(ix) Exit()**

Sorce Code:- { cll.c }

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<alloc.h>
struct node
 {
    int data;
    struct node *next;
 };
struct node *head=NULL,*ptr,*cptr;
void create();
void traverse();
void insertbegin();
void insertend();
void insertspecific();
void deletebegin();
void deleteend();
void deletespecific();
void main()
 {
    intch;
   clrscr();
   printf("1.create\n2.traverse\n3.insert begin\n4.insert end\n5.insert specific");
   printf("\n6.delete begin\n7.delete end\n8.delete specific\n9.exit");
   do
     {
```

```
            printf("\nenterur choice\n");
            scanf("%d",&ch);
            switch(ch)
              {
               case 1:create();
                   break;
               case 2:traverse();
                   break;
               case 3:insertbegin();
                   break;
               case 4:insertend();
                   break;
               case 5:insertspecific();
                   break;
               case 6:deletebegin();
                   break;
               case 7:deleteend();
                   break;
               case 8:deletespecific();
                   break;
               case 9:exit(0);
               default:printf("invalid choice");
               }
             }while(ch<=9);
          getch();
        }
    void create()
     {
       char ch;
       printf("enter first node data\n");
       ptr=(struct node *)malloc(sizeof(struct node));
       scanf("%d",&ptr->data);
       head=ptr;
```

```
        do
          {
            printf("enter next node data\n");
            cptr=(struct node *)malloc(sizeof(struct node));
            scanf("%d",&cptr->data);
            ptr->next=cptr;
            ptr=cptr;
            printf("do u want continue Y/N\n");
            ch=getch();
          }while(ch=='y'||ch=='Y');
        ptr->next=head;
      }
void traverse()
{
    struct node *tmp;
    tmp=head;
    while(tmp->next!=head)
      {
        printf("\t%d->",tmp->data);
        tmp=tmp->next;
      }
printf("\t%d->",tmp->data);
  }
void insertbegin()
  {
    struct node *tmp;
    tmp=head;
    cptr=(struct node *)malloc(sizeof(struct node));
    printf("enter new node data\n");
    scanf("%d",&cptr->data);
      if(head==NULL)
      {
          head=cptr;
```

```
            cptr->next=head;
          }
          else
          {
            while(tmp->next!=head)
            {
                tmp=tmp->next;
            }
            cptr->next=head;
            head=cptr;
            tmp->next=cptr;
          }
      }
    void insertend()
    {
        struct node *tmp;
        tmp=head;
        cptr=(struct node *)malloc(sizeof(struct node));
        printf("enter new node data\n");
        scanf("%d",&cptr->data);
        if(head==NULL)
          {
            head=cptr;
            cptr->next=head;
          }
        else
         {
            while(tmp->next!=head)
            {
               tmp=tmp->next;
            }
           tmp->next=cptr;
           cptr->next=head;
```

```
    }
  }
void insertspecific()
{
    int n1;
    struct node *cptr;
    cptr=(struct node*)malloc(sizeof(struct node));
    printf("\n where you want insert");
    scanf("%d",&n1);
    printf("\n enter new node data");
    scanf("%d",&cptr->data);
    if(head==NULL)
      {
        head=cptr;
        cptr->next=head;
      }
    else
    {
        struct node *temp;
        temp=head;
        while(temp->data!=n1)
        {
          temp=temp->next;
        }
      cptr->next=temp->next;
      temp->next=cptr;
    }
}
void deletebegin()
{
  struct node *tmp1,*tmp2;
  tmp1=head,tmp2=head;
  if(tmp1==NULL)
```

```
 {
   printf("deletion is not possible");
 }
 else
{
   if(tmp1->next==NULL)
    {
        head=NULL;
        free(tmp1);
     }
   else
   {
    while(tmp2->next!=head)
      {
         tmp2=tmp2->next;
      }
    head=tmp1->next;
    tmp2->next=head;
    free(tmp1);
   }
  }
}
void deleteend()
{
  struct node *tmp1,*tmp2;
  tmp1=head;
  if(head==NULL)
  {
     printf("deletion not possible");
  }
  else
     {
        if(tmp1->next==NULL)
```

```
            {
                head=NULL;
                free(tmp1);
            }
            else
            {
              while(temp1->next!=head)
                {
                    tmp2=tmp1;
                    tmp1=tmp1->next;
                }
              tmp2->next=head;
              free(tmp1);
            }
          }
    }
void deletespecific()
{
    int n1;
    struct node *tmp1,*tmp2;
    tmp1=head;
    printf("which node u want delete\n");
    scanf("%d",&n1);
     if(head==NULL)
        {
           printf("deletion is  not possible");
        }
     else
        {
           if(head->next==NULL)
            {
               head=NULL;
                free(tmp1);
```

```
        }
        else
        {
          while(tmp1->data!=n1)
          {
            tmp2=tmp1;
            tmp1=tmp1->next;
          }
        tmp2->next=tmp1->next;
        free(tmp1);
        }
      }
  }
```

Output:-

```
1.create
2.traverse
3.insert begin
4.insert end
5.insert specific
6.delete begin
7.delete end
8.delete specific
9.exit
enter ur choice
1
enter first node data
11
enter next node data
22
do u want continue Y/N
enter next node data
33
do u want continue Y/N

enter ur choice
2
        11->    22->    33->
enter ur choice
```

```
enter ur choice
3
enter new node data
1

enter ur choice
2
        1->     11->    22->    33->
enter ur choice
4
enter new node data
44

enter ur choice
5

 where you want insert 11

 enter new node data 77

enter ur choice
2
        1->     11->    77->    22->    33->    44->
enter ur choice
6

enter ur choice
2
        11->    77->    22->    33->    44->
enter ur choice
7

enter ur choice
2
        11->    77->    22->    33->
enter ur choice
8
which node u want delete
77

enter ur choice
2
        11->    22->    33->
enter ur choice
9
```

Program 4

Aim: Write a C-Program to perform Addition and Multiplication of two Polynomials using linked list.

Source Code:- { poly.c }

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
void create();
void traverse(struct node *);
void add(struct node*,struct node*);
void mul(struct node*,struct node*);
struct node
 {
    int coff;
    int expo;
    struct node *next;
 };
struct node *head=NULL,*poly1=NULL,*poly2=NULL,*sum,*multi;
void main()
   {
       clrscr();
       printf("\nenter first polynomial details");
         create();
         poly1=head;
       printf("\nenter second polynomial detalis");
         create();
         poly2=head;
       printf("\n \nfirst polynomial is:");
          traverse(poly1);
       printf("\n \nsecond polynomial is:");
          traverse(poly2);
       printf("\n \naddition of two polynomials is:");
```

```
            add(poly1,poly2);
             traverse(sum);
          printf("\n \nmultiplication of two polynomials is:");
             mul(poly1,poly2);
          getch();
       }
    void create()
     {
        char ch;
        struct node *ptr,*cptr;
        printf("\nenter first term coefficient and exponent:");
        ptr=(struct node*)malloc(sizeof(struct node));
        scanf("%d%d",&ptr->coff,&ptr->expo);
        head=ptr;
          do
           {
             cptr=(struct node*)malloc(sizeof(struct node));
             printf("\nenter next term coefficient and exponent:");
             scanf("%d%d",&cptr->coff,&cptr->expo);
             ptr->next=cptr;
             ptr=cptr;
             printf("\ndo you want to countinue y/n");
             ch=getche();
           }while(ch=='Y'||ch=='y');
        ptr->next=NULL;
      }
     void traverse( struct node *poly)
     {
       struct node *temp;
       temp=poly;
       while(temp!=NULL)
         {
            printf("%dx^%d+",temp->coff,temp->expo);
```

```
            temp=temp->next;
        }
        printf("0x^0");
    }
void add(struct node *poly1,struct node *poly2)
 {
    struct node *ptr,*cptr;
    ptr=(struct node *)malloc(sizeof(struct node));
    if(poly1->expo>poly2->expo)
      {
        ptr->coff=poly1->coff;
        ptr->expo=poly1->expo;
        poly1=poly1->next;
      }
     else if(poly2->expo>poly1->expo)
      {
        ptr->coff=poly2->coff;
        ptr->expo=poly2->expo;
        poly2=poly2->next;
      }
     else
      {
         ptr->coff=poly1->coff+poly2->coff;
         ptr->expo=poly1->expo;
         poly1=poly1->next;
         poly2=poly2->next;
      }
     sum=ptr;
    while(poly1!=NULL&&poly2!=NULL)
     {
       cptr=(struct node *)malloc(sizeof(struct node));
       if(poly1->expo>poly2->expo)
          {
```

```
            cptr->coff=poly1->coff;
            cptr->expo=poly1->expo;
            poly1=poly1->next;
          }
        else if(poly2->expo>poly1->expo)
          {
            cptr->coff=poly2->coff;
            cptr->expo=poly2->expo;
            poly2=poly2->next;
          }
        else
        {
           cptr->coff=poly1->coff+poly2->coff;
           cptr->expo=poly1->expo;
           poly1=poly1->next;
           poly2=poly2->next;
         }
        ptr->next=cptr;
        ptr=cptr;
      }
    while(poly1!=NULL)
      {
         struct node *cptr;
         cptr=(struct node *)malloc(sizeof(struct node));
         cptr->coff=poly1->coff;
         cptr->expo=poly1->expo;
         ptr->next=cptr;
         ptr=cptr;
         poly1=poly1->next;
      }
    while(poly2!=NULL)
      {
         struct node *cptr;
```

```
            cptr=(struct node *)malloc(sizeof(struct node));
            cptr->coff=poly2->coff;
            cptr->expo=poly2->expo;
            ptr->next=cptr;
            ptr=cptr;
            poly2=poly2->next;
          }
         ptr->next=NULL;
       }
void mul(struct node *poly1,struct node *poly2)
 {
    struct node *ptr,*cptr,*temp1,*temp2;
    temp1=poly1;
    temp2=poly2;
    sum=(struct node*)malloc(sizeof(struct node));
    multi=(struct node*)malloc(sizeof(struct node));
     sum->coff=0;
     sum->expo=0;
     sum->next=NULL;
    while(temp2!=NULL)
      {
       temp1=poly1;
       multi=NULL;
       while(temp1!=NULL)
         {
           cptr=(struct node*)malloc(sizeof(struct node));
           cptr->coff=temp1->coff*temp2->coff;
           cptr->expo=temp1->expo+temp2->expo;
           temp1=temp1->next;
            if(multi==NULL)
              {
                 ptr=(struct node*)malloc(sizeof(struct node));
                 ptr=cptr;
```

```
                multi=cptr;
              }
             else
              {
                ptr->next=cptr;
                ptr=cptr;
              }
            }
        ptr->next=NULL;
        add(multi,sum);
        temp2=temp2->next;
        }
       traverse(sum);
    }
```

