BUBBLE SORTING

File Name	bubblesort.c	Compiler	online gdb compiler
Experiment No	1 a	Data Structure	Array
Date	11.9.2020		
Aim	To implement bubble sorting using C language		

Algorithm

Input: Unsorted array

Output: Sorted array

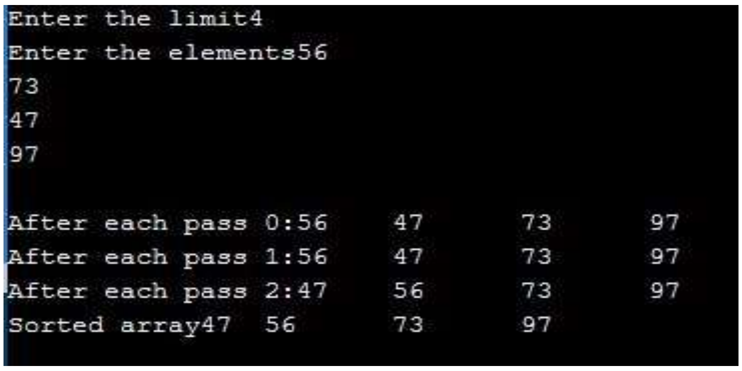
Steps:

```
1. Begin BubbleSort(list)
2. For all elements of list
       If list[i]>list[i+1]
       Swap(list[i],list[i+1]
       End if
       End for
3. Return list
4. End BubbleSort
```

Program

```
#include <stdio.h>
void main()
{
   int i,j,k,t=0,n,a[50];
   printf("Enter the limit");
   scanf("%d",&n);
   printf("Enter the elements");
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   for(i=0;i<n-1;i++)
   {
      for(j=0;j<n-i-1;j++)
      {
         if(a[i]>a[j+1])
         {
            t=a[j];
            a[j]=a[j+1];
            a[j+1]=t;
         }
      }
      printf("\nAfter each pass %d:",i);
      for(k=0;k<n;k++)
      {
         printf("%d\t",a[k]);
      }
   }
   printf("\nSorted array");
   for(i=0;i<n;i++)
   {
      printf("%d\t",a[i]);
   }
}
```

Output



```
Enter the limit4
Enter the elements56
73
47
97

After each pass 0:56    47      73      97
After each pass 1:56    47      73      97
After each pass 2:47    56      73      97
Sorted array47  56      73      97
```

Result: The program was executed and output obtained successfully and was verified

SELECTION SORTING

File Name	selectionsort.c	Compiler	online gdb compiler
Experiment No	1 b	Data Structure	Array
Date	11.9.2020		
Aim	To implement selection sorting using C language		

Algorithm

Input: Unsorted array, X with size, SIZE

Output: Sorted array

Steps:SELECTION(X,SIZE)

```
1. Repeat steps 2 and 3 for k=0,….SIZE-1
2. Call findMinLoc(X,k,size)
3. Exchange X[k] and X[m]
4. Exit
```

findMinLoc(X,k,size)

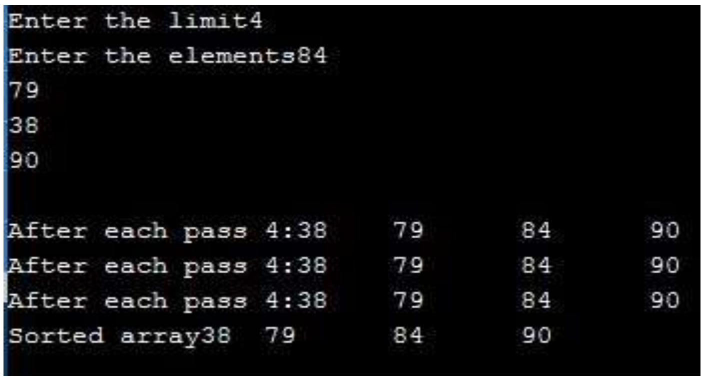
```
1. Set pos=k
2. Repeat for j=k+1,k+2……..size
      If(X[j]<X[pos])    pos=j
3. Return pos
```

Program

```
#include <stdio.h>
void main()
{
  int i,j,k,t,n,a[50],m,p;
  printf("Enter the limit");
  scanf("%d",&n);
  printf("Enter the elements");
  for(i=0;i<n;i++)
  {
    scanf("%d",&a[i]);
  }
  for(k=0;k<n-1;k++)
  {
    p=k;
    for(j=k+1;j<n;j++)
```

```
        {
           if(a[j]<a[p])
           {
              p=j;
           }
        }
        m=p;
        t=a[k];
        a[k]=a[m];
        a[m]=t;
        printf("\nAfter each pass %d:",i);
        for(j=0;j<n;j++)
        {
           printf("%d\t",a[j]);
        }
     }
     printf("\nSorted array");
     for(i=0;i<n;i++)
     {
        printf("%d\t",a[i]);
     }
}
```

Output



```
Enter the limit4
Enter the elements84
79
38
90

After each pass 4:38    79      84      90
After each pass 4:38    79      84      90
After each pass 4:38    79      84      90
Sorted array38  79      84      90
```

Result: The program was executed and output obtained successfully and was verified

INSERTION SORTING

File Name	insertionsort.c	Compiler	online gdb compiler
Experiment No	1c	Data Structure	Array
Date	11.9.2020		
Aim	To implement bibble sorting using C language		

Algorithm

Input: Unsorted array, X with size, SIZE

Output: Sorted array

Steps:SELECTION(X,SIZE)

```
1.  Repeat steps 2 and 3 for k=0,….SIZE-1
2.  Call findMinLoc(X,k,size)
3.  Exchange X[k] and X[m]
4.  Exit
```

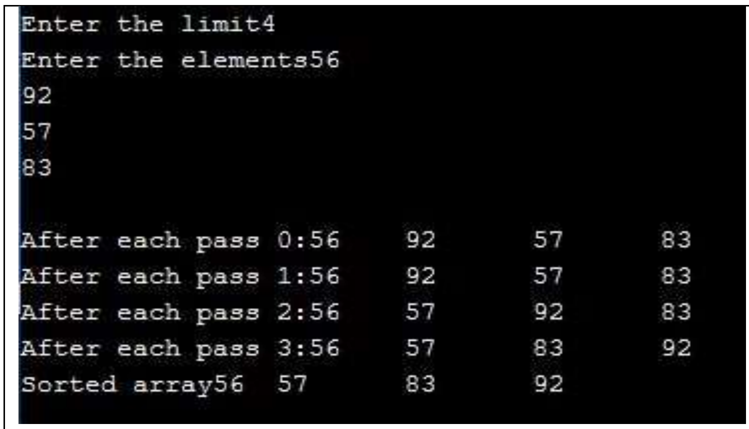
findMinLoc(X,k,size)

```
5.  Set pos=k
6.  Repeat for j=k+1,k+2……..size
        If(X[j]<X[pos])    pos=j
7.  Return pos
8.  Repeat steps 2 to 4 for i=0,….SIZE-1
9.  Set item=X[i] and j=i-1
10. Repeat while ((j>=0)&&(X[j]>item)
        X[j+1]=X[j];
        j=j-1
11. X[j+1]=item
```

Program

```
#include <stdio.h>
void main()
{
   int i,j,k,t,n,a[50];
   printf("Enter the limit");
   scanf("%d",&n);
   printf("Enter the elements");
   for(i=0;i<n;i++)
   {
        scanf("%d",&a[i]);
    }
    for(i=0;i<n;i++)
    {
        t=a[i];
        j=i-1;
        while((j>=0)&&(a[j]>t))
        {
            a[j+1]=a[j];
            j--;
        }
        a[j+1]=t;
        printf("\nAfter each pass %d:",i);
        for(k=0;k<n;k++)
        {
            printf("%d\t",a[k]);
        }
    }
    printf("\nSorted array");
    for(i=0;i<n;i++)
    {
        printf("%d\t",a[i]);
    }
}
```

Output



```
Enter the limit4
Enter the elements56
92
57
83

After each pass 0:56    92      57      83
After each pass 1:56    92      57      83
After each pass 2:56    57      92      83
After each pass 3:56    57      83      92
Sorted array56  57      83      92
```

Result: The program was executed and output obtained successfully and was verified

QUICK SORTING

File Name	quicksort.c	Compiler	online gdb compiler
Experiment No	2 a	Data Structure	Array
Date	19.9.2020		
Aim	To implement quick sorting using C language		

Algorithm

Input: Unsorted array

Output: Sorted array

Data Structure: Array

Quicksort(A, left, right)

Steps:

```
1.If (left<right)
      1.1) q=partition(A, left,right)
      1.2 ) quicksort(A,left,q-1)
      1.3 ) quicksort(A,q+1,right) partition(A, left, right)
1.pivot =  A[left]
2.Set i = left,j = right
3.Repeat steps 4,5,6 while (i<=j)
4.while(A[i]<=pivot && i<=right)
       i=i+1
5.while( a[j] > pivot )
      j=j-1
6.if(i<j)
    | Swap A[i] and A[j]
7.Swap A[left] and A[j]
8.Return j
```

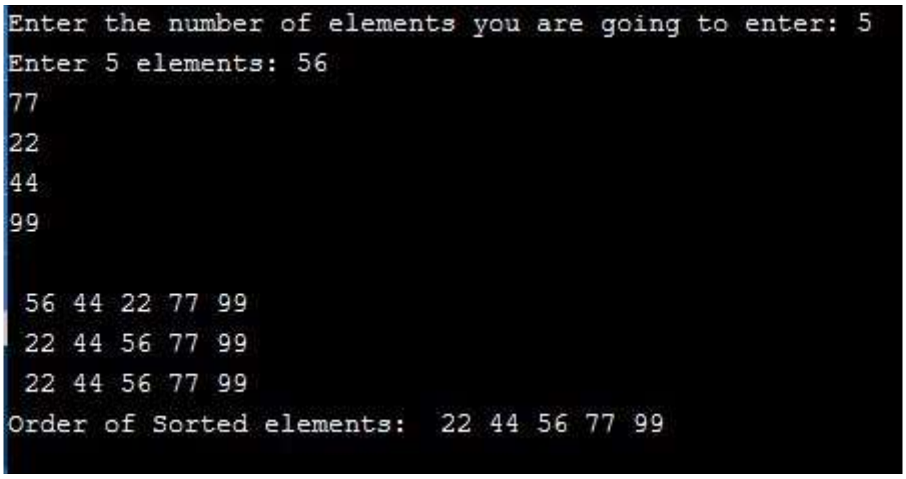
Program

```
#include<stdio.h>
void quicksort(int a[30],int st,int end, int l);
void main()
{
  int i, n, a[30],len;
  printf("Enter the number of elements you are going to enter: "); scanf("%d",&n);
  len = n;
  printf("Enter %d elements: ", n); for(i=0;i<n;i++)
  scanf("%d",&a[i]);
  quicksort(a,0,n-1,len);
```

```
    printf("\nOrder of Sorted elements: ");
    for(i=0;i<n;i++)
    printf(" %d",a[i]);
}
    void quicksort(int a[30],int st,int end,int l)
{
    int i,j,pivot,temp;
    if(st<end)
    {
        pivot = st; i = st;
        j = end;
        while(i<j)
        {
            while(a[i]<=a[pivot]&&i<end)
            i++;
            while(a[j]>a[pivot])
            j--;
            if(i<j)
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
        printf("\n");
        for(i=0;i<l;i++)
        printf(" %d",a[i]);
        temp=a[pivot];
        a[pivot]=a[j];
        a[j]=temp;
        quicksort(a,st,j-1,l);
        quicksort(a,j+1,end,l);
    }
```

