Program-01:

```
#include<stdio.h>
int arr[10]={0,20,50,10,30,40,60,0,0,0};
int L=1,U=6;
int Search_Array(int key)
{
     int i=L,found=0,location=0;
     while((i<=U) && (found==0))
    {
        if(arr[i]==key)
       {
           found=1;
           location=i;
       }
       else
       {
           i=i+1;
       }
    }
    if(found==0)
      printf("\n\nSearch is unsuccessful. %d is not found in the array",key);
    else
    {
      printf("\n\nSearch is successful. %d is found in number %d location of the array.",key,location);
    }
    return location;
 }

int main()
{
    int key,loc;
    printf("Enter a key to search in the array ::: ");
    scanf("%d",&key);
    loc=Search_Array(key);
    return 0;
}
```

Program-02:

```
#include<stdio.h>
int arr[10]={0,20,50,10,30,40,60,0,0,0};
int L=1,U=6;
int Insert_Array(int key,int location)
{
    if(arr[U+1]!=0)
        return 0;
    else
    {
        int i=U;
        while (i>=location)
        {
            arr[i+1]=arr[i];
            i=i-1;
        }
        arr[location]=key;
        U=U+1;
        return 1;
    }
}

int main()
{
    int key,loc,flag;
    printf("Enter the location where you want to insert ::: ");
    scanf("%d",&loc);
    printf("Enter the value that you want to insert ::: ");
    scanf("%d",&key);
    flag=Insert_Array(key,loc);
    if (flag==1)
    {
        printf("\n\nAfter insert %d at %d location the array is as follow:::\n",key,loc);
        for(int i=L;i<=U;i++)
        {
            printf(" %d ",arr[i]);
        }
    }
    else
         printf("\n\nArray is Full. No insertion is possible.");
    return 0;
 }
```

Program-01:

```
#include<stdio.h>
int arr[10]={0,20,50,10,30,40,60,0,0,0};
int L=1,U=6;
int Sort_Array()
{
    int i=U,j,temp;
    while(i>=L)
      {
        j=L;
        while(j<i)
          {
            if(arr[j]>arr[j+1])
               {
                   temp=arr[j];
                   arr[j]=arr[j+1];
                   arr[j+1]=temp;
               }
            j=j+1;
          }
        i=i-1;
      }
 }

int main()
{
     int i;
     printf("Before Sort the array is ::: ");
     for(i=L;i<=U;i++)
          printf(" %d ",arr[i]);
     Sort_Array();
     printf("\n\nAfter Sort the array is ::: ");
     for(i=L;i<=U;i++)
          printf(" %d ",arr[i]);
     return 0;
}
```

Program-02:

```
#include<stdio.h>
int arr[10]={0,20,50,10,30,40,60,0,0,0};
int L=1,U=6;
int Search_Array(int key)
{
    int i=L,found=0,location=0;
    while((i<=U) && (found==0))
    {
        if(arr[i]==key)
        {
            found=1;
            location=i;
    }
    else
      i=i+1;
  }
  return location;
}

void Array_Delete(int key)
{
  int i=Search_Array(key);
  if(i==0)
      printf("\n\n%d is not found. No DELETION !!!\n",key);
  else
  {
      while(i<U)
      {
         arr[i]=arr[i+1];
         i=i+1;
      }
   }
   arr[U]=0;
   U=U-1;
 }

 int main()
 {
   int key,i;
   printf("Before deletion the array is ::: ");
   for(i=1;i<=U;i++)
     printf(" %d ",arr[i]);
   printf("\n\nEnter a key to delete from the array ::: ");
   scanf("%d",&key);
   Array_Delete(key);
   printf("\n\nAfter deletion the array is ::: ");
   for(i=1;i<=U;i++)
     printf(" %d ",arr[i]);
   printf("\n\n\n");
 }
```

Program-01:

```
#include<stdio.h>
int A[20];
int A1[10]={0,05,15,25,35,45,55,65,75,85};
int A2[10]={0,10,20,30,40,50,60,70,80,90};
int l1=1,l2=1,u1=9,u2=9;
int l=1,u;
int merge()