Output:-

```
enter first polynomial details
enter first term coefficient and exponent:2 2

enter next term coefficient and exponent:1 1

do you want to countinue y/ny
enter next term coefficient and exponent:1 0

do you want to countinue y/nn
enter second polynomial detalis
enter first term coefficient and exponent:3 2

enter next term coefficient and exponent:4 1

do you want to countinue y/nn

first polynomial is:2x^2+1x^1+1x^0+0x^0

second polynomial is:3x^2+4x^1+0x^0

addition of two polynomials is:5x^2+5x^1+1x^0+0x^0

multiplication of two polynomials is:6x^4+11x^3+7x^2+4x^1+0x^0+0x^0_
```

Program 5

(a) Aim: To write a C-program to implement Stack using Linked List and perform the following operations.

(i) Push()
(ii) Pop()
(iii) Display()
(iv) Exit()

Sorce Code:- { stakll.c }

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
void push();
void pop();
void display();
struct node
  {
      int data;
      struct node *next;
  };
struct node *top=NULL;
struct node *ptr,*cptr;
void main()
  {
   int ch;
   clrscr();
   do
    {
      printf("\n1.push\n2.pop\n3.display\n4.exit\n");
      printf("enter your choice\n");
      scanf("%d",&ch);
      switch(ch)
        {
          case 1:push();
```

```
                    break;
                case 2:pop();
                    break;
                case 3:display();
                    break;
                case 4:exit(0);
                    break;
                default:printf("\n invalid choice");
              }
        }while(ch<=4);
        getch();
      }
    void push()
     {
        struct node *cptr;
        cptr=(struct node*)malloc(sizeof(struct node));
        printf("enter node data\n");
        scanf("%d",&cptr->data);
         if(top==NULL)
          {
            cptr->next=NULL;
            top=cptr;
          }
         else
          {
            cptr->next=top;
             top=cptr;
          }
       }
     void pop()
       {
        if(top==NULL)
         {
```

```
        printf("\n list is empty,deletion is not possible");
      }
     else
      {
        printf("deleted element is %d\n",top->data);
        top=top->next;
      }
   }
  void display()
   {
     struct node *temp;
     temp=top;
     printf("\nelements in stack are:");
       while(temp!=NULL)
        {
          printf("\t\n%d",temp->data);
          temp=temp->next;
        }
     }
```

Output:-

```
1.push
2.pop
3.display
4.exit
enter your choice
1
enter node data
23

1.push
2.pop
3.display
4.exit
enter your choice
1
enter node data
32

1.push
2.pop
3.display
4.exit
enter your choice
1_
```

```
enter node data
46

1.push
2.pop
3.display
4.exit
enter your choice
3

elements in stack are:
46
32
23
1.push
2.pop
3.display
4.exit
enter your choice
2
```



```
deleted element is 46

1.push
2.pop
3.display
4.exit
enter your choice
3

elements in stack are:
32
23
1.push
2.pop
3.display
4.exit
enter your choice
2
```



```
deleted element is 32

1.push
2.pop
3.display
4.exit
enter your choice
2
deleted element is 23

1.push
2.pop
3.display
4.exit
enter your choice
3

elements in stack are:
1.push
2.pop
3.display
4.exit
enter your choice
4
```

Program 5

(b) Aim: To write a C-program to implement Queue using Linked List and perform the following operations.

(i) Enqueue()
(ii) Dequeue()
(iii) Display()
(iv) Exit()

Source Code:- { quell.c }

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
void enq();
void deq();
void display();
struct node
  {
     int data;
     struct node *next;
  };
struct node *front=NULL,*rear=NULL;
void main()
  {
    int ch;
    clrscr();
    do
      {
        printf("\n1.enqueue\n2.dequeue\n3.display\n4.exit\n");
        printf("enter your choice\n");
        scanf("%d",&ch);
        switch(ch)
          {
             case 1:enq();
                    break;
```

```
                case 2:deq();
                        break;
                case 3:display();
                        break;
                case 4:exit(0);
                default:printf("\n invalid choice");
                }
              }while(ch<=4);
             getch();
           }
        void enq()
        {
          struct node *cptr;
          cptr=(struct node *)malloc(sizeof(struct node));
          printf("enter node data\n");
          scanf("%d",&cptr->data);
          if(rear==NULL)
             {
               cptr->next=NULL;
               front=cptr;
               rear=cptr;
             }
           else
             {
               cptr->next=NULL;
               rear->next=cptr;
               rear=cptr;
             }
         }
      void deq()
        {
          if(rear==NULL)
            {
```

```
            printf("\n list is empty,deletion is not possible");
        }
        else
        {
            printf("deleted element is %d\n",front->data);
            front=front->next;
        }
    }
void display()
    {
        struct node *temp;
        temp=front;
        printf("elements are :");
        while(temp!=NULL)
        {
            printf("\t %d",temp->data);
            temp=temp->next;
        }
    }
```

Output:-

```
1.enqueue
2.dequeue
3.display
4.exit
enter your choice
1
enter node data
7

1.enqueue
2.dequeue
3.display
4.exit
enter your choice
1
enter node data
13

1.enqueue
2.dequeue
3.display
4.exit
enter your choice
1
```

```
enter node data
24

1.enqueue
2.dequeue
3.display
4.exit
enter your choice
3
elements are :   7       13      24
1.enqueue
2.dequeue
3.display
4.exit
enter your choice
2
deleted element is 7

1.enqueue
2.dequeue
3.display
4.exit
```



```
enter your choice
2
deleted element is 7

1.enqueue
2.dequeue
3.display
4.exit
enter your choice
2
deleted element is 13

1.enqueue
2.dequeue
3.display
4.exit
enter your choice
3
elements are :   24
1.enqueue
2.dequeue
3.display
4.exit
enter your choice
4_
```

Program 6

(a) Aim: Write a C-program to implement Stack using Arrays and perform following operations.