}

Output



```
Enter the number of elements you are going to enter: 5
Enter 5 elements: 56
77
22
44
99

 56 44 22 77 99
 22 44 56 77 99
 22 44 56 77 99
Order of Sorted elements:  22 44 56 77 99
```

Result: The program was executed and output obtained successfully and was verified

MERGE SORTING

File Name	mergesort.c	Compiler	online gdb compiler
Experiment No	2 b	Data Structure	Array
Date	19.9.2020		
Aim	To implement merge sorting using C language		

Algorithm

Input: Unsorted array, X with size, SIZE

Output: Sorted array

Data Structure: Array

Mergesort(A, left, right)

Steps:

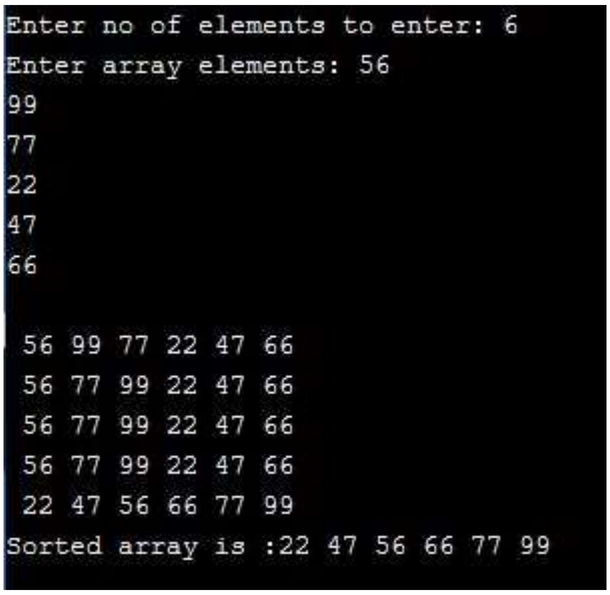
```
1.If (left<right) then
      1.1 mid=(left + right)/2
      1.2 Mergesort(A, left, mid)
       1.3 Mergesort(A,mid+1,right)
       1.4 Merge(A,left,mid,mid+1,right)
Merge(A, left,mid,mid+1,right)
1.Set i=left, j=mid+1
2.while(i<=mid && j<=right)
     2.1 if(A[i] < A[j])
               temp[k]=A[i],k=k+1,i=i+1
          else
                temp[k]=A[j],k=k+1,j=j+1
           endif
       endwhile
3. while(i<=mid)
      temp[k]=A[i],k=k+1,i=i+1
      endwhile
4. while(j<=right)
     temp[k]=A[j],k=k+1,j=j+1
     endwhile
5. Set i=left,j=0
6. while(i<=right)
     4.   A[i]=temp[j] 6.
     5.   2 i=i+1,j=j+1
7.endwhile
```

Program

```
#include<stdio.h>

void mergesort(int a[],int i,int j,int l);

void merge(int a[],int i1,int j1,int i2,int j2);
```

```
int main()
{
   int a[30],n,i;
   printf("Enter no of elements to enter: "); scanf("%d",&n);
   printf("Enter array elements: ");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   mergesort(a,0,n-1,n);
   printf("\nSorted array is :"); for(i=0;i<n;i++)
   printf("%d ",a[i]);
   return 0;
}
void mergesort(int a[],int i,int j,int l)
{
   int mid;
   if(i<j)
   {
      mid=(i+j)/2;
      mergesort(a,i,mid,l);
      mergesort(a,mid+1,j,l);
      merge(a,i,mid,mid+1,j);
      printf("\n");
      for(i=0;i<l;i++)
      printf(" %d",a[i]);
   }
}
void merge(int a[],int i1,int j1,int i2,int j2)
{
   int temp[50];
   int i,j,k;
   i=i1;
   j=i2;
   k=0;
   while(i<=j1 && j<=j2)
   {
      if(a[i]<a[j])
      temp[k++]=a[i++];
      else
      temp[k++]=a[j++];
   }
      while(i<=j1)
      temp[k++]=a[i++];
      while(j<=j2)
      temp[k++]=a[j++];
      for(i=i1,j=0;i<=j2;i++,j++)
      a[i]=temp[j];
}
```

Output



```
Enter no of elements to enter: 6
Enter array elements: 56
99
77
22
47
66

 56 99 77 22 47 66
 56 77 99 22 47 66
 56 77 99 22 47 66
 56 77 99 22 47 66
 22 47 56 66 77 99
Sorted array is :22 47 56 66 77 99
```

Result: The program was executed and output obtained successfully and was verified

LINEAR SEARCHING

File Name	linearsearch.c	Compiler	online gdb compiler
Experiment No	3 a	Data Structure	Array
Date	25.9.2020		
Aim	To implement linear searching using C language		

Algorithm

Input: Unsorted array

Output: Sorted array

Steps:

Linear_search(X,N,key)

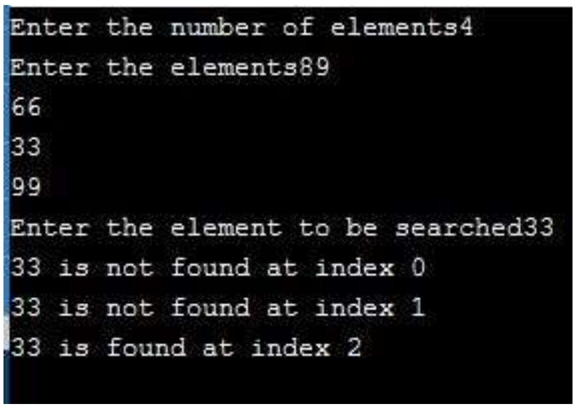
```
5. Set LOC=0
6. Repeat steps while LOC<N
    1. If(X[10]==key)
        Return LOC
        Else
        Return -1
        End while
```

Program

```
#include <stdio.h>
int linear_search(int a[],int size,int key)
{
  int i;
  for(i=0;i<size;i++)
  if(a[i]==key)
  {
    return i;
  }
  else
  {
    printf("%d is not found at index %d",key,i);
    printf("\n");
  }
  return -1;
}
```

```
int main(void)
{
   int x[50],i,size,key;
   printf("Enter the number of elements");
   scanf("%d",&size);
   printf("Enter the elements");
   for(i=0;i<size;i++)
   {
      scanf("%d",&x[i]);
   }
   printf("Enter the element to be searched");
   scanf("%d",&key);
   int result=linear_search(x,size,key);
   if(result==-1)
   {
      printf("%d is not found in array",key);
   }
   else
   {
      printf("%d is found at index %d",key,result);
   }
   return 0;
}
```

Output



```
Enter the number of elements4
Enter the elements89
66
33
99
Enter the element to be searched33
33 is not found at index 0
33 is not found at index 1
33 is found at index 2
```

Result: The program was executed and output obtained successfully and was verified

BINARY SEARCHING

File Name	binarysearching.c	Compiler	online gdb compiler
Experiment No	3 b	Data Structure	Array
Date	25.9.2020		
Aim	To implement binary searching using C language		

Algorithm

Input: Unsorted array

Output: Sorted array

Steps:

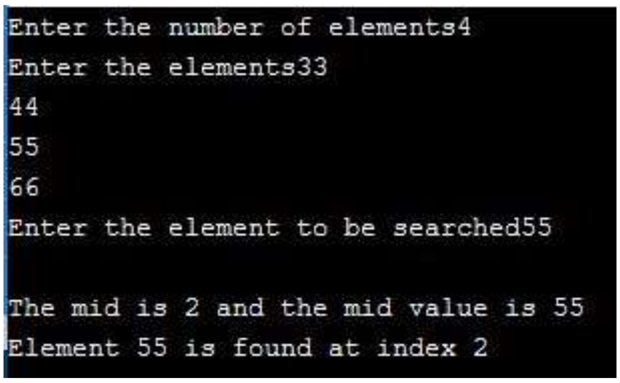
Binary_search(X,N,key)

```
1. Set left=0,rifht=N-1
2. Repeat steps 3 and 4 while left<=right
3. Mid=(left+right)/2
4. If A[mid]<key then
       Set left=mid+1
       Else if A[mid]>key
       Set right=mid-1
       Else
       Return mid//found at location mid
       End while
5. Return 0
```

Program

```
#include <stdio.h>
int binary_search(int a[],int size,int key)
{
   int i,mid,L=0,R=size;
   while(L<=R)
   {
      mid=(L+R)/2;
      printf("\nThe mid is %d and the mid value is %d",mid,a[mid]);
      if(a[mid]<key)
      {
         L=mid+1;
      }
      else if(a[mid]>key)
      {
            R=mid-1;
        }
        else
        {
            return mid;
        }
    }
    printf("\n");
    return -1;
}
int main(void)
{
    int a[50],i,size,val;
    printf("Enter the number of elements");
    scanf("%d",&size);
    printf("Enter the elements");
    for(i=0;i<size;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("Enter the element to be searched");
    scanf("%d",&val);
    int result=binary_search(a,size,val);
    if(result==-1)
    {
        printf("\nElement is not found in array");
    }
    else
    {
        printf("\nElement %d is found at index %d",val,result);
    }
    return 0;
}
```

Output



```
Enter the number of elements4
Enter the elements33
44
55
66
Enter the element to be searched55

The mid is 2 and the mid value is 55
Element 55 is found at index 2
```

Result: The program was executed and output obtained successfully and was verified

POLYNOMIAL ADDITION

File Name	polynomialadd.c	Compiler	online gdb compiler
Experiment No	4 a	Data Structure	Array
Date	9.10.2020		
Aim	To implement polynomial addition using C language		

Algorithm

INPUT: 2input polynomials namely X1,X2 and resultant polynomial X3.