{
    int i1=l1,i2=l2;
    int i=l;
    while(i1<=u1)
    {
        A[i]=A1[i1];
        i++;
        i1++;
    }
    while(i2<=u2)
    {
        A[i]=A2[i2];
        i++;
        i2++;
    }
 }

 int main()
 {
     int i;
     printf("Array 1 : ");
     for(i=l1;i<=u1;i++)
     {
         printf("%d  ",A1[i]);
     }
     printf("\n\n");
     printf("Array 2 : ");
     for(i=l2;i<=u2;i++)
        printf("%d  ",A2[i]);
     printf("\n\n");
     u=u1+u2-l2+1;
     printf("Merged array :: ");
     merge();
     for(i=1;i<=u;i++)
     {
        printf(" %d ",A[i]);
     }
     return 0;
  }
```

Program-02:

```
#include<stdio.h>
int Data[14]={0,10,05,0,50,100,65,0,30,0,0,48,0,80};
int Link[14]={0,8,11,0,1,1000,13,0,2,0,0,6,0,5};
void SL_Traverse(int Header)
{
    int ptr=Header;
    while(ptr!=1000)
      {
          printf(" %d ",Data[ptr]);
          ptr=Link[ptr];
      }
}

int main()
{
      int Header;
      printf("Enter the location of Header ::: ");
      scanf("%d",&Header);
      printf("Traversal of the Linked List ::: ");
      SL_Traverse(Header);
      return 0;
}
```

Program :

```
#include<stdio.h>
int Data[16]={0,10,05,0,50,100,65,0,30,0,0,48,0,80,0,0};
int Link[16]={0,8,11,7,1,1000,13,9,2,10,12,6,14,5,15,1000};
int Header=4,avail=3;
void SL_Traverse()
 {
    int ptr=Header;
    printf("\n\nTraversal of the Linked List ::: ");
    while(ptr!=1000)
    {
        printf(" %d ",Data[ptr]);
        ptr=Link[ptr];
    }
 }

 void Avail_Traverse()
 {
    int ptr=avail;
    printf("\n\nTraversal of the Available positions ::: ");
    while(ptr!=1000)
    {
        printf(" %d ",ptr);
        ptr=Link[ptr];
    }
 }

 int GetNode()
 {
    if(avail==0)
    {
         printf("\n\nInsufficient memory: Unable to allocate memory!!!");
         return 0;
    }
    else
    {
        int temp=avail;
        avail=Link[avail];
        return temp;
    }
 }

 void SL_Insert_Front()
 {
     int value;
     int new_node=GetNode();
     printf("\nEnter value to insert at FRONT of SL::: ");
     scanf("%d",&value);
```

```
   if(new_node==0)
   {
        printf("\nMemory Underflow: No insertion");
        return;
   }
   else
   {
        Link[new_node]=Header;
        Data[new_node]=value;
        Header=new_node;
   }
 }

 void SL_Insert_End()
 {
    int value,ptr;
    int new_node=GetNode();
    printf("\nEnter value to insert at the END of SL::: ");
    scanf("%d",&value);
    if(new_node==0)
    {
        printf("\nMemory Underflow: No insertion");
        return;
    }
    else
    {
        ptr=Header;
        while(Link[ptr]!=1000)
          ptr=Link[ptr];
        Link[ptr]=new_node;
        Data[new_node]=value;
        Link[new_node]=1000;
    }
}

void SL_Insert_Any()
{
   int value,ptr,key;
   int new_node=GetNode();
   if(new_node==0)
   {
        printf("\nMemory Underflow: No insertion");
        return;
   }
   else
   {
        printf("\nEnter the value after which you want to insert :::");
        scanf("%d",&key);
        printf("\nEnter value to insert after %d of SL::: ",key);
        scanf("%d",&value);
        ptr=Header;
        while((Data[ptr]!=key)&&(Link[ptr]!=1000))
            ptr=Link[ptr];
          if(Link[ptr]==1000)
              printf("\n%d is not available in the list ::: ");
          else
          {
                Link[new_node]=Link[ptr];
                Data[new_node]=value;
                Link[ptr]=new_node;