(i) Push()
(ii) Pop()
(iii) Display()
(iv) Exit()

Source Code:- { stack.c }

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 6
int menu();
void push();
void pop();
void display();
int stack[MAX];
int top=0;
int main()
  {
   int ch;
 printf("\nStack using array");
 do
    {
      ch=menu();
      switch(ch)
        {
          case 1:push();
             break;
          case 2:pop();
             break;
          case 3:display();
             break;
```

```
                case 4:exit(0);
                default: printf("invalid choice");
                }
            getch();
          }while(1);
      }
int menu()
{
   int ch;
   printf("\n1.push()\n2.pop()\n3.display()\n4.Exit");
   printf("\nenter your choice");
   scanf("%d",&ch);
   return ch;
}
void display()
  {
      int i;
      if(top==0)
        {
          printf("\nstack is empty");
          return;
        }
      else
        {
          printf("\nelements in stack\n");
          for(i=top-1;i>=0;i--)
            printf("%d\n",stack[i]);
        }
  }
void push()
   {
     int data;
      if(top==MAX)
```

```
        {
          printf("\nstack overflow");
          return;
        }
      else
        {
          printf("\nenter data");
          scanf("%d",&data);
          stack[top]=data;
          top=top+1;
          printf("data is pushed into stack");
        }
   }
void pop()
  {
    if(top==0)
      {
        printf("\nstack underflow");
        return;
      }
    else
      {
         printf("\npopped elements is %d",stack[--top]);
      }
   }
```

Output:-

```
stack using arrays
1.push()
2.pop()
3.display()
4.Exit
enter your choice 1

enter data 12
data is pushed into stack
1.push()
2.pop()
3.display()
4.Exit
enter your choice 1

enter data 23
data is pushed into stack
1.push()
2.pop()
3.display()
4.Exit
enter your choice 1

enter data 34
data is pushed into stack
1.push()
2.pop()
3.display()
4.Exit
enter your choice 3

elements in stack
34
23
12

1.push()
2.pop()
3.display()
4.Exit
enter your choice 2

popped elements is 34
1.push()
2.pop()
3.display()
4.Exit
enter your choice 3
```

```
elements in stack
23
12

1.push()
2.pop()
3.display()
4.Exit
enter your choice 2

popped elements is 23
1.push()
2.pop()
3.display()
4.Exit
enter your choice 3

elements in stack
12

1.push()
2.pop()
3.display()
4.Exit
enter your choice 4_
```

Program 6

(b) Aim: Write a C-program to implement Queue Data Structure using arrays and perform following operations.

(i) Enqueue
(ii) Dequeue
(iii) Display
(iv) Exit

Sorce Code:- { queue.c }

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 10
void enq();
void display();
void deq();
int Q[size],front=-1,rear=-1;
void main()
  {
  int ch;
  clrscr();
  printf("*** Queue DataStructure ***");
   do
    {
      printf("\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit");
      printf("\nenter your choice: ");
      scanf("%d",&ch);
       switch(ch)
         {
            case 1:enq();
                   break;
            case 2:deq();
                   break;
            case 3:display();
```

```
                    break;
                case 4:exit(0);
                    break;
                default:printf("\ninvalid choice");
              }
           }while(ch<=4);
           getch();
      }
    void enq()
    {
        int data;
        if(rear==size-1)
          {
            printf("\nqueue is full");
          }
        else
          {
            printf("\nenter your data: ");
            scanf("%d", &data);
            rear=rear+1;
            Q[rear]=data;
          }
     }
    void deq()
      {
        if(rear==front)
          {
            printf("\nqueue is empty");
          }
        else
        {
            front=front+1;
            printf("\ndeleted element is %d",Q[front]);
```

```
    }
  }
void display()
  {
      int i;
      printf("\nthe elements in queue is:");
      for(i=front+1;i<=rear;i++)
        {
            printf("%d\t",Q[i]);
        }
  }
```

Output:-

```
*** Queue DataStructure ***
1.Enqueue
2.Dequeue
3.Display
4.Exit
enter your choice: 1

enter your data: 23
```

```
1.Enqueue
2.Dequeue
3.Display
4.Exit
enter your choice: 1

enter your data: 45

1.Enqueue
2.Dequeue
3.Display
4.Exit
enter your choice: 1

enter your data: 67_
```

```
2.Dequeue
3.Display
4.Exit
enter your choice: 3

the elements in queue is:23     45      67
1.Enqueue
2.Dequeue
3.Display
4.Exit
enter your choice: 2

deleted element is 23
1.Enqueue
2.Dequeue
3.Display
4.Exit
enter your choice: 3

the elements in queue is:45     67
1.Enqueue
2.Dequeue
3.Display
4.Exit
enter your choice:
```



```
enter your choice: 2

deleted element is 45
1.Enqueue
2.Dequeue
3.Display
4.Exit
enter your choice: 2

deleted element is 67
1.Enqueue
2.Dequeue
3.Display
4.Exit
enter your choice: 3

the elements in queue is:
1.Enqueue
2.Dequeue
3.Display
4.Exit
enter your choice: 4_
```

Program 7

(a) Aim: Write a C-program to convert infix to post-fix expression by using Stack.

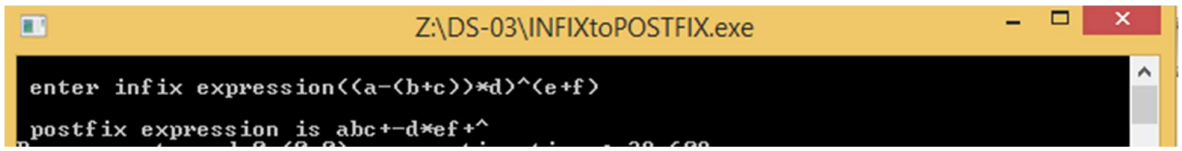
Source code:

```
#include<stdio.h>
#define size 30
int prec(char op);
void infix(char exp[30]);
int stack[size];
char post[size];
int top=-1;
char op;
int i=0,j=0,k;
int cr,st;
 main()
{
   char exp[30];
   printf("\n enter infix expression");
   scanf("%s",exp);
infix(exp);
 }
void infix(char exp[30])
{
while(exp[i]!='\0')
{
 if(exp[i]=='(')
  {
    top++;
   stack[top]=exp[i];
  }
 else if(exp[i]>='a'&&exp[i]<='z'||exp[i]>='A'&&exp[i]<='Z'||exp[i]>='0'&&exp[i]<='9')
 {
  post[j]=exp[i];
  j++;
 }
   else if(exp[i]=='+'||exp[i]=='-'||exp[i]=='*'||exp[i]=='/-'||exp[i]=='^')
   {
```

```
    cr=prec(exp[i]);
    st=prec(stack[top]);
    if(top==-1)
    {
    top++;
    stack[top]=exp[i];
    }
    else
    {
       while(st>=cr)
    {
    post[j]=stack[top];
    j++;
    top--;
    st=prec(stack[top]);
    }
    top++;
    stack[top]=exp[i];
   }
  }
 else if(exp[i]==')')
 {
    while(stack[top]!='(')
    {
       post[j]=stack[top];
       j++;
       top--;
    }
    top--;
 }
 else
    {
    printf("\ninvalid expression");
    }
    i++;
 }
```

```
 for(k=top;k>=0;k--)
 {
 post[j]=stack[k];
 j++;
 }
 printf("\n postfix expression is %s",post);
}
int prec(char op)
 {
 if(op=='+'||op=='-')
 return 1;
 else if(op=='*'||op=='/')
 return 2;
 else if(op=='^')
 return 3;
 else
 return 0;
 }
```

Output 1:



Output2:

Program 7

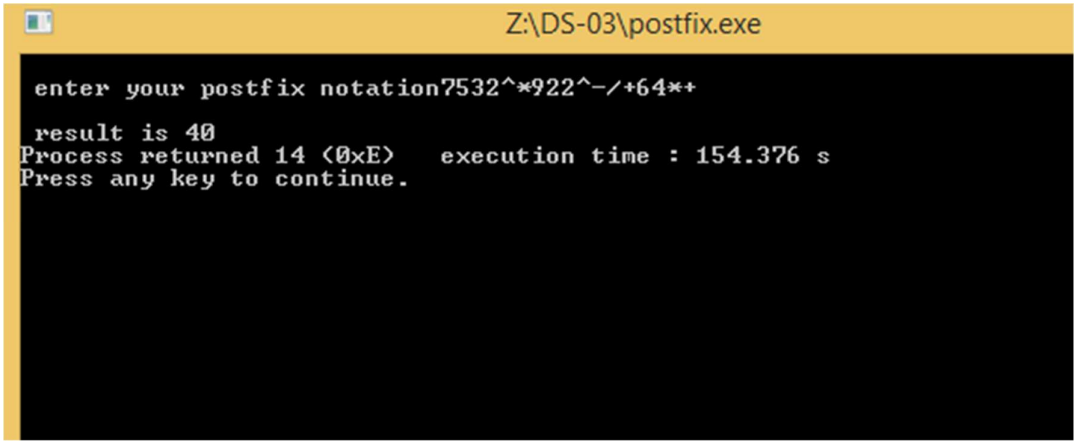
(b) Aim: Write a C-Program to Evolution of Post-fix expression by using Stack.