M is the number of terms in X1 and N is the number of terms in X2.

OUTPUT:Resultant polynomial X3, which contains the sum of X1 and X2.

DATA STRUCTURE:Array

Int Polynomial addition(struct polyX1[], struct poly X2[],struct poly X3[],int M, int N)

```
1.  Set i=0,j=0,k=0.
2.  Repeat step 3 while(i<M&&j<N)
3.  If (X1[i].expo==X2[j].expo)
    3.1 X3[k].coeff=X1[i].coeff+X2[j].coeff
    3.2 X3[k].expo=X1[i].expo
    3.3 i++,j++,k++
          else
    3.1 X3[k].coeff=X1[i].coeff
    3.2 X3[k].expo=X1[i].expo
    3.3 i++,k++
          Else
    3.1 X3[k].coeff=X2[j].coeff
    3.2 X3[k].expo=X2[j].expo
    3.3 j++,k++
4. while(i<M)        /* for rest over terms of polynomial 1 */
    4.1 X3[k].coeff=X1[i].coeff
    4.2 X3[k].expo=X1[i].expo
    4.3 i++, k++
5. while(j<N) /* for rest over terms of polynomial 2 */
    5.1 p3[k].coeff=p2[j].coeff
    5.2 p3[k].expo=p2[j].expo
```

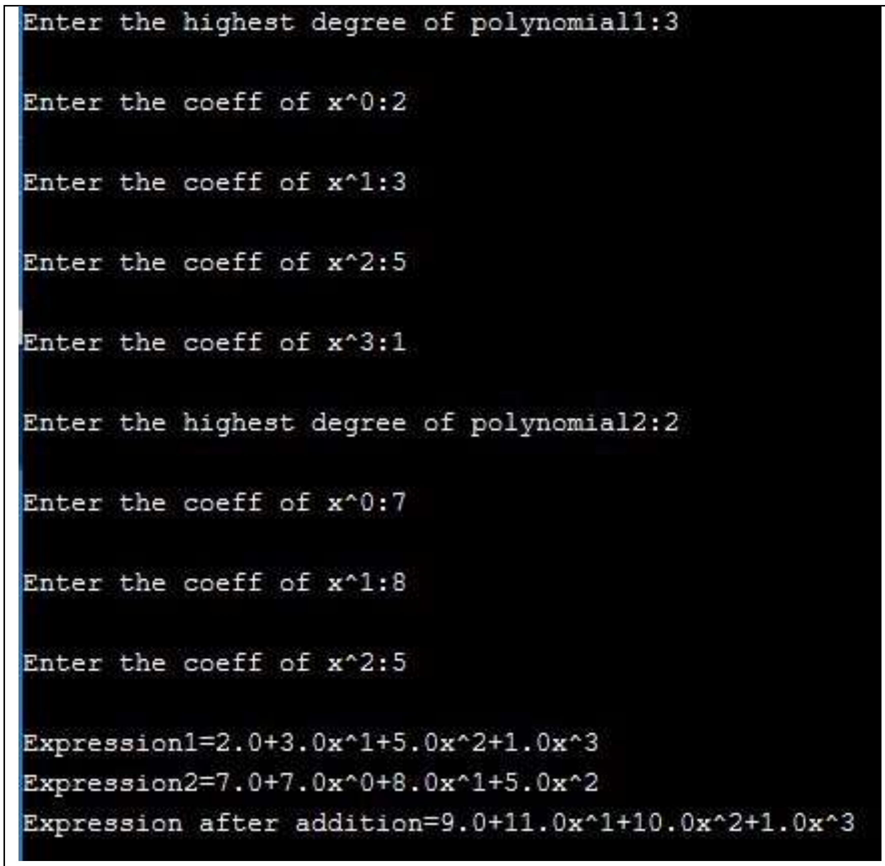
```
        5.3 j++,k++
    6.return (k)       /* k is number of terms in resultant polynomial*/
```

Program

```
#include <stdio.h>
#include<math.h>
struct poly
{
   float coeff;
   int exp;
};
struct poly a[50],b[50],c[50],d[50];
int main()
{
   int i;
   int deg1,deg2;
   int k=0,l=0,m=0;
   printf("Enter the highest degree of polynomial1:");
   scanf("%d",&deg1);
   for(i=0;i<=deg1;i++)
   {
      printf("\nEnter the coeff of x^%d:",i);
      scanf("%f",&a[i].coeff);
      a[k++].exp=i;
   }
   printf("\nEnter the highest degree of polynomial2:");
   scanf("%d",&deg2);
   for(i=0;i<=deg2;i++)
   {
      printf("\nEnter the coeff of x^%d:",i);
      scanf("%f",&b[i].coeff);
      b[l++].exp=i;
   }
   printf("\nExpression1=%.1f",a[0].coeff);
   for(i=1;i<=deg1;i++)
   {
      printf("+%.1fx^%d",a[i].coeff,a[i].exp);
   }
   printf("\nExpression2=%.1f",b[0].coeff);
   for(i=0;i<=deg2;i++)
   {
      printf("+%.1fx^%d",b[i].coeff,b[i].exp);
   }
   if(deg1>deg2)
   {
      for(i=0;i<=deg2;i++)
      {
         c[m].coeff=a[i].coeff+b[i].coeff;
         c[m].exp=a[i].exp;
         m++;
      }
      for(i=deg2+1;i<=deg1;i++)
      {
         c[m].coeff=a[i].coeff;
         c[m].exp=a[i].exp;
         m++;
      }
   }
   else
   {
      for(i=0;i<=deg1;i++)
```

```
        {
          c[m].coeff=a[i].coeff+b[i].coeff;
          c[m].exp=a[i].exp;
          m++;
        }
        for(i=deg1+1;i<=deg2;i++)
        {
          c[m].coeff=b[i].coeff;
          c[m].exp=b[i].exp;
          m++;
        }
    }
    printf("\nExpression after addition=%.1f",c[0].coeff);
    for(i=1;i<m;i++)
    {
        printf("+%.1fx^%d",c[i].coeff,c[i].exp);
    }
    return 0;
}
```

Output



```
Enter the highest degree of polynomial1:3

Enter the coeff of x^0:2

Enter the coeff of x^1:3

Enter the coeff of x^2:5

Enter the coeff of x^3:1

Enter the highest degree of polynomial2:2

Enter the coeff of x^0:7

Enter the coeff of x^1:8

Enter the coeff of x^2:5

Expression1=2.0+3.0x^1+5.0x^2+1.0x^3
Expression2=7.0+7.0x^0+8.0x^1+5.0x^2
Expression after addition=9.0+11.0x^1+10.0x^2+1.0x^3
```

Result: The program was executed and output obtained successfully and was verified

TRANSPOSE OF A SPARSE MATRIX

File Name	sparsematrix.c	Compiler	online gdb compiler
Experiment No	4 b	Data Structure	Array
Date	9.10.2020		
Aim	To implement sparse matrix using C language		

Algorithm

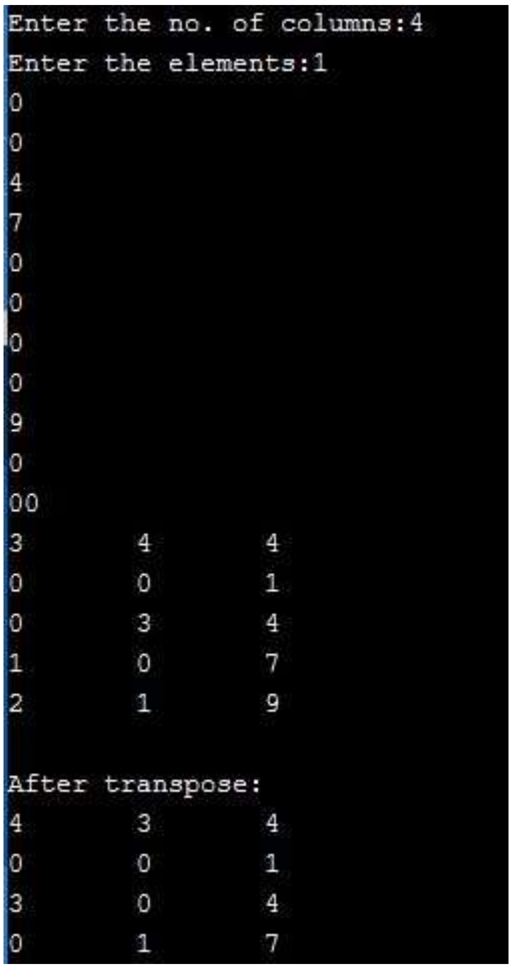
```
Input:  A 2D array named a.
Ouput: Resultant 2D array b.
Datastructure: Array
 SparseMatrix SparseMatrix::Transpose()
Construct a SparseMatrix, b(cols, rows, terms);
If (terms > 0) {
Let rowSize be an integer array of size cols.
Let rowStart be an integer array of size cols.
Initialize each element in rowSize to 0.
For (i=0; i
 rowSize [smArray[i].col]++;
rowStart [0] = 0;
For (i = 1; i < cols; i++)
rowStart [i] = rowSize [i-1] + rowStart [i-1];
For (i=0; i < terms; i++) {
j = rowStart [ smArray [i].col ];
Copy smArray[ i ] to smArray[ j ];
Interchange row and col of smArray[ j ];
rowStart [ smArray [i].col ]++;
Return b;
```