             }
      }
 }

int main()
{
     int choice;
     while(1)
     {
         printf("\n\nEnter 1 for Print Linked List");
         printf("\nEnter 2 for Print Avail List");
         printf("\nEnter 3 for insert element in Front of SL");
         printf("\nEnter 4 for insert element in End of SL");
         printf("\nEnter 5 for insert element in ANY position of SL");
         printf("\nEnter 10 for Exit ::: ");
         scanf("%d",&choice);
         if(choice==10)
              break;
         switch(choice)
         {
              case 1: SL_Traverse(); break;
              case 2: Avail_Traverse(); break;
              case 3: SL_Insert_Front(); break;
              case 4: SL_Insert_End(); break;
              case 5: SL_Insert_Any(); break;
              default: printf("\nWrong Choice !!!");
         }
     }
}
```

Program :

```
#include<stdio.h>
int Data[16]={0,10,05,0,50,100,65,0,30,0,0,48,0,80,0,0};
int Link[16]={0,8,11,7,1,1000,13,9,2,10,12,6,14,5,15,1000};
int Header=4,avail=3;
void SL_Traverse()
{
    int ptr=Header;
    printf("\n\nTraversal of the Linked List ::: ");
    while(ptr!=1000)
    {
         printf(" %d ",Data[ptr]);
         ptr=Link[ptr];
    }
}

void Avail_Traverse()
{
    int ptr=avail;
    printf("\n\nTraversal of the Available positions ::: ");
    while(ptr!=1000)
    {
         printf(" %d ",ptr);
         ptr=Link[ptr];
    }
}

int ReturnNode(int ptr)
{
     int ptr1=avail;
     while(Link[ptr1]!=1000)
          ptr1=Link[ptr1];
     Link[ptr1]=ptr;
     Link[ptr]=1000;
}

void SL_Delete_Front()
{
     int ptr1,ptr=Header;
     if(ptr==1000)
     {
          printf("\nThe list is empty: No deletion");
          return;
     }
     else
     {
           ptr1=Link[ptr];
           Header=ptr1;
           ReturnNode(ptr);
     }
    SL_Traverse();
}

void SL_Delete_End()
{
    int ptr1,ptr=Header;
    if(Link[ptr]==1000)
    {
        printf("\nThe list is empty: No deletion");
        return;
    }
    else
    {
         while(Link[ptr]!=1000)
         {
              ptr1=ptr;
              ptr=Link[ptr];
         }
         Link[ptr1]=1000;
         ReturnNode(ptr);
     }
     SL_Traverse();
 }

 void SL_Delete_Any()
 {
     int key,ptr,ptr1=Header;
     ptr=Link[ptr1];
     printf("\n\nWhich value you want to delete? ::: ");
     scanf("%d",&key);
     while(ptr!=1000)
     {
         if(Data[ptr]!=key)
         {
             ptr1=ptr;
             ptr=Link[ptr];
         }
         else
         {
             Link[ptr1]=Link[ptr];
             ReturnNode(ptr);
             SL_Traverse();
             return;
         }
     }
     if(ptr==1000)
     {
         printf("\n%d is not exists in the list. : No deletion\n",key);
         return;
     }
     SL_Traverse();
  }
```

```
int main()
{
    int choice;
    while(1)
    {
        printf("\n\nEnter 1 for Print Linked List");
        printf("\nEnter 2 for Print Avail List");
        printf("\nEnter 3 for DELETE element from Front of SL");
        printf("\nEnter 4 for DELETE element from END of SL");
        printf("\nEnter 5 for DELETE element from ANY position of SL");
        printf("\nEnter 6 for Exit ::: ");
        scanf("%d",&choice);
        if(choice==6)
           break;
        switch(choice)
        {
            case 1: SL_Traverse(); break;
            case 2: Avail_Traverse(); break;
            case 3: SL_Delete_Front(); break;