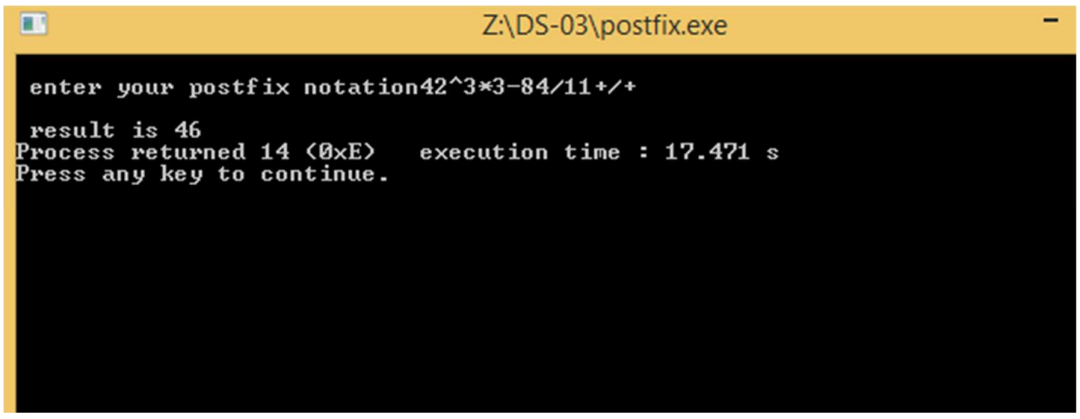
Source code: - { post.c }

```
#include<stdio.h>
#include<math.h>
#define size 30
int stack[size],top=-1;
void main()
{
inti,x,y;
charstr[50];
printf("\n enter your postfix notation");
scanf("%s",str);
for(i=0;str[i]!='\0';i++)
  {
switch (str[i])
   {
case  '+':  x=stack[top];
             y=stack[top-1];
             top--;
             stack[top]=y+x;
             break;
  case '-':  x=stack[top];
             y=stack[top-1];
            top--;
           stack[top]=y-x;
           break;
case '*':  x=stack[top];
            y=stack[top-1];
              top--;
             stack[top]=y*x;
             break;
  case '/':  x=stack[top];
             y=stack[top-1];
             top--;
```

```
                stack[top]=y/x;
                break;
  case '^':  x=stack[top];
                y=stack[top-1];
                top--;
                stack[top]=pow(y,x);
                break;
default :if(str[i]>=48&&str[i]<=57)
                {
                     top++;
                     stack[top]=str[i]-48;
                }
                    break;
                }
               }
printf("\n result is %d",stack[top]);
      }
```

Output :





Program 8

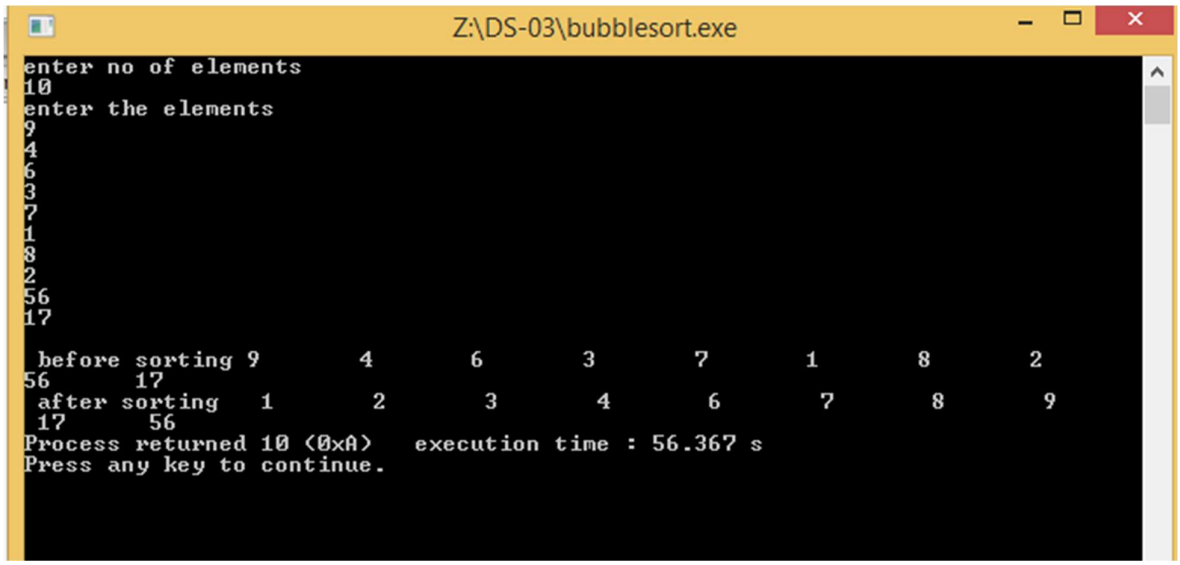
(a) Aim: Write a C program to implement bubble sorting.

Source code:

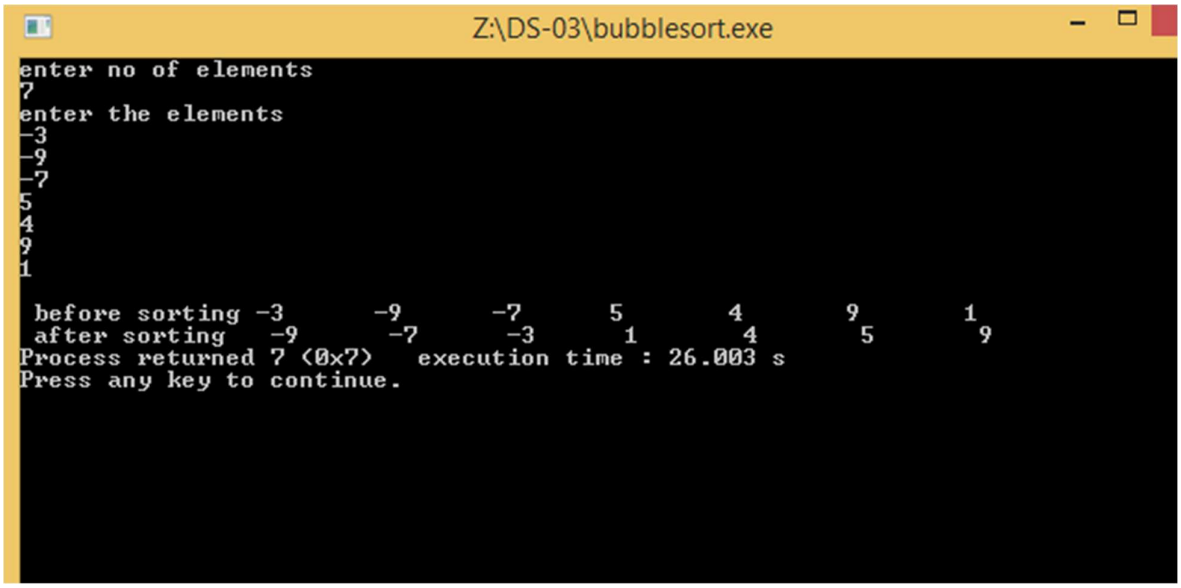
```
#include<stdio.h>
void main()
{
 int a[10],i,j,temp,n;
 printf("enter n value");
 scanf("%d",&n);
 printf("Enter elements are\n");
 for(i=0;i<n;i++)
 {
  scanf("%d",&a[i]);
  }
  printf("\nBefore sorting");
  for(i=0;i<n;i++)
  {
   printf("\t%d",a[i]);
   }
   for(i=1;i<n;i++)
   {
    temp=a[i];
    j=i-1;
    while(j>=0&&a[j]>temp)
    {
     a[j+1]=a[j];
     j=j-1;
     }
     a[j+1]=temp;
     }
     printf("\nAfter sorting");
     for(i=0;i<n;i++)
     {
      printf("\t%d",a[i]);
      }
```

```
}
```

Output 1:



Output 2:



Program 8

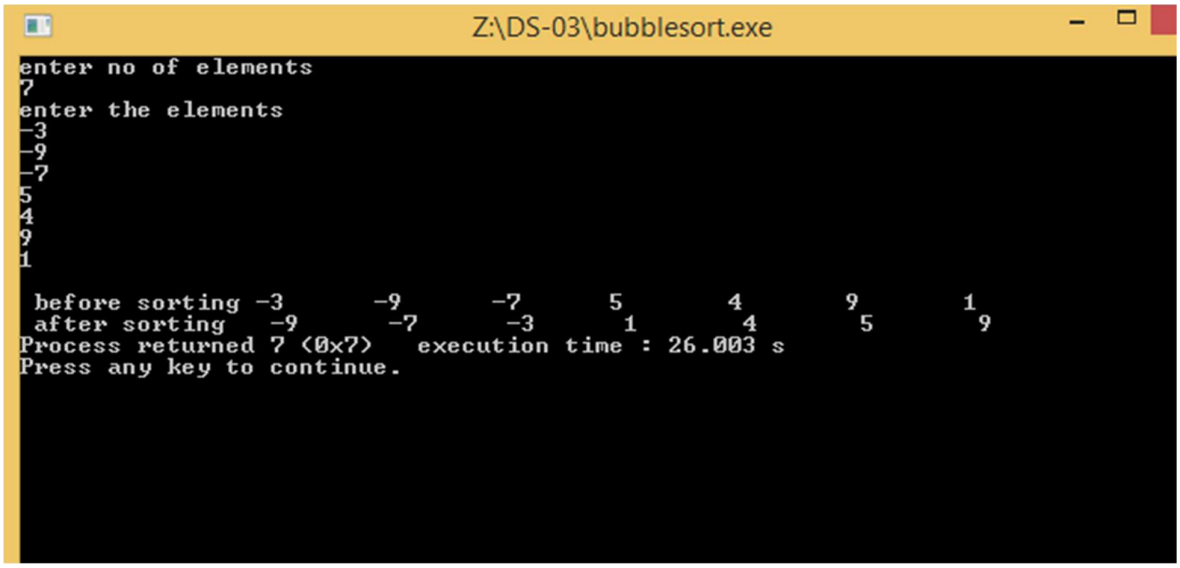
(b) Aim: Write a C program to implement insertion sorting.

Source code:

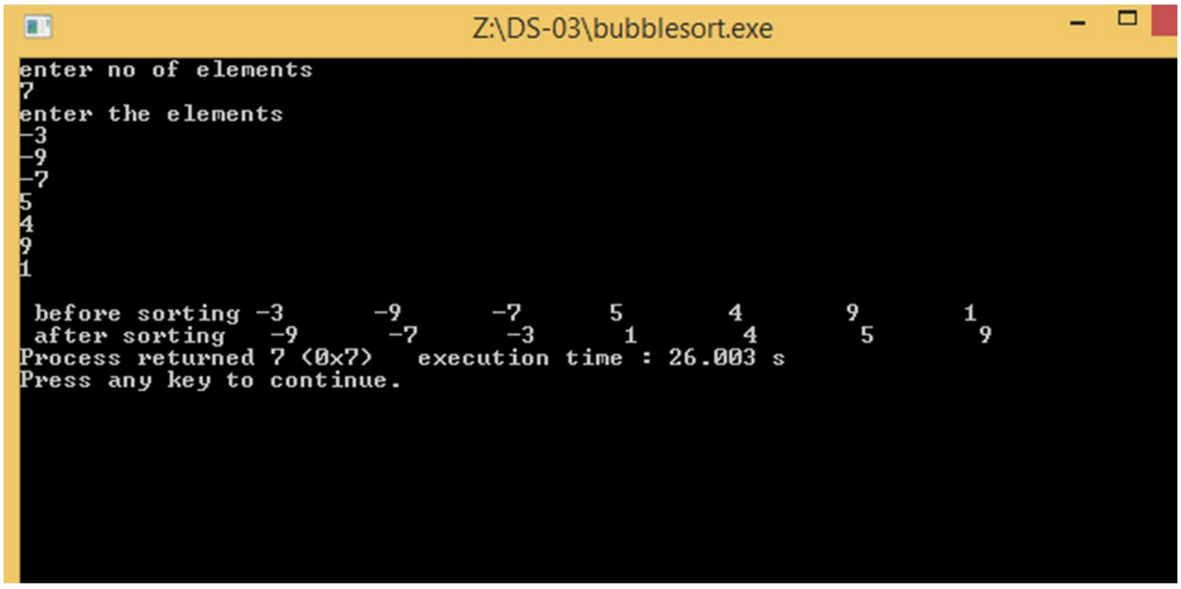
```
#include<stdio.h>
void main()
{
 int a[10],i,j,temp,n;
 printf("enter n value");
 scanf("%d",&n);
 printf("Enter elements are\n");
 for(i=0;i<n;i++)
 {
  scanf("%d",&a[i]);
  }
  printf("\nBefore sorting");
  for(i=0;i<n;i++)
  {
   printf("\t%d",a[i]);
   }
   for(i=1;i<n;i++)
   {
    temp=a[i];
    j=i-1;
    while(j>=0&&a[j]>temp)
    {
     a[j+1]=a[j];
     j=j-1;
     }
     a[j+1]=temp;
     }
     printf("\nAfter sorting");
     for(i=0;i<n;i++)
     {
      printf("\t%d",a[i]);
      }
```

```
}
```

Output 1:



Output 2:



Program 8

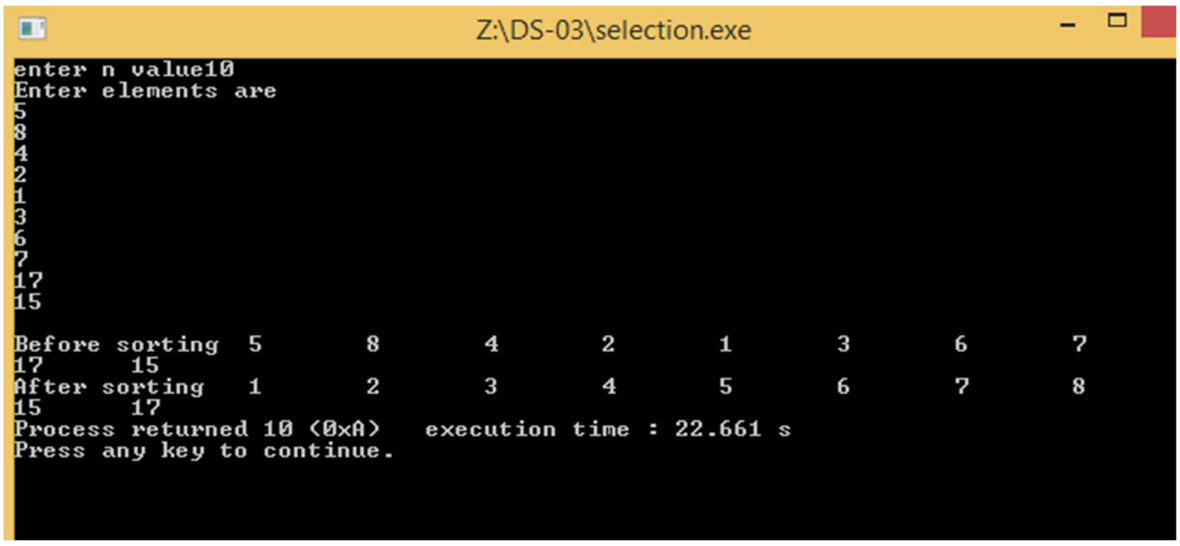
(c) Aim: Write a C program to implement Selection sorting.

Source code:

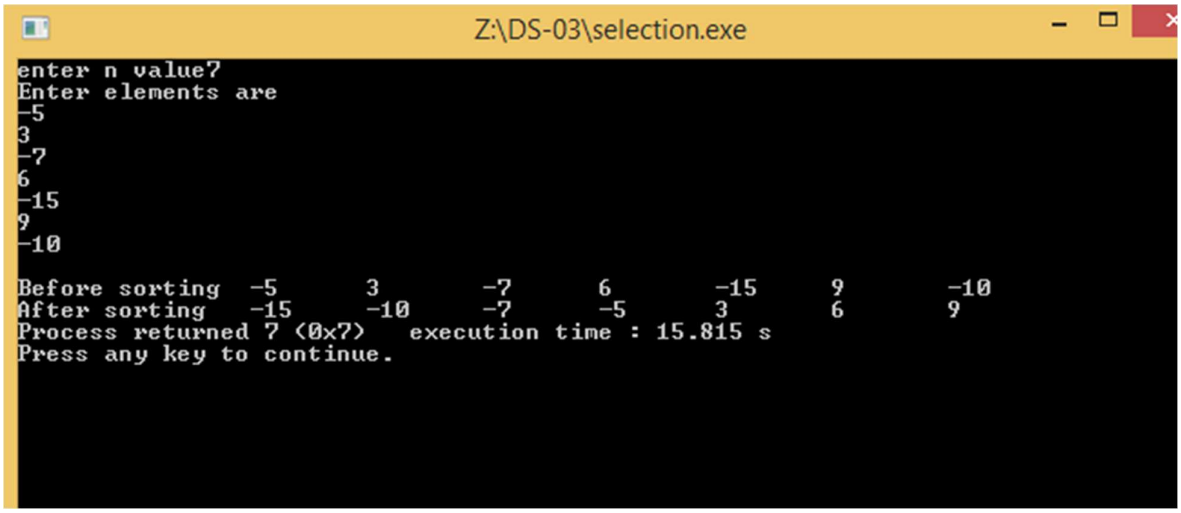
```
#include<stdio.h>
void main()
{
 int a[10],i,j,temp,n,min;
 printf("enter n value");
 scanf("%d",&n);
 printf("Enter elements are\n");
 for(i=0;i<n;i++)
 {
  scanf("%d",&a[i]);
  }
  printf("\nBefore sorting");
  for(i=0;i<n;i++)
  {
   printf("\t%d",a[i]);
   }
   for(i=0;i<n;i++)
   {
    min=i;
    for(j=i+1;j<n;j++)
    {
    if(a[j]<a[min])
    min=j;
     }
     temp=a[i];
     a[i]=a[min];
     a[min]=temp;
     }
     printf("\nAfter sorting");
     for(i=0;i<n;i++)
     {
      printf("\t%d",a[i]);
```

```
    }
  }
```

Output 1:



Output 2:



Program 8

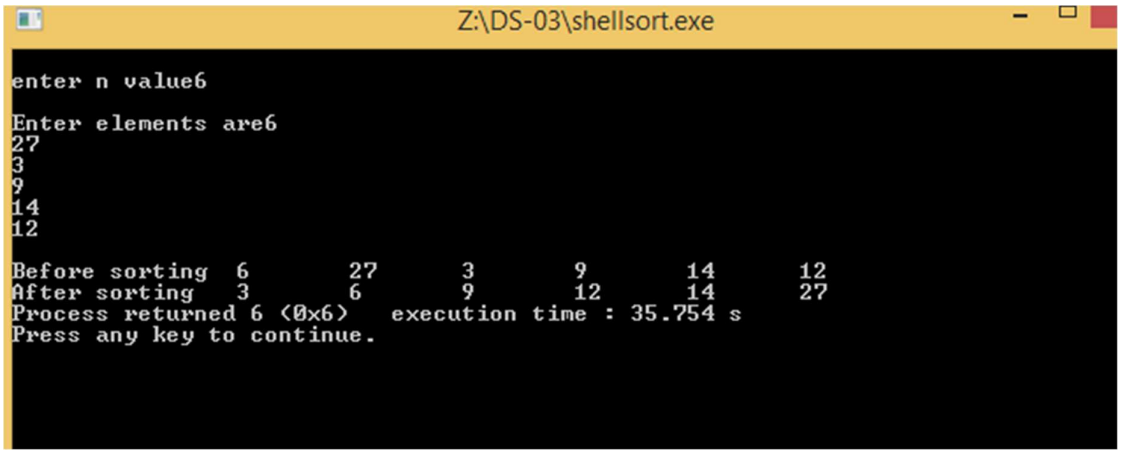
d) Aim: Write a C program to implement Shell sorting.

Source code:

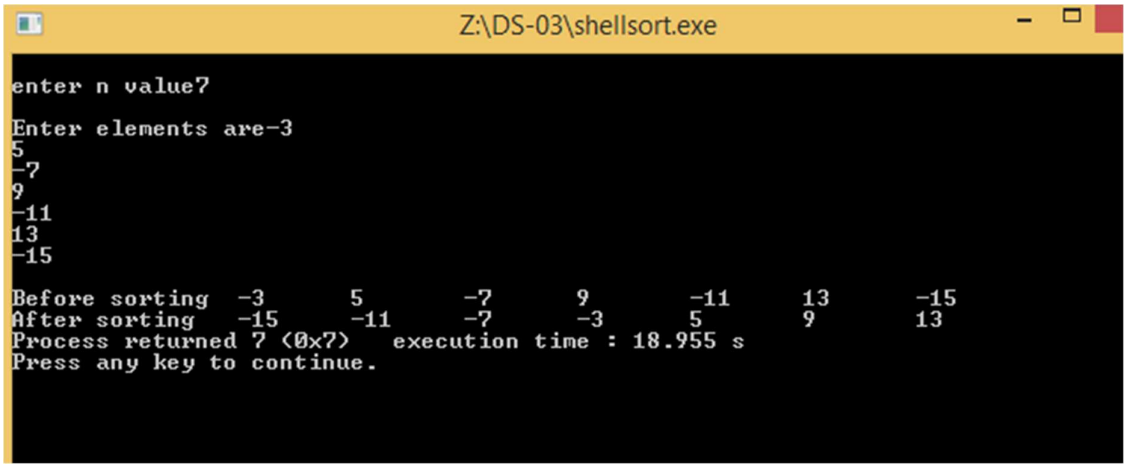
```
#include<stdio.h>
void main()
{
 int a[10],i,j,temp,n,k;
 printf("\nenter n value");
 scanf("%d",&n);
 printf("\nEnter elements are");
 for(i=0;i<n;i++)
 {
  scanf("%d",&a[i]);
  }
  printf("\nBefore sorting");
  for(i=0;i<n;i++)
  {
   printf("\t%d",a[i]);
   }
   for(i=n/2;i>0;i--)
   {
    for(j=1;j<n;j++)
    {
      for(k=j-1;k>=0;k=k-i)
      {
       if(a[k+i]>=a[k])
        break;
        else
        {
         temp=a[k];
         a[k]=a[k+i];
         a[k+i]=temp;
         }
        }
      }
```

```
}
    printf("\nAfter sorting");
    for(i=0;i<n;i++)
    {
       printf("\t%d",a[i]);
    }
}
```

Output 1:



Output 2:



Program 9

Aim: Write a C-program to create a Binary tree and perform following traversal techniques.

(a) Inorder
(b) Preorder
(c) Postorder
(d) Leafnodes

Source Code:- { tree1.c }

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
   {
        int data;
        struct node *left,*right;
    };
struct node *root=NULL;
void create();
void insert(struct node*,struct node *);
void preorder(struct node*);
void postorder(struct node*);
void inorder(struct node*);
void main()
      {
       int ch;
       clrscr();
       do
          {
            printf("\n1.create\n2.inorder\n3.preorder\n4.postorder\n5.exit\n");
            printf("enter ur choice\n");
            scanf("%d",&ch);
             switch(ch)
                {
                   case 1:create();
```

```
                    break;
                case 2:inorder(root);
                    break;
                case 3:preorder(root);
                    break;
                case 4:postorder(root);
                    break;
                case 5:exit(0);
                default:printf("invalid choice");
              }
            }while(ch<=5);
          getch();
        }
void create()
  {
    char ch;
    struct node *cptr;
     do
       {
         cptr=(struct node*)malloc(sizeof(struct node));
         printf("enter ur element\n");
         scanf("%d",&cptr->data);
         cptr->right=NULL;
         cptr->left=NULL;
         if(root==NULL)
           {
              root=cptr;
           }
         else
          {
             insert(root,cptr);
          }
          printf("do u want continue Y/N\n");
```

```
            ch=getch();
        }while(ch=='y'||ch=='Y');
    }

void insert(struct node *root,struct node *cptr)
   {
    char ch;
    printf("where to insert l/r %d\n",root->data);
    ch=getch();
    if(ch=='l'||ch=='L')
        {
          if(root->left==NULL)
               root->left=cptr;
          else
              insert(root->left,cptr);
        }
      else if(ch=='r'||ch=='R')
        {
          if(root->right==NULL)
              root->right=cptr;
              else
              insert(root->right,cptr);
        }
       else
        {
        printf("invalid selection\n");
        }
   }
void inorder(struct node *root)
 {
    struct node *tmp;
    tmp=root;
    if(tmp!=NULL)
```

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

Output:-

```
1.create
2.inorder
3.preorder
4.postorder
5.exit
enter ur choice
1
enter ur element
10
do u want continue Y/N
enter ur element
20
where to insert l/r 10
do u want continue Y/N
enter ur element
30
where to insert l/r 10
where to insert l/r 20
do u want continue Y/N
enter ur element
50
where to insert l/r 10
do u want continue Y/N
enter ur element
```

```
60
where to insert l/r 10
where to insert l/r 50
do u want continue Y/N
enter ur element
45
where to insert l/r 10
where to insert l/r 50
do u want continue Y/N
enter ur element
75
where to insert l/r 10
where to insert l/r 20
do u want continue Y/N