Program

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

```
struct sparse
{
   int row;
   int column;
}sp;

int main()
{
   int i,j,n,a[30][30],row,column,b[30][30],k=1,t[30][30];
   printf("Enter the no. of rows:");
   scanf("%d",&sp.row);
   printf("Enter the no. of columns:");
   scanf("%d",&sp.column);
   printf("Enter the elements:");
   for(i=0;i<sp.row;i++)
   {
      for(j=0;j<sp.column;j++)
      {
         scanf("%d",&a[i][j]);
      }
   }
   b[0][0]=sp.row;
   b[0][1]=sp.column;
   for (i = 0; i <sp.row; i++)
   {
      for (j = 0; j <sp.column; j++)
      {
         if (a[i][j]!= 0)
         {
            b[k][0] = i;
            b[k][1] = j;
            b[k][2] = a[i][j];
            k++;
         }
      }
    b[0][2] = k-1;
   }
      for(i=0;i<k;i++)
      {
      printf("%d\t%d\t%d\n", b[i][0], b[i][1], b[i][2]);
      }
      printf("\n");
     printf("After transpose:\n");
     k=1;
         b[0][0]=sp.column;
         b[0][1]=sp.row;

         for(i=0;i<sp.row;i++)
         {
            for(j=0;j<sp.column;j++)
            {
               if (a[i][j]!= 0)
         {
            b[k][0] = j;
            b[k][1] = i;
            b[k][2] = a[i][j];
            k++;
         }
            }
                   b[0][2]=k-1;
         }
         for(i=0;i<k;i++)
      {
      printf("%d\t%d\t%d\n", b[i][0], b[i][1], b[i][2]);
      }
      printf("\n");
}
```

Output



```
Enter the no. of columns:4
Enter the elements:1
0
0
4
7
0
0
0
0
9
0
00
3       4       4
0       0       1
0       3       4
1       0       7
2       1       9

After transpose:
4       3       4
0       0       1
3       0       4
0       1       7
```

Result: The program was executed and output obtained successfully and was verified

STACK AND QUEUE

File Name	stackndqueue.c	Compiler	online gdb compiler
Experiment No	5 a	Data Structure	Array
Date	23.10.2020		
Aim	To implement stack and queue using C language		

Algorithm

Input: Array arr[]

Output: arr[] after insertion and deletion using stack and queue implementation

Data Structure: Array

void stackop()

1. Initialize top with -1

2. Ask the user for the operation to perform. Repeat this step until user's input is valid. Label this part as 'task'.

3. For push operation :

a. Increment value of top by 1;

b. If top>length of array, print "Stack is full", goto task(step 2)

c. Otherwise, input a number to arr[top] and ask if user want to enter another number

d. If user doesn't, goto task (step 2)

e. Otherwise repeat steps a to d;

4. For Pop operation :

a. Decrement value of top by 1;

b. If top<0, print "Stack is empty", goto task(step 2)

c. Otherwise, print "Element popped", and ask if user want to pop another number

d. If user doesn't, goto task (step 2)

e. Otherwise repeat steps a to d;

5. For Peek operation :

a. Print arr[top]

b. goto task(step 2)

6. For printing :

a. For i= top till i>=0, print arr[i]

```
void stackop()
1. Initialize front and rear with -1
2. Ask the user for the operation to perform. Repeat this step until user's input is valid. Label this part as
'task'.
3. For Enqueue operation :
a. If front== -1, increment value of front by 1
b. Increment value of rear by 1
c. If rear > length of array :
i. If front !=0, use a loop to shift the elements in the array to extreme left from index 0 and
reallocate value of front and rear
ii. Otherwise, decrement value of rear by 1, and print queue is full, and goto task (step 2)
d. Otherwise input element to arr[rear] and ask if the user want to enter another number
e. Otherwise repeat steps a to d;
4. For Dequeue operation :
a. If front==rear, store -1 to front and rear and print"Queue is now empty". Then goto task(Step2)
b. Otherwise increment value of front by 1, print "Element deleted", and ask if user wants to remove
another number
c. If user doesn't, goto task(step2)
a. Otherwise repeat steps a to d;
5. For Peek operation :
c. Print arr[front]
d. goto task(step 2)
6. For printing :
a. For i= front till ==rear, print arr[i]
void main()
1. Ask the user which type data structure he wants to use.
2. Invoke stackop() and queueop() based on users choice
```

Program

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

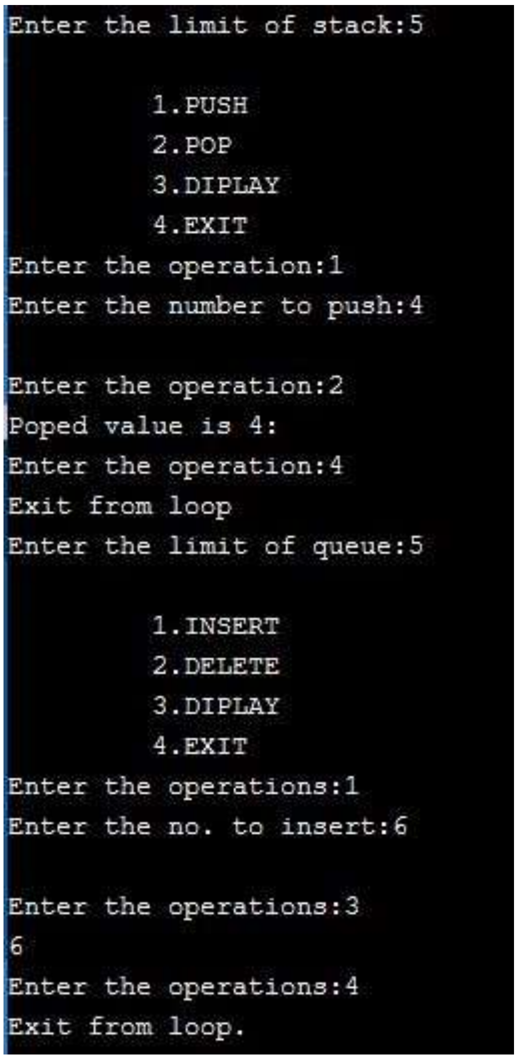
```
int main()
{
    int s[30],n,Q[30],lim;
    printf("Enter the limit of stack:");
    scanf("%d",&n);
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DIPLAY\n\t 4.EXIT");
    stack(s,n);

    printf("\nEnter the limit of queue:");
    scanf("%d",&lim);
    printf("\n\t 1.INSERT\n\t 2.DELETE\n\t 3.DIPLAY\n\t 4.EXIT");
    queue(Q,lim);
}
int stack(int s[10],int n)
{
    int top,op,i,a;
    top=-1;
    do
    {
    printf("\nEnter the operation:");
    scanf("%d",&op);
    switch(op)
    {
        case 1:
        {
        if(top<n)
         {
         top=top+1;
          printf("Enter the number to push:");
          scanf("%d",&a);
          s[top]=a;
        }
```

```
else
{
printf("Stack is full.");
}
break;
}
case 2:
{
   if(top==-1)
{
 printf("Underflow.");
}
else
{
 printf("Poped value is %d:",s[top]);
 top=top-1;
 }
 break;
}
case 3:
{
  if(top>=0)
  {
 for(int i=top;i>=0;i--)
  {
    printf("%d\t",s[i]);
  }
  }
 else
{
printf("Stack is empty.");
}
```

```
        break;
        }
        default:
        {
            printf("Exit from loop");
            break;
        }
    }
    }
    while(op!=4);
    {
    return 0;
    }
}
int queue(int Q[10],int lim)
{
    int front,rear,choice,l,i,item;
    front=rear=-1;
 do{
        printf("\nEnter the operations:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
            if(rear==lim-1)
            {
                printf("Queue is full.");
            }
            else
            {
                front=0;
    printf("Enter the no. to insert:");
    scanf("%d",&l);
    rear=rear+1;
    Q[rear]=l;
}
break;
}
case 2:
{
    if(front==-1)
    {
        printf("Queue is empty.");
    }
    else
    {
        item=Q[front];
        front=front+1;
        printf("Deleted item is %d",item);
        rear=rear-1;
    }
    break;
}
case 3:
{
    for(i=0;i<=rear;i++)
    {
        printf("%d\t",Q[i]);
    }
    break;
}
default:
{
```

```
                printf("Exit from loop.");
                break;
            }
        }
    }
    while(choice!=4);
    {
    return 0;
    }
}
```

Output



```
Enter the limit of stack:5

         1.PUSH
         2.POP
         3.DIPLAY
         4.EXIT
Enter the operation:1
Enter the number to push:4

Enter the operation:2
Poped value is 4:
Enter the operation:4
Exit from loop
Enter the limit of queue:5

         1.INSERT
         2.DELETE
         3.DIPLAY
         4.EXIT
Enter the operations:1
Enter the no. to insert:6

Enter the operations:3
6
Enter the operations:4
Exit from loop.
```

Result: The program was executed and output obtained successfully and was verified

PRIORITY QUEUE, CIRCULAR QUEUE

File Name	priority.c	Compiler	online gdb compiler
Experiment No	5 b	Data Structure	Array
Date	23.10.2020		
Aim	To implement priority queue using C language		

Algorithm

Input: Array arr[]

Output: arr[] after insertion and deletion using priority, circular or deque implementation