            case 4: SL_Delete_End(); break;
            case 5: SL_Delete_Any(); break;
            default: printf("\nWrong Choice !!!");
        }
    }
 }
```

Program :

```
#include<stdio.h>
int Data[18]={0,10,05,0,50,100,65,0,30,0,0,48,0,80,0,0,0,0};
int RLink[18]={0,8,11,7,1,1000,13,9,2,10,12,6,14,5,15,1000,4,3};
int LLink[18]={0,4,8,17,16,13,11,3,1,7,9,2,10,6,12,14,1000,1000};
int Header=16,avail=17;
void DL_Traverse()
{
    int temp;
    int ptr=RLink[Header];
    printf("\n\nTraversal of Double the Linked List from left to right::: ");
    while(ptr!=1000)
    {
        printf(" %d ",Data[ptr]);
        temp=ptr;
        ptr=RLink[ptr];
    }
    printf("\n\nTraversal of Double the Linked List from Right to Left::: ");
    ptr=temp;
    while(ptr!=Header)
    {
        printf(" %d ",Data[ptr]);
        ptr=LLink[ptr];
    }
}
void Avail_Traverse()
{
    int ptr=RLink[avail];
    printf("\n\nTraversal of the Available positions ::: ");
    while(ptr!=1000)
    {
        printf(" %d ",ptr);
        ptr=RLink[ptr];
    }
}
int GetNode()
{
    if(avail==0)
    {
         printf("\n\nInsufficient memory: Unable to allocate memory!!!");
         return 0;
    }
    else
    {
        int temp=RLink[avail];
        RLink[avail]=RLink[temp];
        return temp;
    }
}
```

```
void DL_Insert_Front()
{
    int value,ptr;
    ptr=RLink[Header];
    int new_node=GetNode();
    printf("\nEnter value to insert at front of DL::: ");
    scanf("%d",&value);
    if(new_node!=0)
    {
         LLink[new_node]=Header;
         RLink[Header]=new_node;
         RLink[new_node]=ptr;
         LLink[ptr]=new_node;
         Data[new_node]=value;
    }
    else
    {
         printf("\nUnable to allocate memory: Insertion is not possible");
    }
    DL_Traverse();
 }
 void DL_Insert_End()
 {
    int ptr=Header,value,temp;
    while(RLink[ptr]!=1000)
    {
       ptr=RLink[ptr];
    }
    int new_node=GetNode();
    printf("\nEnter value to insert at end of DL::: ");
    scanf("%d",&value);
    if(new_node!=0)
    {
       LLink[new_node]=ptr;
       RLink[ptr]=new_node;
       RLink[new_node]=1000;
       Data[new_node]=value;
    }
    else
    {
       printf("\nUnable to allocate memory: Insertion is not possible.");
    }
    DL_Traverse();
 }
 void DL_Insert_Any()
 {
    int value,ptr,ptr1,key;
    printf("\nWhere do you want to insert new value of DL::: ");
    scanf("%d",&key);
    printf("\nEnter value to insert after the place of %d of DL::: ",key);
    scanf("%d",&value);
    int new_node=GetNode();
    ptr=Header;
    while((Data[ptr]!=key) && (RLink[ptr]!=1000))
```

```
  {
       ptr=RLink[ptr];
  }
  if(new_node==0)
  {
      printf("\nUnable to allocate memory: Insertion is not possible");
      return;
  }
  if(RLink[ptr]==1000)
  {
      LLink[new_node]=ptr;
      RLink[ptr]=new_node;
      RLink[new_node]=1000;
      Data[new_node]=value;
  }
  else
  {
       ptr1=RLink[ptr];
       LLink[new_node]=ptr;
       RLink[new_node]=ptr1;
       RLink[ptr]=new_node;
       LLink[ptr1]=new_node;
       ptr=new_node;
       Data[new_node]=value;
  }
  DL_Traverse();
}

int main()
{
    int choice;
    DL_Traverse();
    while(1)
    {
        printf("\n\nEnter 1 for Print Double Linked List");
        printf("\nEnter 2 for Print Avail List");
        printf("\nEnter 3 for insert element in Front of DL");
        printf("\nEnter 4 for insert element in End of DL");