1.create
2.inorder
3.preorder
4.postorder
5.exit
enter ur choice
2
        30      20      75      10      45      50      60
```

```
1.create
2.inorder
3.preorder
4.postorder
5.exit
enter ur choice
3
        10      20      30      75      50      45      60
1.create
2.inorder
3.preorder
4.postorder
5.exit
enter ur choice
4
       30      75      20      45      60      50      10
1.create
2.inorder
3.preorder
4.postorder
5.exit
enter ur choice
5
```

Program-10

Aim: Write a C-Program to implement Binary Search Tree and perform following opearations.

(a) Create()
(b) Search()
(c) Delete()
(d) Display()

Source Code:- { bst.c }

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
struct node
  {
     int data;
     struct node *left,*right;
  };
struct node *root=NULL,*parent;
void create();
void insert(struct node*,struct node*);
void inorder(struct node *);
struct node *search(struct node *,int key);
void del(struct node *,int key);
void main()
   {
        int ch,key;
        struct node *tmp;
        clrscr();
        printf("\nBinary search tree");
        do
          {
               printf("\n1.create\n2.search\n3.delete\n4.display\n5.exit");
               printf("\nenter ur choice");
               scanf("%d",&ch);
```

```
                switch(ch)
                    {
                        case 1: create();
                                break;
                        case 2: printf("\nEnter the which element u want search");
                                scanf("%d",&key);
                                tmp=search(root,key);
                                 break;
                        case 3: printf("\n Enter which element u want delete");
                                scanf("%d",&key);
                                del(root,key);
                                break;
                        case 4: if(root==NULL)
                                 {
                                        printf("tree is not created");
                                 }
                                else
                                    {
                                        printf("\nThe tree is:");
                                        inorder(root);
                                    }
                                break;
                        case 5: exit(0);
                                break;
                        default: printf("invalid choice");
                    }
                }while(ch<=5);
            getch();
           }
    void create()
            {
                char ch;
                struct node *cptr;
```

```
                do
                 {
                 cptr=(struct node*)malloc(sizeof(struct node));
                 cptr->left=NULL;
                 cptr->right=NULL;
                 printf("\nEnter element ");
                 scanf("%d",&cptr->data);
                 if(root==NULL)
                         {
                         root=cptr;
                         }
                 else
                         {
                         insert(root,cptr);
                         }
                 printf("do u want continue Y/N");
                 ch=getch();
                }while(ch=='y'||ch=='Y');
        }
void insert(struct node *root,struct node *cptr)
          {
                if(cptr->data<root->data)
                 {
                 if(root->left==NULL)
                         {
                         root->left=cptr;
                         }
                 else
                         {
                         insert(root->left,cptr);
                         }
                 }
                else
```

```
                {
                    if(root->right==NULL)
                        {
                            root->right=cptr;
                        }
                    else
                        {
                            insert(root->right,cptr);
                         }
                }
        }
struct node *search(struct node *root,int key)
        {
                struct node *tmp;
                tmp=root;
                while(tmp!=NULL)
                   {
                        if(tmp->data==key)
                         {
                            printf("\nThe %d element is present",tmp->data);
                            return tmp;
                         }
                        parent=tmp;
                        if(key<tmp->data)
                         {
                            tmp=tmp->left;
                         }
                        else
                         {
                           tmp=tmp->right;
                         }
                   }
                return NULL;
```

```
        }
void del(struct node *root,int key)
    {
        struct node *tmp1,*tmp2,*parent;
        tmp1=search(root,key);
        parent=tmp1;
        if(tmp1->left!=NULL&&tmp1->right!=NULL)
          {
            tmp2=tmp1->right;
            while(tmp2!=NULL)
                {
                  if(tmp2->left!=NULL)
                    {
                      tmp2=tmp2->left;
                    }
                  else
                    {
                      parent=tmp2;
                      tmp1->data=tmp2->data;
                      tmp2=NULL;
                    }
                }
            parent->left=NULL;
            parent->right=NULL;
            tmp1->data=NULL;
            free(tmp1);
            printf("\nnow delete it");
            return;
          }
        else if(tmp1->left!=NULL&&tmp1->right==NULL)
            {
              if(parent->left==tmp1)
                  parent->left=tmp1->left;
```

```
                else
                        parent->right=tmp1->left;
                tmp1=NULL;
                free(tmp1);
                printf("\nnow delete it");
                return;
            }
        else if(tmp1->left==NULL&&tmp1->right!=NULL)
            {
                if(parent->left==tmp1)
                        parent->left=tmp1->right;
                else
                        parent->right=tmp1->right;
                tmp1=NULL;
                free(tmp1);
                printf("\nnow delete it");
                return;
             }
        else if(tmp1->left==NULL&&tmp1->right==NULL)
            {
                tmp1->data=NULL;
                free(tmp1);
                printf("\nnow delete it:");
                return;
            }
    }
void inorder(struct node *tmp)
    {
        if(tmp!=NULL)
            {
                inorder(tmp->left);
                printf("\t%d",tmp->data);
                inorder(tmp->right);
```

```
        }
    }
```

Output:-

```
Binary search tree
1.create
2.search
3.delete
4.display
5.exit
enter ur choice1

Enter element 27
do u want continue Y/N
Enter element 14
do u want continue Y/N
Enter element 36
do u want continue Y/N
Enter element 7
do u want continue Y/N
Enter element 15
do u want continue Y/N
1.create
2.search
3.delete
4.display
5.exit
enter ur choice4
```

```
The tree is:    7       14      15      27      36
1.create
2.search
3.delete
4.display
5.exit
enter ur choice3

 Enter which element u want delete14

The 14 element is present
now delete it
1.create
2.search
3.delete
4.display
5.exit
enter ur choice4

The tree is:    7       0       15      27      36
1.create
2.search
3.delete
4.display
5.exit
enter ur choice 2

Enter the which element u want search 27

The 27 element is present
1.create
2.search
3.delete
4.display
5.exit
enter ur choice 3

 Enter which element u want delete 27

The 27 element is present
now delete it
1.create
2.search
3.delete
4.display
5.exit
enter ur choice 4

The tree is:    7       0       15      0       36
1.create
2.search
3.delete
4.display
5.exit
```

Program 11

Aim : Write a C-Program to implement AVL Tree and perform following operations.

i) Insertion()
ii) Deletion()
iii) Searching()
iv) Traversing()
v) Exit()

Source Code:- { avltre.c }

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
struct avltreenode
{
        int data;
        int bfactor;
        struct avltreenode *link[2];

};
struct avltreenode *root=NULL;
void insertion(int data);
void deletion(int data);
void searchelement(int data);
void inordertraversal(struct avltreenode *root);
void main()
{
        int ch,key;

        printf("\tAVL Tree\n");
        while(1)
        {
                printf("1.Insertion\t2.Deletion\n3.Searching\t4.Traversing\n5.Exit\n");
                printf("Enter your choice\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:printf("Enter the key value\n");
                                        scanf("%d",&key);
                                        insertion(key);
                                        break;
                        case 2:printf("Enter the key value to delete");
                                       scanf("%d",&key);
                                       deletion(key);
                                        break;
                        case 3:printf("Enter the search key\n");
                                        scanf("%d",&key);
                                searchelement(key);

                                        break;
                        case 4:inordertraversal(root);
```

```
                                        printf("\n");
                                        break;
                        case 5:exit(0);
                        default:printf("\nWrong option!");
                }
                printf("\n");
        }
}
struct avltreenode *createnode(int data)
{
        struct avltreenode *newnode;
        newnode=(struct avltreenode *)malloc(sizeof(struct avltreenode));
        newnode->data=data;
        newnode->bfactor=0;
        newnode->link[0]=newnode->link[1]=NULL;
        return newnode;
}
void insertion(int data)
{
        struct avltreenode *bf,*parentbf,*tree,*subtree;
        struct avltreenode *current,*parent,*ptr,*newnode,*temp;
        int res=0,linkdir[32],i=0;
        if(!root)
        {
                root=createnode(data);
                return;
        }
        bf=parentbf=root;
        /*find the location for inserting the new node*/
        for(current=root;current!=NULL;ptr=current,current=current->link[res])
        {
                if(data==current->data)
                {
                        printf("\nCannot insert duplicate");
                        return;
                }
                res=(data>current->data)?1:0;
                parent=current;
                if(current->bfactor!=0)
                {
                        bf=current;
                        parentbf=ptr;
                        i=0;
                }
                linkdir[i++]=res;
        }
        newnode=createnode(data);
        parent->link[res]=newnode;
        res=linkdir[i=0];
        for (current=bf;current!=newnode;res=linkdir[++i])
        {
                if(res==0)
                current->bfactor--;
                else
```