Data Structure: Array

```
void priorQueue()
1. Set flag=0
2. Initialize front and rear with -1
3. Ask the user for the type of priority queue to use – Min Priority Queue/ Max Priority Queue. Repeat this
step unless user's input is valid. Label this part as 'select'.
4. For Min Priority Queue :
a. Ask the user for operation to perform. Repeat this step if user's input is invalid. Label this part as 'task'
b. For Enqueue operation :
i. Label this part as 'priorenqueue'
ii. If front == -1, set front=0, and goto 'inp' (Step vA)
iii. Else If front!=0 && rear>=length of arr, use loop and shift the elements in the array to extreme
left starting from index 0, then reset front and rear accordingly , and goto 'inp'(Step vA)
iv. Else If front==0 && rear>=length of arr, print 'Queue is full" and goto task(Step a)
v. Else :
A. Label this part as 'inp'
B. Increment value of rear by 1 and store a number from user to temporary variable 'temp'
C. If front ==0 and rear == 0, store temp to arr[front]
D. Otherwise, do insertion sort to place temp in correct position in the array based on flag==0
or flag==1 (ascending/descending order)
vi. Ask user if he wants to enter another number. If he wants, repeat Step i to vi
```

```
    c. For Dequeue operation:
        i. Label this part as ‘priordequeue’
        ii. If front == rear, set front and rear to -1, and print “Queue is now empty”
        iii. Else if front<0, print “Queue is empty”
        iv. Else, increment value of front by 1 and print “Element deleted”
        v. Ask user if he wants to delete another number. If he does, repeat steps i to v
    d. For peek operation :
        i. Label this part as ‘priorpeek’
        ii. Print arr[front]
    e. For printing, goto priorprint (Step 6)
    f. If flag==1, goto task 2 (Step 5b)
    g. If flag==0, goto task (Step 4a)
5. For Max Priority Queue :
    a. Set flag=1
    b. Label this part as ‘task2’
    c. Ask user to choose an operation to perform. Repeat this until a valid operation is chosen
    d. For Enqueue operation, goto priorenqueue (Step 4b(i))
    e. For Dequeue operation, goto priordequeue (Step 4c(i))
    f. For Peek operation, goto priorpeek (Step 4d(i))
    g. For Printing, goto priorprint (Step 6)
    h. Goto task2
6. For i=front till i<=rear, print arr[i]. Label this part as ‘priorprint’
void deque()
1. Initialize front and rear with -1
2. Ask the user for the operation to perform. Repeat this step until user’s input is valid. Label this part as
   ‘select’.
3. For Insertion at front :
    a. If front== -1, increment value of front and rear by 1
    b. Else if front == 0 && rear<length of arr, shift elements to the right of the array by 1 index using loop,
       and set front = 0 and rear=rear+1
```

c. Else if front == 0 and rear==9, print “the queue is full” and goto select (Step 2)

d. Else, decrement the value of front by 1

e. Input a number from user and store it to arr[front]

f. Ask user if he wants to insert another number at front. If he wants, repeat steps a to f

g. Goto select (Step 2)

4. For Insertion at end :

a. If front== -1,increment value of front and rear by 1

b. Else if front>0 && rear==9, shift the elements in the array to left by 1 index using loop, and set rear=length of arr and front = front+1

c. Else if front==0 && rear==length of arr, print “the queue is full”, and goto select (Step 2)

d. Else increment value of rear by 1

e. Input a number from user and store it to arr[rear]

f. Ask user if he want to enter another number at rear. If he wants, repeat steps a to f

g. Goto select (Step 2)

5. For Deletion at front :

a. If front==-1, print “the queue is empty”, and goto select (Step 2)

b. Else if front==rear, set front and rear = - 1, , print “the queue is now empty”, and goto select (Step 2)

c. Else increment the value of front by 1, and print “Element deleted from front”

d. Ask user if he wants to delete another number from the front. If he wants, then repeat steps a to d

e. Goto select (Step 2)

6. For Deletion at end :

a. If front==-1, print “the queue is empty”, and goto select (Step 2)

b. Else if front==rear, set front and rear = - 1, , print “the queue is now empty”, and goto select (Step 2)

c. Else decrement the value of rear by 1, and print “Element deleted from rear”

d. Ask user if he wants to delete another number from the rear. If he wants, then repeat steps a to d

e. Goto select (Step2)

7. For Printing , print arr[i] for i=front till i<=rear

void circQueue()

1. Initialize front and rear with 0

2. Ask the user for the operation to perform. Repeat this step until user’s input is valid. Label this part as

```
'select'.
3. For Enqueue operation :
a. If front==0, set the value of front and rear to 1
b. Else :
i. Set next = (rear%length of arr)+1
ii. If front==0, set front and rear to 1
iii. Else, print "The queue is full", and goto select (Step 2)
c. Input a number from the user and store to arr[rear]
d. Ask the user if he want to insert another number. If he wants, repeat steps a to d
e. Goto select (Step 2)
4. For Dequeue operation :
a. If front== 0, print "the queue is empty", and goto select (Step 2)
b. Else if front==rear, set front and rear to 0, print "Queue is now empty", and goro select (Step 2)
c. Else set front=(front%length of arr)+1, and print "Element has been deleted"
d. Ask user if he wants to delete another number. If he wants, repeat steps a to d
e. Goto select (Step 2)
5. For Peek operation :
a. Print arr[front]
b. Goto select (Step 2)
6. For Printing , print arr[i] for i=front till i<=rear
void main()
1. Ask user which type of queue, he wants to use. Repeat this until a valid input is obtained
2. Invoke functions priorQueue(), deque(), and circQueue based on the choice
```

Program

```
#include <stdio.h>
#include <ctype.h>
void priorQueue()
{
 int op,temp,front,rear,i,j,flag;
 front=rear=-1;
 char op2;
```

```
int arr[10];
select:
printf("Choose the type of priority queue\n");
printf("1 - Min Priority Queue (The minimum number gets the highest priority)\n");
printf("2 - Max Priority Queue (The maximum number gets the highest priority\n");
fflush(stdin);
scanf("%d",&op);
if(op!=1 && op!=2)
goto select;
switch(op)
{
case 1:
{
flag=0;
do
{
task:
printf("\nSelect the operation : \n");
printf("\te -> Equeue \t\t d -> Dequeue \t\t p -> Peek \t\t f -> Finish\n");
fflush(stdin);
scanf("%c",&op2);
tolower(op2);

if(op2=='e')
{
priorenqueue:
do
{
if(front==-1)
{
front=0;
goto inp;
```

```
}
else if(rear>=9 && front!=0)
{
for(i=0,j=front;j<=rear;i++,j++)
{
arr[i]=arr[j];
}
front = 0;
rear = i;
goto inp;
}
else if(rear>=9)
{
printf("The queue is full");
goto task;
}
else
{
inp:
rear++;
printf("Enter the number\n");
scanf("%d",&temp);
if(front==0 && rear==0)
{
arr[front]=temp;
}
else
{
if(flag==0) //For Min Priority Wueue
{
j=rear-1;
while (j>=0 && arr[j]>temp)
{
arr[j+1] = arr[j];
j--;
}
arr[j+1] = temp;
}
if(flag==1) //For Max Priority Queue
{
j=rear-1;
while (j>=0 && arr[j]<temp)
{
arr[j+1] = arr[j];
j--;
}
arr[j+1] = temp;
}
}
}
fflush(stdin);
printf("Do u want to enter another number? (y/n)\n");
scanf("%c",&op2);
op2=tolower(op2);
}
while(op2=='y');
}
else if(op2=='d')
{
priordequeue:
do
{
if(front==rear)
{
```

```
front=rear=-1;
printf("The queue is now empty\n");
}
else if(front<0)
{
printf("The queue is empty\n");
}
else
{
front++;
printf("Element deleted\n");
}
fflush(stdin);
printf("Do u want to enter another number? (y/n)\n");
scanf("%c",&op2);
op2=tolower(op2);
}
while(op2=='y');
}
else if(op2=='p')
{
priorpeek:
printf("The element at front (highest priority) is :\t%d\n",arr[front]);
}
else if(op2=='f')
goto priorprint;
if(flag==1)
goto task2;
if(flag==0)
goto task;
}
while(op2!='e' && op2!='d' && op2=='p' && op2!='f');
```

```
break;
}
case 2:
{
flag=1;
do
{
task2:
printf("\nSelect the operation : \n");
printf("\te -> Equeue \t\t d -> Dequeue \t\t p -> Peek \t\t f -> Finish\n");
fflush(stdin);
scanf("%c",&op2);
tolower(op2);

if(op2=='e')
goto priorenqueue;
else if(op2=='d')
goto priordequeue;
else if(op2=='p')
goto priorpeek;
else if(op2=='f')
goto priorprint;
goto task2;
}
while(op2!='e' && op2!='d' && op2=='p' && op2!='f');
break;
}
//Print the Priority Queue
priorprint:
printf("---------\nPrinting the queue \n");
for(i=front;i<=rear;i++)
printf("%d\t",arr[i]);
```

```
 }
}
//-------------------------------------------------------------------
void deque()
{
 int arr[10],front,rear,op,i,j,temp;
 char op2;
 front=rear=-1;
 select:
 printf("\nChoose an operation to perform ?\n");
 printf("1 - Insertion at front\n2 - Insertion at end\n3 - Deletion at front\n4 - Deletion at end\n5 - Print\n\n");
 scanf("%d",&op);
 if(op==1)
 {
 do
 {
 if(front==-1)
 {
 front++;
 rear++;
 }
 else if(front==0 && rear<9)
 {
 for(i=rear,j=rear+1;i>=0;i--,j--)
 {
 arr[j]=arr[i];
 }
 front=0;
 rear++;;
 }
 else if(front==0 && rear==9)
 {
```

```
printf("The queue is full\n");
goto select;
}
else
front--;
printf("Enter a number\n");
scanf("%d",&temp);
arr[front]=temp;
printf("Do u want to insert another number at front ? (y/n)\n");
fflush(stdin);
scanf("%c",&op2);
}
while(tolower(op2)=='y');
goto select;
}
else if(op==2)
{
do
{
if(front==-1)
{
front++;
rear++;
}
else if(front>0 && rear==9)
{
for(i=front-1,j=front;j<=rear;i++,j++)
{
arr[i]=arr[j];
}
front--;
rear=9;;
```