        printf("\nEnter 5 for insert element in Any position of DL");
        printf("\nEnter 6 for Exit ::: ");
        scanf("%d",&choice);
        if(choice==6)
          break;
        switch(choice)
        {
            case 1: DL_Traverse(); break;
            case 2: Avail_Traverse(); break;
            case 3: DL_Insert_Front(); break;
            case 4: DL_Insert_End();break;
            case 5: DL_Insert_Any();break;
            default: printf("\nWrong Choice !!!");
        }
    }
 }
```

Program :

```
#include<stdio.h>
int Data[18]={0,10,05,0,50,100,65,0,30,0,0,48,0,80,0,0,0,0};
int RLink[18]={0,8,11,7,1,1000,13,9,2,10,12,6,14,5,15,1000,4,3};
int LLink[18]={0,4,8,17,16,13,11,3,1,7,9,2,10,6,12,14,1000,1000};
int Header=16,avail=17;
void DL_Traverse()
{
    int temp;
    int ptr=RLink[Header];
    printf("\n\nTraversal of Double the Linked List from left to right::: ");
    while(ptr!=1000)
    {
        printf(" %d ",Data[ptr]);
        temp=ptr;
        ptr=RLink[ptr];
    }
    printf("\n\nTraversal of Double the Linked List from Right to Left::: ");
    ptr=temp;
    while(ptr!=Header)
    {
         printf(" %d ",Data[ptr]);
         ptr=LLink[ptr];
    }
}

void Avail_Traverse()
{
    int ptr=RLink[avail];
    printf("\n\nTraversal of the Available positions ::: ");
    while(ptr!=1000)
    {
         printf(" %d ",ptr);
         ptr=RLink[ptr];
    }
}

void ReturnNode(int ptr)
{
     int ptr1=avail;
     while(RLink[ptr1]!=1000)
          ptr1=RLink[ptr1];
     RLink[ptr1]=ptr;
     RLink[ptr]=1000;
     LLink[ptr]=ptr1;
}
```

```
void DL_Delete_Front()
{
     int ptr1,ptr;
     ptr=RLink[Header];


    if(ptr==0)
    {
        printf("\nList is empty.No deletion is made!!!!!!\n");
    }
    else
    {
         ptr1=RLink[ptr];
         RLink[Header]=ptr1;
         if(ptr1!=0)
         {
                LLink[ptr1]=Header;
         }
         ReturnNode(ptr);
     }
     DL_Traverse();
}

void DL_Delete_End()
{
     int ptr=Header,ptr1;
     while(RLink[ptr]!=1000)
     {
         ptr=RLink[ptr];
     }
     if(ptr==Header)
     {
        printf("List is empty: No deletion is made");
        return;
     }
     else
     {
        ptr1=LLink[ptr];
        RLink[ptr1]=1000;
        ReturnNode(ptr);
    }
    DL_Traverse();
}

void DL_Delete_Any()
{
    int ptr=RLink[Header];
    int ptr1,ptr2,key;
    if(ptr==1000)
    {
        printf("List is empty: No deletion is made");
        return;
    }
    printf("\n\nWhich value you want to delete? ::: ");
    scanf("%d",&key);
    while((Data[ptr]!=key) && (RLink[ptr]!=1000))
    {
        ptr=RLink[ptr];
    }
    if(Data[ptr]==key)
    {
        ptr1=LLink[ptr];
        ptr2=RLink[ptr];
        RLink[ptr1]=ptr2;
        if(ptr2!=1000)
        {
           LLink[ptr2]=ptr1;
        }
        ReturnNode(ptr);
        DL_Traverse();
    }
    else
        printf("The node does not exits in the given list");
}

int main()
{
     int choice;
     DL_Traverse();
     while(1)
     {
        printf("\n\nEnter 1 for Print Double Linked List");
        printf("\nEnter 2 for Print Avail List");
        printf("\nEnter 3 for delete element from Front of DL");
        printf("\nEnter 4 for delete element from End of DL");
        printf("\nEnter 5 for delete element from Any position of DL");
        printf("\nEnter 6 for Exit ::: ");
        scanf("%d",&choice);
        if(choice==6)
          break;