```
            current->bfactor++;
            current=current->link[res];
        }
        if(bf->bfactor==2)
        {
            printf("\nbfactor=2");
            temp=bf->link[1];
            if(temp->bfactor==1)
            {
                subtree=temp;
                bf->link[1]=temp->link[0];
                temp->link[0]=bf;
                temp->bfactor=bf->bfactor=0;
            }
            else
            {
                subtree=temp->link[0];
                temp->link[0]=subtree->link[1];
                subtree->link[1]=temp;
                bf->link[1]=subtree->link[0];
                subtree->link[0]=bf;
                if(subtree->bfactor==-1)
                {
                    bf->bfactor=0;
                    temp->bfactor=1;
                }
                else if(subtree->bfactor==0)
                {
                    bf->bfactor=0;
                    temp->bfactor=0;
                }
                else if(subtree->bfactor==1)
                {
                    bf->bfactor=-1;
                    temp->bfactor=0;
                }
                subtree->bfactor=0;
            }
        }
        else if(bf->bfactor==-2)
        {
            temp=bf->link[0];
            if(temp->bfactor==-1)
            {
                subtree=temp;
                bf->link[0]=temp->link[1];
                temp->link[1]=bf;
                temp->bfactor=bf->bfactor=0;
            }
            else
            {
                subtree=temp->link[1];
                temp->link[1]=subtree->link[0];
                subtree->link[0]=temp;
```

```
                        bf->link[0]=subtree->link[1];
                        subtree->link[1]=bf;
                        if(subtree->bfactor==-1)
                        {
                                bf->bfactor=1;
                                temp->bfactor=0;
                        }
                        else if(subtree->bfactor==0)
                        {
                                bf->bfactor=0;
                                temp->bfactor=0;
                        }
                        else if(subtree->bfactor==1)
                        {
                                bf->bfactor=0;
                                temp->bfactor==-1;
                        }
                        subtree->bfactor=0;
                }
        }
        else
        {
        return;
        }
        if(bf==root)
        {
                root=subtree;
                return;
        }
        if(bf!=parentbf->link[0])
        {
                parentbf->link[1]=subtree;
        }
        else
        {
                parentbf->link[0]=subtree;
        }
        return;
}
void deletion(int data)
{
        int linkdir[32],res=0,i=0,j=0,index=0,key;
        struct avltreenode *ptr[32],*current,*temp,*x,*y,*z;
        current=root;
        if(!root)
        {
                printf("\nTree not present");
                return;
        }
        if((root->data==data)&&(root->link[0]==NULL)&&(root->link[1]==NULL))
        {
                free(root);
                root=NULL;
                return;
```

```
}
while(current!=NULL)
{
        if(current->data==data)
        break;
        res=data>current->data?1:0;
        linkdir[i]=res;
        ptr[i++]=current;
        current=current->link[res];
}
if(!current)
{
        printf("\nGiven data is not present");
        return;
}
index=linkdir[i-1];
temp=current->link[1];
if(current->link[1]==NULL)
{
        if(i==0)
        {
                temp=current->link[0];
                free(current);
                root=temp;
                return;
        }
        else
        {
                ptr[i-1]->link[index]=current->link[0];
        }
}
else if(temp->link[0]==NULL)
{
        temp->link[0]=current->link[0];
        temp->bfactor=current->bfactor;
        if(i>0)
        {
                ptr[i-1]->link[index]=temp;
        }
        else
        {
                root=temp;
        }
        linkdir[i]=1;
        ptr[i++]=temp;
}
else
{
        j=i++;
        while(1)
        {
                linkdir[i]=0;
                ptr[i++]=temp;
                x=temp->link[0];
```

```
                if(x->link[0]==NULL)
                break;
                temp=x;
        }
        x->link[0]=current->link[0];
        temp->link[0]=x->link[1];
        x->link[1]=current->link[1];
        x->bfactor=current->bfactor;
        if(j>0)
        {
                ptr[j-1]->link[index]=x;
        }
        else
        {
                root=x;
        }
        linkdir[j]=1;
        ptr[j]=x;
}
free(current);
for(i=i-1;i>0;i=i--)
{
        x=ptr[i];
        if(linkdir[i]==0)
        {
                x->bfactor++;
                if(x->bfactor==1)
                {
                        break;
                }
                else if(x->bfactor==2)
                {
                        y=x->link[1];
                        if(y->bfactor==-1)
                        {
                                z=y->link[0];
                                y->link[0]=z->link[1];
                                z->link[1]=y;
                                x->link[1]=z->link[0];
                                z->link[0]=x;
                                if(z->bfactor==-1)
                                {
                                        x->bfactor=0;
                                        y->bfactor=1;
                                }
                                else if(z->bfactor==0)
                                {
                                        x->bfactor=0;
                                        y->bfactor=0;
                                }
                                else if(z->bfactor==1)
                                {
                                        x->bfactor=-1;
                                        y->bfactor=0;
```

```
                        }
                        z->bfactor=0;
                        if(i>0)
                        {
                                index=linkdir[i-1];
                                ptr[i-1]->link[index]=z;
                        }
                        else
                        {
                                root=z;
                        }
                }
                else
                {
                        x->link[1]=y->link[0];
                        y->link[0]=x;
                        if(i>0)
                        {
                                index=linkdir[i-1];
                                ptr[i-1]->link[index]=y;
                        }
                        else
                        {
                                root=y;
                        }
                        if(y->bfactor==0)
                        {
                                x->bfactor=1;
                                y->bfactor=-1;
                                break;
                        }
                        else
                        {
                                x->bfactor=0;
                                y->bfactor=0;
                        }
                }
        }

}
else
{
        x->bfactor--;
        if(x->bfactor==-1)
        {
                break;
        }
        else if(x->bfactor==-2)
        {
                y=x->link[0];
                if(y->bfactor==1)
                {
                        z=y->link[1];
                        y->link[1]=z->link[0];
```

```
                z->link[0]=y;
                x->link[0]=z->link[1];
                z->link[1]=x;
                if(z->bfactor==-1)
                {
                        x->bfactor=1;
                        y->bfactor=0;
                }
                else if(z->bfactor==0)
                {
                        x->bfactor=0;
                        y->bfactor=0;
                }
                else if(z->bfactor==1)
                {
                        x->bfactor=0;
                        y->bfactor=-1;
                }
                z->bfactor=0;
                if(i>0)
                {
                        index=linkdir[i-1];
                        ptr[i-1]->link[index]=z;
                }
                else
                {
                        root==z;
                }
        }
        else
        {
                x->link[0]=y->link[1];
                y->link[1]=x;
                if(i<0)
                {
                        root=y;
                }
                else
                {
                        index=linkdir[i-1];
                        ptr[i-1]->link[index]=y;
                }
                if(y->bfactor==0)
                {
                        x->bfactor=-1;
                        y->bfactor=1;
                        break;
                }
                else
                {
                        x->bfactor=0;
                        y->bfactor=0;
                }
        }
```

```
                        }
                }
        }
}
void searchelement(int data)
{
        int flag=0,res=0;
        struct avltreenode *node=root;
        if(!node)
        {
                printf("avl tree unavailable!\n");
                return;
        }
        while(node!=NULL)
        {
                if(data==node->data)
                {
                        printf("%d is present in avl tree\n",data);
                        flag=1;
                        break;
                }
                res=data>node->data?1:0;
                node=node->link[res];
        }
        if(!flag)
        printf("Search element not found in avl tree");
        return;
}
void inordertraversal(struct avltreenode *mynode)
{
        if(mynode)
        {
                inordertraversal(mynode->link[0]);
                printf("\t%d",mynode->data);
                inordertraversal(mynode->link[1]);
        }
        return;
}
```

Output:-

```
        AVL Tree
1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
3

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
15

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
25
```

```
bfactor=2
1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
10

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
14

bfactor=2
1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
83
1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
9

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
11

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
13

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
24

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
1
Enter the key value
7
```

```
bfactor=2
1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
4
        3       7       9       10      11      13      14      15      24
25      83

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
3
Enter the search key
7
7 is present in avl tree

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
3
Enter the search key
24
24 is present in avl tree

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
2
Enter the key value to delete 11

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
4
        3       7       9       10      13      14      15      24      25
83

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
2
Enter the key value to delete 10

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
4
        3       7       9       13      14      15      24      25      83

1.Insertion     2.Deletion
3.Searching     4.Traversing
5.Exit
Enter your choice
5
```