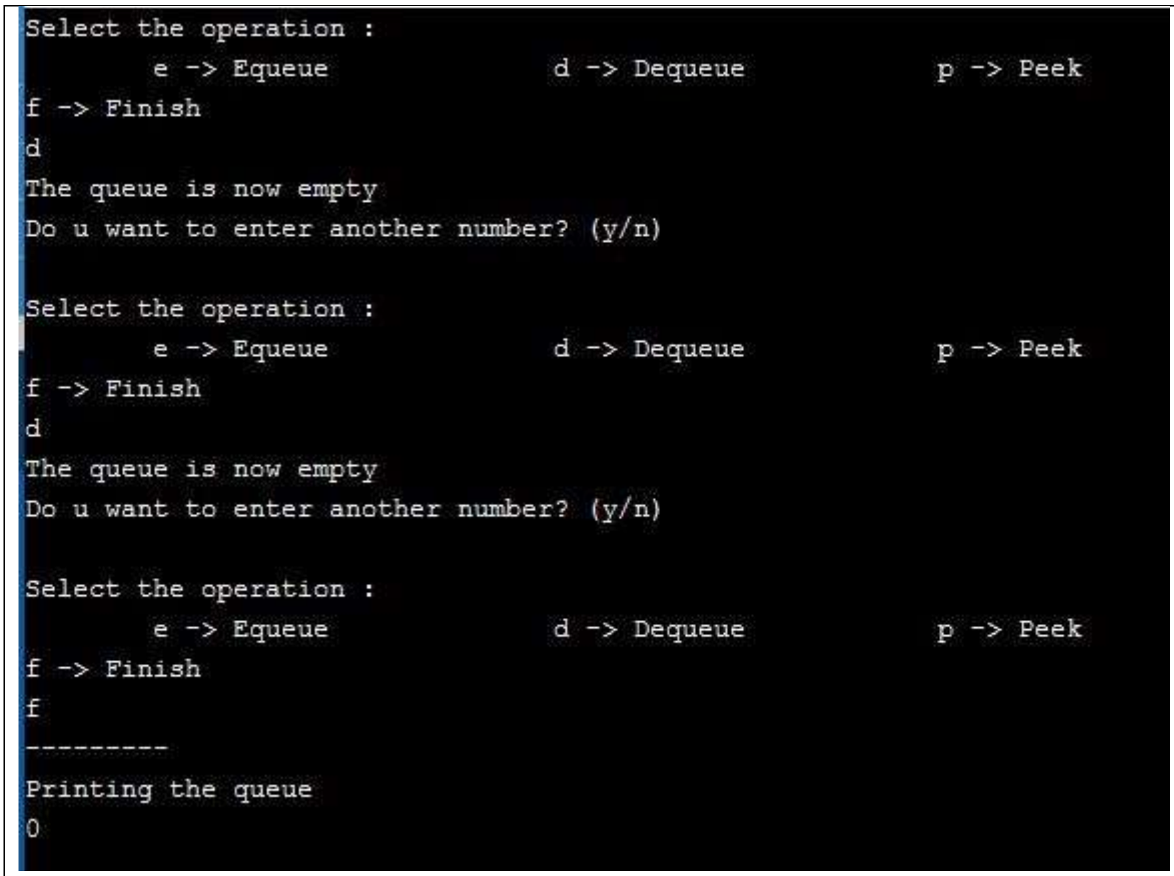
```
}
else if(front==0 && rear==9)
{
printf("The queue is full\n");
goto select;
}
else
rear++;
printf("Enter a number\n");
scanf("%d",&temp);
arr[rear]=temp;
printf("Do u want to insert another number at rear ? (y/n)\n");
fflush(stdin);
scanf("%c",&op2);
}
while(tolower(op2)=='y');
goto select;
}
else if (op==3)
{
do
{
if (front == -1)
{
printf("Queue is empty\n");
goto select;
}
else if(front == rear)
{
front = -1;
rear=-1;
printf("Queue is now empty\n");
```

```
goto select;
}
else
{
front++;
printf("Element has been deleted from front\n");
}
printf("Do u want to delete another number from front ? (y/n)\n");
fflush(stdin);
scanf("%c",&op2);
}
while(tolower(op2)=='y');
goto select;
}
else if (op==4)
{
do
{
if (front == -1)
{
printf("Queue is empty\n");
goto select;
}
else if(front == rear)
{
front = -1;
rear=-1;
printf("Queue is now empty\n");
goto select;
}
else
{
 rear--;
 printf("Element has been deleted from rear\n");
 }
 printf("Do u want to delete another number from rear ? (y/n)\n");
 fflush(stdin);
 scanf("%c",&op2);
 }
 while(tolower(op2)=='y');
 goto select;
 }
 else if(op==5)
 {
 printf("Printing the queue\n");
 for(i=front;i<=rear;i++)
 printf("%d\t",arr[i]);
 }
 else
 goto select;
}
//-----------------------------------------------------------------------------
void circQueue()
{
 int arr[10],next,i,temp,front,rear;
 char op;
 front=rear=0;
 select:
 printf("\nChoose an operation to perform \n");
 printf("e - Enqueue \t\t d- Dequeue \t\t p - Peek \t\t f - finalize and print\n\n");
 fflush(stdin);
 scanf("%c",&op);
 op=tolower(op);
 switch(op)
```

```
{
case 'e':
{
do
{
if(front==0)
front=rear=1;
else
{
next = (rear%9)+1;
if(next!=front)
rear=next;
else
{
printf("The queue is full\n");
goto select;
}
}
printf("Enter a number\n");
scanf("%d",&temp);
arr[rear]=temp;
printf("Do u want to insert another number ? (y/n)\n");
fflush(stdin);
scanf("%c",&op);
}
while(tolower(op)=='y');
goto select;
}
case 'd':
{
do
{
```

```
if (front == 0)
{
printf("Queue is empty\n");
goto select;
}
else if(front == rear)
{
front = 0;
rear= 0;
printf("Queue is now empty\n");
goto select;
}
else
{
front =(front%9)+1;
printf("Element has been deleted\n");
}
printf("Do u want to delete another number? (y/n)\n");
fflush(stdin);
scanf("%c",&op);
}
while(tolower(op)=='y');
goto select;
}
case 'p':
printf("The element at front is %d\n",arr[front]);
goto select;
case 'f':
{
printf("Printing the queue\n");
for (i = front;i<=rear;i++)
{
 printf("%d\t",arr[i]);
 }
 }
 }
}
//------------------------------------------------------------------------------
void main()
{
 char op;
 printf("Which type of queue do u want to use ?\n");
 select:
 printf("Enter your selection :\n");
 printf("\tp -> priority queue \t\t d -> deque \t\t c -> circular queue\n");
 scanf("%c",&op);
 op=tolower(op);
 if(op!='p' && op!='d' && op!='c')
 goto select;
 switch (op)
 {
 case 'p':
 priorQueue();
 break;
 case 'd':
 deque();
 break;
 case 'c':
 circQueue();
 break;
 }
}
```

Output



```
Select the operation :
        e -> Equeue              d -> Dequeue             p -> Peek
f -> Finish
d
The queue is now empty
Do u want to enter another number? (y/n)

Select the operation :
        e -> Equeue              d -> Dequeue             p -> Peek
f -> Finish
d
The queue is now empty
Do u want to enter another number? (y/n)

Select the operation :
        e -> Equeue              d -> Dequeue             p -> Peek
f -> Finish
f
---------
Printing the queue
0
```

Result: The program was executed and output obtained successfully and was verified

CONVERSION OF ONE NOTATION TO ANOTHER NOTATION

File Name	prepostinfix.c	Compiler	online gdb compiler
Experiment No	6	Data Structure	Array
Date	30.11.2020		
Aim	To convert one notation to another notation using C language		

Algorithm

Infix to postfix

INPUT:Character Array of Expression, P.

OUTPUT: Character Array of Stack, Q.

DATA STRUCTURE:Array

```
1. Add ")" at the end of P.
2. Scan P from left to right and  repeat the steps 3&4.
3. If an operand occurs, PUSH it to stack.
4. If an operator occurs, then
      • Remove the top elements of the stack.
      • Evaluate.
      • Place the result of step b back to stack.
5. Set the value equals to TOP element of the stack
```

Infix to prefix

INPUT:Character Array of Expression, P.

OUTPUT: Character Array of Stack, Q.

DATA STRUCTURE:Array

```
1. Push ")" onto STACK, and add "(" to end of the A.
2. Scan A from right to left and repeat stpe 3 to 6 for each element of A until the STACK is empty.
3. If an operand is encountered add it to B.
4. If a right parenthesis is encountered push it onto STACK.
5. If an operator is encountered then:
      a. Repeatedly pop from STACK and add to B each operator.
      b. Add operator to STACK.
6. If left parenthesis is encountered then:
   a. Repeatedly pop from the STACK and add to B.
   b. Remove the left parenthesis.
```

7. Exit.

Program

<u>Infix to postfix</u>

```
#include<stdio.h>
#include<ctype.h>
char stack[100];
int top = -1;
void push(char x)
{
   stack[++top] = x;
}
char pop()
{
   if(top == -1)
      return -1;
   else
      return stack[top--];
}
int priority(char x)
{
   if(x == '(')
      return 0;
   if(x == '+' || x == '-')
      return 1;
   if(x == '*' || x == '/')
      return 2;
   return 0;
}
```

```
int main()
{
    char exp[100];
    char *e, x;
    printf("Enter the expression : ");
    scanf("%s",exp);
    printf("\n");
    e = exp;
    while(*e != '\0')
    {
        if(isalnum(*e))
        printf("%c ",*e);
        else if(*e == '(')
        push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '(')
            printf("%c ", x);
        }
        else
        {
            while(priority(stack[top]) >= priority(*e))
            printf("%c ",pop());
            push(*e);
        }
        e++;
    }
    while(top != -1)
    {
        printf("%c ",pop());
    }
    return 0;
}
```

Infix to prefix

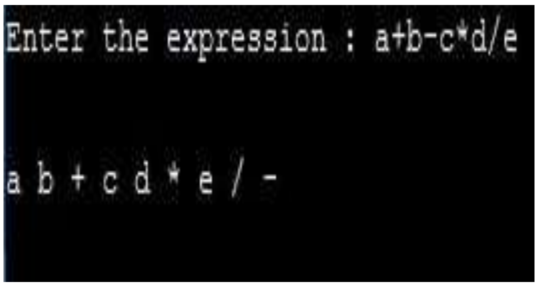
```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>

#define BLANK ' '
#define TAB '\t'
#define MAX 50

long int pop();
long int eval_pre();
char infix[MAX], prefix[MAX];
long int stack[MAX];
int top;
int isempty();
int white_space(char symbol);

void infix_to_prefix();
int priority(char symbol);
void push(long int symbol);
long int pop();
long int eval_pre();

int main()
{
        long int value;
        top = -1;
        printf("Enter infix : ");
        gets(infix);
        infix_to_prefix();
        printf("prefix : %s\n",prefix);
        value=eval_pre();
        printf("Value of expression : %ld\n",value);
        return 0;
}
void infix_to_prefix()
{
        int i,j,p,n;
        char next ;
        char symbol;
        char temp;
        n=strlen(infix);
        p=0;

        for(i=n-1; i>=0; i--)
        {
                symbol=infix[i];
                if(!white_space(symbol))
                {
                        switch(symbol)
                        {
                        case ')':
                                push(symbol);
                                break;
                        case '(':
                                while( (next=pop()) != ')')
                                        prefix[p++] = next;
                                break;
                        case '+':
                        case '-':
```

```
                case '*':

                case '/':

                case '%':

                case '^':

                        while( !isempty( ) &&  priority(stack[top])> priority(symbol) )

                        prefix[p++] = pop();

                        push(symbol);

                        break;

                default:

                     prefix[p++] = symbol;

                }

        }

    }

    while(!isempty( ))

            prefix[p++] = pop();

    prefix[p] = '\0';

    for(i=0,j=p-1;i<j;i++,j--)

    {

            temp=prefix[i];

            prefix[i]=prefix[j];

            prefix[j]=temp;

    }

}

int priority(char symbol )

{

    switch(symbol)

    {

    case ')':

            return 0;

    case '+':

    case '-':

            return 1;
    case '*':
    case '/':
    case '%':
            return 2;
    case '^':
            return 3;
    default :
             return 0;
    }
}

void push(long int symbol)
{
    if(top > MAX)
    {
            printf("Stack overflow\n");
            exit(1);
    }
    else
    {
            top=top+1;
            stack[top] = symbol;
    }
}

long int pop()
{
    if(top == -1 )
    {
            printf("Stack underflow \n");
            exit(2);
    }
    return (stack[top--]);
}
int isempty()
{
    if(top==-1)
            return 1;
    else
            return 0;
}
int white_space(char symbol)
{
    if(symbol==BLANK || symbol==TAB || symbol=='\0')
            return 1;
    else
            return 0;
}
long int eval_pre()
{
    long int a,b,temp,result;
    int i;

    for(i=strlen(prefix)-1;i>=0;i--)
    {
            if(prefix[i]<='9' && prefix[i]>='0')
                    push( prefix[i]-48 );
            else
            {
                    b=pop();
                    a=pop();
                    switch(prefix[i])
                    {
                    case '+':
```

```
                        temp=b+a; break;
                case '-':
                        temp=b-a;break;
                case '*':
                        temp=b*a;break;
                case '/':
                        temp=b/a;break;
                case '%':
                        temp=b%a;break;
                case '^':
                        temp=pow(b,a);
                }
                push(temp);
        }
    }
    result=pop();
    return result;
}
```