        switch(choice)
        {
             case 1: DL_Traverse(); break;
             case 2: Avail_Traverse(); break;
             case 3: DL_Delete_Front(); break;
             case 4: DL_Delete_End(); break;
             case 5: DL_Delete_Any(); break;
             default: printf("\nWrong Choice !!!");
         }
     }
 }
```

Program :

```
#include<stdio.h>
int Stack[11],top=0,Size=10;
void Print_Stack()
{
    if(top==0)
    {
        printf("\n\nStack is empty !!!!\n");
    }
    else
    {
        printf("\n The Stack is as follow ::: \n");
        for(int i=top;i>=1;i--)
        {
              printf("%d\n",Stack[i]);
        }
     }
}

void Push()
{
    int item;
    printf("\nEnter value to push at stack ::: ");
    scanf("%d",&item);
    if(top>=Size)
         printf("\n\nStack is full!!!!\n");
    else
    {
         top=top+1;
         Stack[top]=item;
    }
    Print_Stack();
}

void Pop()
{
    int item;
    if(top<1)
         printf("\n\nStack is empty!!!!\n");
    else
    {
         item=Stack[top];
         top=top-1;
    }
    printf("\n%d is pop from stack. ",item);
    Print_Stack();
}
```

```
void Status()
{
    float free;
    if(top<1)
         printf("\n\nStack is empty!!!!\n");

    else if(top>=Size)
         printf("\n\nStack is full!!!!\n");
    else
    {
         printf("\nThe element at Top is %d",Stack[top]);
         free=((Size-top)%Size)*10;
         printf("\nPercentage of FREE Stack is %.0f percent.",free);
    }
}

int main()
{
   int i,choice;
   for(i=0;i<10;i++)
       Stack[i]=0;
   while(1)
   {
       printf("\n\nEnter 1 for Print Stack");
       printf("\nEnter 2 for PUSH");
       printf("\nEnter 3 for POP");
       printf("\nEnter 4 for STATUS");
       printf("\nEnter 5 for Exit ::: ");
       scanf("%d",&choice);
       if(choice==5)
          break;
       switch(choice)
       {
            case 1: Print_Stack(); break;
            case 2: Push(); break;
            case 3: Pop(); break;
            case 4: Status(); break;
            default: printf("\nWrong Choice !!!");
       }
    }
 }
```

Program :

```
#include<stdio.h>
#include<string.h>
charSymbol_Priority[8][3]={'+','2','1','','2','1','*','4','3','/','4','3','^','5','6','O','8','7','(','0','9',')','-','0'};
char Stack[7]={' ',' ',' ',' ',' ',' ',' '};
int top=0;
 void Push(char item)
 {
    top=top+1;
    Stack[top]=item;
 }

 char Pop()
 {
    int item=Stack[top];
    top=top-1;
    return item;
 }

 int ISP(char symbol)
 {
     int i;
     for(i=0;i<=7;i++)
         if(Symbol_Priority[i][0]==symbol)
            break;
     return (int)Symbol_Priority[i][1];
  }

  int ICP(char symbol)
  {
      int i;
      for(i=0;i<=7;i++)
          if(Symbol_Priority[i][0]==symbol)
             break;
      return (int)Symbol_Priority[i][2];
  }

 char Check_Operand(char item)
 {
     if((item >= 65 && item <= 90) || (item >= 97 && item <= 122))
         return item;
     else
         return ' ';
 }

 void InFix_To_PostFix(char symbol[20])
  {
     char item,x;
     int Symbol_count=0;
     printf("\n\nPostFix Expression ::: ");
    Push('(');
    while(top>0)
    {
          item=symbol[Symbol_count];
          x=Pop();
          if(item==Check_Operand(item))
          {
              Push(x);
              printf(" %c ",item);
          }
          else if(item==')')
          {
              while(x!='(')
              {
                   printf(" %c ",x);