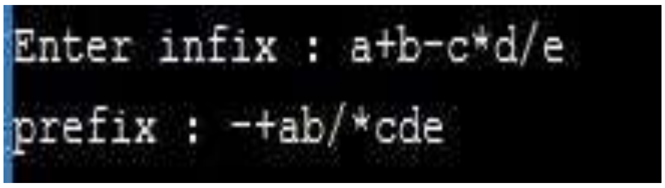
Output

Infix to postfix



```
Enter the expression : a+b-c*d/e

a b + c d * e / -
```

Infix to prefix



```
Enter infix : a+b-c*d/e
prefix : -+ab/*cde
```

Result: The program was executed and output obtained successfully and was verified

SINGLY LINKED LIST

File Name	Singlylinkedlist.c	Compiler	online gdb compiler
Experiment No	7	Data Structure	Array
Date	6.11.2020		
Aim	To implement singly linked list using C language		

Algorithm

Input: A singly linked list with data.

Output: A singly list after performing the desired operation.

Data structure: Singly linked list whose pointer to the header node is the HEADER.

1.INSERTION AT BEGINNING

```
• Algorithm: Insert a new node temp with data 'item'
1. Create a new node temp //struct node temp = (struct node)
2. If (temp==NULL) malloc(sizeof(struct node));
3. print "memory underflow, no insertion"
4. else
5. temp->data= item
6. temp-> link=head
7. head=temp
```

2.INSERTION AT END

```
Traverse()
 1. Set ptr=head; //initialize the pointer ptr
 2. While (ptr->link!=null) do
3. ptr= ptr->link; //ptr now points to the next node
 4. ptr->link= temp
 5. temp->data=item
```

3.INSERTION AT ANY POSITION

```
1. Read the value key of node after which a new node is to be placed
2. Set ptr=head
 3. Repeat while (ptr-> data!=key) and (ptr->link!=NULL)
4. ptr=ptr-> link
5. If (ptr->link==NULL)
 6. print “search fails”);
 7. else
 8. temp->link= ptr-> link
 9. ptr->link= temp
```

4.DELETION AT BEGINNING

```
1. If (head==NULL)
 2. Print ‘list empty’
3. Else
 4. head=head-> link
```

5.DELETION AT END

```
1. Set current=head, prev=head
2. Repeat while(current-> link!=null)
3. prev= current
4. current=current->link
5. prev->link=null
```

6.DELETION FROM ANY POSITION

```
1. Read the value key that is to be deleted
 2. Set current=head, prev=head
 3. Repeat while(current->link!=null)&& (current-> data!=key)
 4. prev= current
 5. current=current->link
 6. If (current==null)
```

```
7. print “element not found”
8. Else
 9. prev->link=current->link
```

7.TRAVERSAL

```
Traverse()
1. Set ptr=head; //initialize the pointer ptr
2. While (ptr!=null) do
3. print ptr->data
4. ptr= ptr->link
```

Program

```
#include<stdio.h>
#include<stdlib.h>
void createList();
void displayList();
void insertNodeAtBeginning();
void dList();
void insertNodeAtEnd();
void diList();
void insertNodeAtMiddle();
void disList();
void deleteFirstNode();
void dispList();
void deleteLastNode();
void displList();
void deleteMiddleNode();
void displaList();
void traverseList();
struct node
{
    int data;
    struct node *next;
}*head;
void createList(int n)
{
    struct node *newNode, *temp;
    int data,i;
    head=(struct node *)malloc(sizeof(struct node));
    if(head==NULL)
    {
        printf("Unable to allocate memory");
        exit(0);
    }
    printf("Enter the data of node 1:");
    scanf("%d",&data);
    head->data=data;
    head->next=NULL;
    temp=head;
    for(i=2;i<=n;i++)
    {
        newNode=(struct node *)malloc(sizeof(struct node));
        if(newNode==NULL)
        {
            printf("Unable to allocate memory.");
            break;
        }
        printf("Enter the data of node %d:",i);
        scanf("%d",&data);
        newNode->data=data;
        newNode->next=NULL;
        temp->next=newNode;
        temp=temp->next;
    }
}
void displayList()
{
    struct node *temp;
    if(head==NULL)
    {
        printf("List is Empty.");
    }
    else
    {
        temp=head;
        while(temp!=NULL)
        {
            printf("Data=%d\n",temp->data);
            temp=temp->next;
        }
    }
}
void insertNodeAtBeginning(int data)
{
    struct node *newNode;
    newNode=(struct node *)malloc(sizeof(struct node));
    if(newNode==NULL)
    {
        printf("Unable to allocate memory.");
    }
    else
    {
        newNode->data=data;
        newNode->next=head;
        head=newNode;
        printf("Data inserted successfully\n");
    }
}
void dList()
{
    struct node *temp;
    if(head==NULL)
    {
        printf("List is Empty.");
    }
    else
    {
        temp=head;
        while(temp!=NULL)
        {
            printf("Data=%d\n",temp->data);
            temp=temp->next;
        }
    }
}
void insertNodeAtEnd(int data)
{
    struct node *newNode, *temp;
    newNode=(struct node *)malloc(sizeof(struct node));
    if(newNode==NULL)
    {
        printf("Unable to allocate memory.");
    }
    else
    {
        newNode->data=data;
        newNode->next=NULL;
        temp=head;
        while(temp!=NULL&&temp->next!=NULL)
        temp=temp->next;
        temp->next=newNode;
        printf("Data inserted successfully\n");
    }
}
void diList()
{
    struct node *temp;
    if(head==NULL)
    {
        printf("List is Empty.");
    }
    else
    {
        temp=head;
        while(temp!=NULL)
        {
            printf("Data=%d\n",temp->data);
            temp=temp->next;
        }
    }
}
void insertNodeAtMiddle(int data, int position)
{
    int i;
    struct node *newNode, *temp;
    newNode = (struct node*)malloc(sizeof(struct node));
    if(newNode == NULL)
    {
        printf("Unable to allocate memory.");
    }
    else
    {
        newNode->data = data;
        newNode->next = NULL;
        temp = head;
        for(i=2; i<=position-1; i++)
        {
            temp = temp->next;

            if(temp == NULL)
                break;
        }
        if(temp != NULL)
        {
            newNode->next = temp->next;
            temp->next = newNode;
            printf("DATA INSERTED SUCCESSFULLY\n");
        }
        else
        {
            printf("UNABLE TO INSERT DATA AT THE GIVEN POSITION\n");
        }
    }
}
void disList()
{
    struct node *temp;
    if(head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        while(temp != NULL)
        {
            printf("Data = %d\n", temp->data);
            temp = temp->next;
        }
    }
}
void deleteFirstNode()
{
    struct node *toDelete;

    if(head == NULL)
    {
        printf("List is already empty.");
    }
    else
    {
        toDelete = head;
        head = head->next;
        printf("\nData deleted = %d\n", toDelete->data);
        free(toDelete);
        printf("SUCCESSFULLY DELETED FIRST NODE FROM LIST\n");
    }
}
void dispList()
{
    struct node *temp;
    if(head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        while(temp != NULL)
        {
            printf("Data = %d\n", temp->data);
            temp = temp->next;
        }
    }
}
void deleteLastNode()
{
    struct node *toDelete, *secondLastNode;
    if(head == NULL)
    {
        printf("List is already empty.");
    }
    else
    {
        toDelete = head;
        secondLastNode = head;
        while(toDelete->next != NULL)
        {
            secondLastNode = toDelete;
            toDelete = toDelete->next;
        }

        if(toDelete == head)
        {
            head = NULL;
        }
        else
        {
            secondLastNode->next = NULL;
        }
        free(toDelete);

        printf("SUCCESSFULLY DELETED LAST NODE OF LIST\n");
    }
}
void displList()
{
    struct node *temp;
    if(head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        while(temp != NULL)
        {
            printf("Data = %d\n", temp->data);
            temp = temp->next;
        }
    }
}

void deleteMiddleNode(int position)
{
    int i;
    struct node *toDelete, *prevNode;
    if(head == NULL)
    {
        printf("List is already empty.");
    }
    else
    {
        toDelete = head;
        prevNode = head;
        for(i=2; i<=position; i++)
        {
            prevNode = toDelete;
            toDelete = toDelete->next;
            if(toDelete == NULL)
                break;
        }
        if(toDelete != NULL)
        {
            if(toDelete == head)
                head = head->next;
            prevNode->next = toDelete->next;
            toDelete->next = NULL;
            free(toDelete);
            printf("SUCCESSFULLY DELETED NODE FROM MIDDLE OF LIST\n");
        }
        else
        {
            printf("Invalid position unable to delete.");
        }
    }
}
void displaList()
```