                   x=Pop();
              }
          }
          else if(ISP(x)>= ICP(item))
          {
              while(ISP(x)>= ICP(item))
              {
                   printf(" %c ",x);
                   x=Pop();
              }
              Push(x);
              Push(item);
          }
          else if(ISP(x) < ICP(item))
          {
              Push(x);
              Push(item);
          }
          else
               printf("\n\nInvalid Expression.!!!!\n\n");

          Symbol_count=Symbol_count+1;
     }
}

int main()
{
    char exp[20];
    printf("Enter InFix Expression :::: ");
    scanf("%s",&exp);
    InFix_To_PostFix(exp);
}
```

Program :

```
 #include<stdio.h>
 char que[10]={' ',' ',' ','A','B','C','D',' ',' ',' '};
 int rear=6,frnt=3,que_size=9;
 void Print_Queue()
 {
      if(frnt==0 && rear==0)
             printf("\n\nQueue is empty !!!!\n");
      else
      {
             printf("\n The Queue is as follow ::: ");
             for(int i=frnt;i<=rear;i++)
                   printf("  %c  ",que[i]);
       }
 }

 void EnQueue()
 {
      if(rear==que_size)
           printf("\n\nQueue is full!!!!\n");
      else
      {
             char item;
             printf("\nEnter value to insert at queue ::: ");
             item=getchar();
             item=getchar();
             if(rear==0 && frnt==0)
                frnt=1;
             rear=rear+1;
             que[rear]=item;
       }
       Print_Queue();
  }

 void DeQueue()
 {
      int item;
      if(frnt==0)
            printf("\n\nQueue is empty!!!!\n");
      else
      {
            item=que[frnt];
            if(frnt==rear)
            {
                  frnt=0;
                  rear=0;
            }
```

```
          else
              frnt=frnt+1;
      }
      Print_Queue();
}

int main()
{
    int choice;
    while(1)
    {
        printf("\n\nEnter 1 for Print Queue");
        printf("\nEnter 2 for EnQueue");
        printf("\nEnter 3 for DeQueue");
        printf("\nEnter 4 for Exit ::: ");
        scanf("%d",&choice);
        if(choice==4)
           break;
        switch(choice)
        {
              case 1: Print_Queue(); break;
              case 2: EnQueue(); break;
              case 3: DeQueue(); break;
              default: printf("\nWrong Choice !!!");
        }
    }
}
```

Program :

```
#include<stdio.h>
char tree[20]={' ','+','-','*','A','B','C','/',' ',' ',' ',' ',' ',' ','D','E',' ',' ',' ',' '};
void Print_Tree()
{
    printf("\n\nThe tree is as follow ::: \n");
    for (int i=1;i<=19;i++)
    {
        if(i==1||i==3||i==7||i==15)
            printf(" %c \n",tree[i]);
        else
            printf(" %c ",tree[i]);
    }
    printf("\n\n");
}

void Root()
{
    printf("\nThe Root of the Tree is ::: %c ",tree[1]);
}

void Parent()
{
    char item;
    int i;
    printf("Enter the node value for which you want to search :::");
    scanf("%c",&item);
    scanf("%c",&item);
    if(tree[1]==item)
       printf("\n %c is Root Node. It has not any Parent....",item);
    else
    {
       for(i=1;i<=19;i++)
       if(tree[i]==item)
          break;
       printf("\nNode ::: %c,  Parent ::: %c",tree[i],tree[i/2]);
    }
 }

void Child()
{
    char item;
    int i;
    printf("Enter the node value for which you want to find child :::");
    scanf("%c",&item);
    scanf("%c",&item);
    for(i=1;i<=19;i++)
      if(tree[i]==item)
         break;
```

```
  if(tree[i*2]!=' ' && tree[(i*2)+1]!=' ')
    printf(“\nParent Node :: %c\nLeft child :: %c\nRight child :: %c\n”,tree[i],tree[i*2],tree[i*2+1]);
  else if(tree[i*2]!=' ' && tree[(i*2)+1]==' ')
       printf("\nParent Node ::: %c\nLeft Child ::: %c\nNo Right Child .......\n",tree[i],tree[i*2]);
  else if(tree[i*2]==' ' && tree[(i*2)+1]!=' ')
       printf("\nParent Node ::: %c\nNo Left Child .......\nRight Child ::: %c\n",tree[i],tree[(i*2)+1]);
  else
       printf("\n%c is Leaf Node. No Child Exists.....\n",tree[i]);
}

void Degree()
{
     char item;
     int i;
     printf("Enter the node value for which you want to find Degree :::");
     scanf("%c",&item);
     scanf("%c",&item);
     for(i=1;i<=19;i++)
         if(tree[i]==item)
            break;
     if(tree[i*2]!=' ' && tree[(i*2)+1]!=' ')
        printf("\nThe degree of the Node is 2");
     else if(tree[i*2]!=' ' && tree[(i*2)+1]==' ')
         printf("\nThe degree of the Node is 1");
     else if(tree[i*2]==' ' && tree[(i*2)+1]!=' ')
         printf("\nThe degree of the Node is 1");
     else
         printf("\nThe degree of the Node is 0");
 }

void Leaf_Node()
{
     int i;
     printf("\nLeaf Nodes of the tree are :::: ");
     for(i=1;i<=19;i++)
         if((tree[i]!=' ' && tree[i*2]==' ' && tree[(i*2)+1]==' ') || (tree[i]!=' ' && (i*2)>19))
             printf(" %c ",tree[i]);
}

void Siblings()
{
     char item;
     int i;
     printf("Enter the node value for which you want to find Siblings :::");
     scanf("%c",&item);
     scanf("%c",&item);
     for(i=1;i<=19;i++)
         if(tree[i]==item)
            break;
     if(i==1)
        printf("\n%c is Root Node. It has not any Siblings...\n",tree[i]);
    else
    {
       if((i%2)==0 && tree[i+1]!=' ')
           printf("\nNode ::: %c\nSiblings ::: %c\n",tree[i],tree[i+1]);
       else if((i%2)==0 && tree[i+1]==' ')
           printf("\nNode ::: %c\nNo Siblings ....\n",tree[i],tree[i+1]);
       else if((i%2)==1 && tree[i-1]!=' ')
           printf("\nNode ::: %c\nSiblings ::: %c\n",tree[i],tree[i-1]);
       else if((i%2)==1 && tree[i-1]==' ')
           printf("\nNode ::: %c\nNo Siblings ....\n",tree[i],tree[i-1]);
    }
}

void Level()
{
    char item;
    int i,lev=0;
    printf("\nEnter node find out level: ");
    scanf("%c",&item);
    scanf("%c",&item);
    for(i=1;i<=16;i++)
    {
       if(tree[i]==item)
          break;
    }
    while(i/2)
    {
       lev++;
       i=i/2;
    }
    printf("Level is %d ",lev);
}

int main()
{
   int choice;
   while(1)
   {
       printf("\n\nEnter 1 for Print Tree");
       printf("\nEnter 2 for Find Root");
       printf("\nEnter 3 for Find Parent");
       printf("\nEnter 4 for Find Child");
       printf("\nEnter 5 for Find Degree");
       printf("\nEnter 6 for Find Leaf Nodes");
       printf("\nEnter 7 for Find Siblings");
       printf("\nEnter 8 for Find Level of a Node");
       printf("\nEnter 9 for Exit ::: ");
       scanf("%d",&choice);
       if(choice==9)
          break;
```

```
      switch(choice)
      {
           case 1: Print_Tree(); break;
           case 2: Root(); break;
           case 3: Parent(); break;
           case 4: Child(); break;
           case 5: Degree(); break;
           case 6: Leaf_Node(); break;
           case 7: Siblings(); break;
           case 8: Level(); break;
           default: printf("\nWrong Choice !!!");
       }
    }
}
```