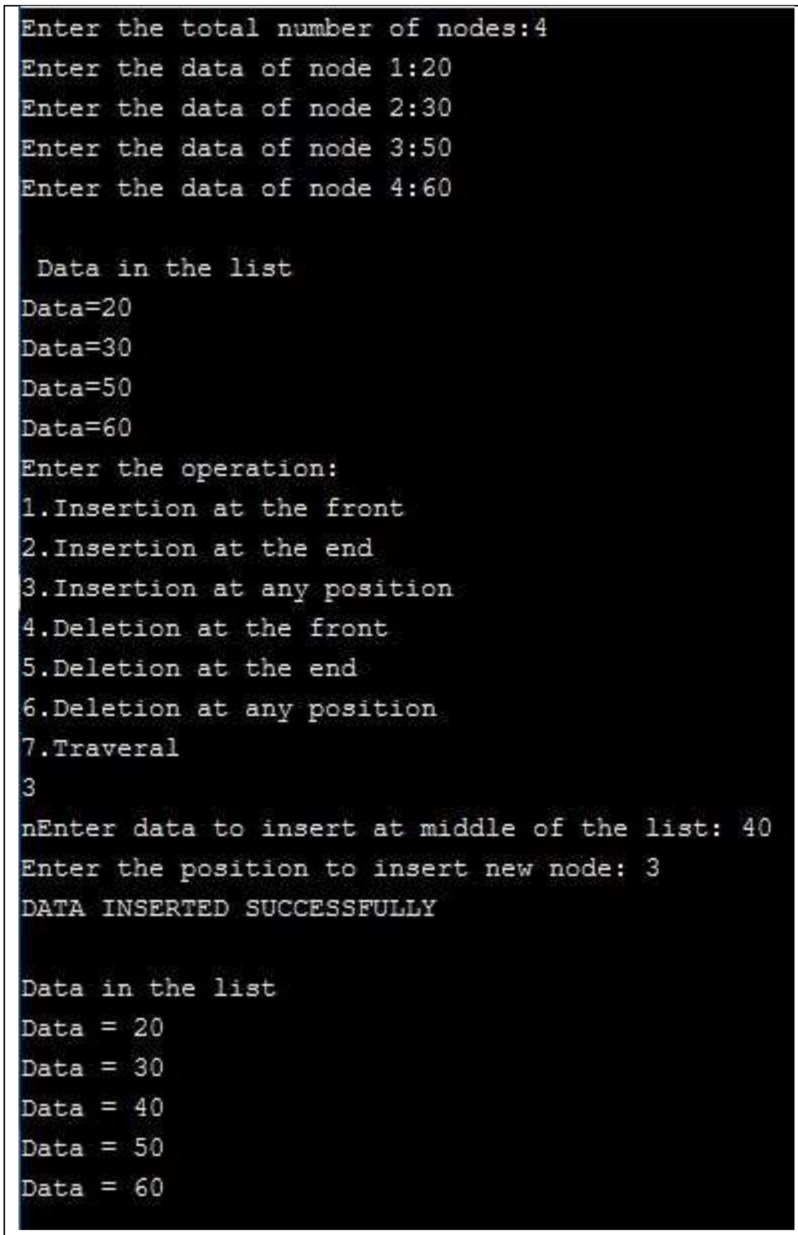
```
{
    struct node *temp;
    if(head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        while(temp != NULL)
        {
            printf("Data = %d\n", temp->data);
            temp = temp->next;
        }
    }
}
void traverseList()
{
    struct node *temp;
    if(head==NULL)
    {
        printf("List is empty.");
        return;
    }
    temp=head;
    while(temp!=NULL)
    {
        printf("Data=%d\n",temp->data);
        temp=temp->next;
    }
}
int main()
```

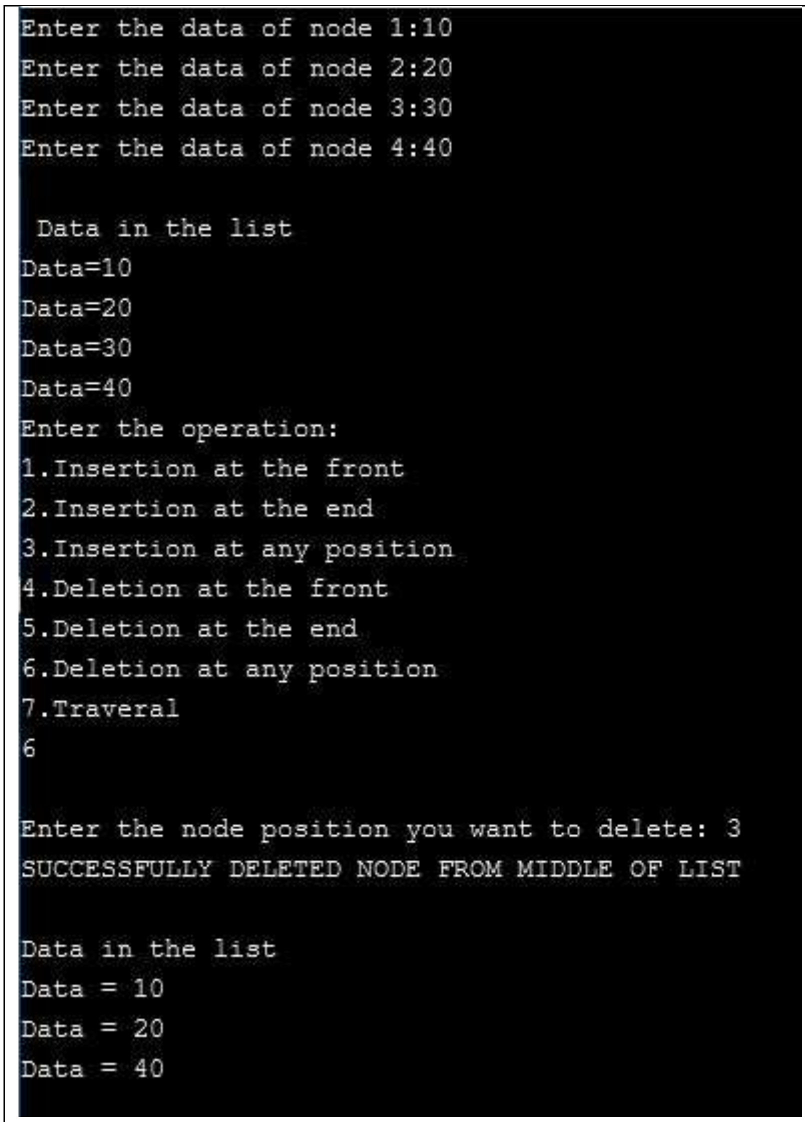
```
{
    int n,data,position,choice,option;
    printf("Enter the total number of nodes:");
    scanf("%d",&n);
    createList(n);
    printf("\n Data in the list \n");
    displayList();
    printf("Enter the operation:\n1.Insertion at the front \n2.Insertion at the end \n3.Insertion at any position
\n4.Deletion at the front \n5.Deletion at the end \n6.Deletion at any position \n7.Traveral\n");
    scanf("\n%d",&option);
    switch(option)
    {
    case 1:
    {
    printf("\nEnter data to be inserted at beginning of the list:");
    scanf("%d",&data);
    insertNodeAtBeginning(data);
    printf("\n Data in the list \n");
    dList();
    break;
    }
    case 2:
    {
    printf("\nEnter data to be inserted at end of the list:");
    scanf("%d",&data);
    insertNodeAtEnd(data);
    printf("\n Data in the list \n");
    diList();
    break;
    }
    case 3:
    {
```

```
printf("nEnter data to insert at middle of the list: ");
scanf("%d", &data);
printf("Enter the position to insert new node: " );
scanf("%d", &position);
insertNodeAtMiddle(data, position);
printf("\nData in the list \n");
disList();
break;
}
case 4:
{
printf("\nPress 1 to delete first node: ");
scanf("%d", &choice);
if(choice == 1)
deleteFirstNode();
printf("\nData in the list \n");
dispList();
break;
}
case 5:
{
printf("\nPress 1 to delete last node: ");
scanf("%d", &choice);
if(choice == 1)
deleteLastNode();
printf("\nData in the list \n");
displList();
break;
}
case 6:
{
printf("\nEnter the node position you want to delete: ");
```

```
    scanf("%d", &position);
    deleteMiddleNode(position);
    printf("\nData in the list \n");
    displaList();
    break;
    }
    case 7:
    {
    printf("\n Data in the list \n");
    traverseList();
    break;
    }
    }
    return 0;
}
```

Output


```
Enter the total number of nodes:4
Enter the data of node 1:20
Enter the data of node 2:30
Enter the data of node 3:50
Enter the data of node 4:60

 Data in the list
Data=20
Data=30
Data=50
Data=60
Enter the operation:
1.Insertion at the front
2.Insertion at the end
3.Insertion at any position
4.Deletion at the front
5.Deletion at the end
6.Deletion at any position
7.Traveral
3
nEnter data to insert at middle of the list: 40
Enter the position to insert new node: 3
DATA INSERTED SUCCESSFULLY

Data in the list
Data = 20
Data = 30
Data = 40
Data = 50
Data = 60
```



```
Enter the data of node 1:10
Enter the data of node 2:20
Enter the data of node 3:30
Enter the data of node 4:40

 Data in the list
Data=10
Data=20
Data=30
Data=40
Enter the operation:
1.Insertion at the front
2.Insertion at the end
3.Insertion at any position
4.Deletion at the front
5.Deletion at the end
6.Deletion at any position
7.Traveral
6

Enter the node position you want to delete: 3
SUCCESSFULLY DELETED NODE FROM MIDDLE OF LIST

Data in the list
Data = 10
Data = 20
Data = 40
```

Result: The program was executed and output obtained successfully and was verified

POLYNOMIAL ADDITIN USING LINKED LIST

File Name	Singlylinkedlist.c	Compiler	online gdb compiler
Experiment No	8	Data Structure	Linked list
Date	13.11.2020		
Aim	To implement polynomial addition using linked list using C language		

Algorithm

Input: Two polynomials P and Q whose header points are PHEADER and QHEADER.

Output: A polynomial R is the sum of P and Q having the header RHEADER.

Data Structure: Singly linked list structure for representing a term in a single variable polynomial.

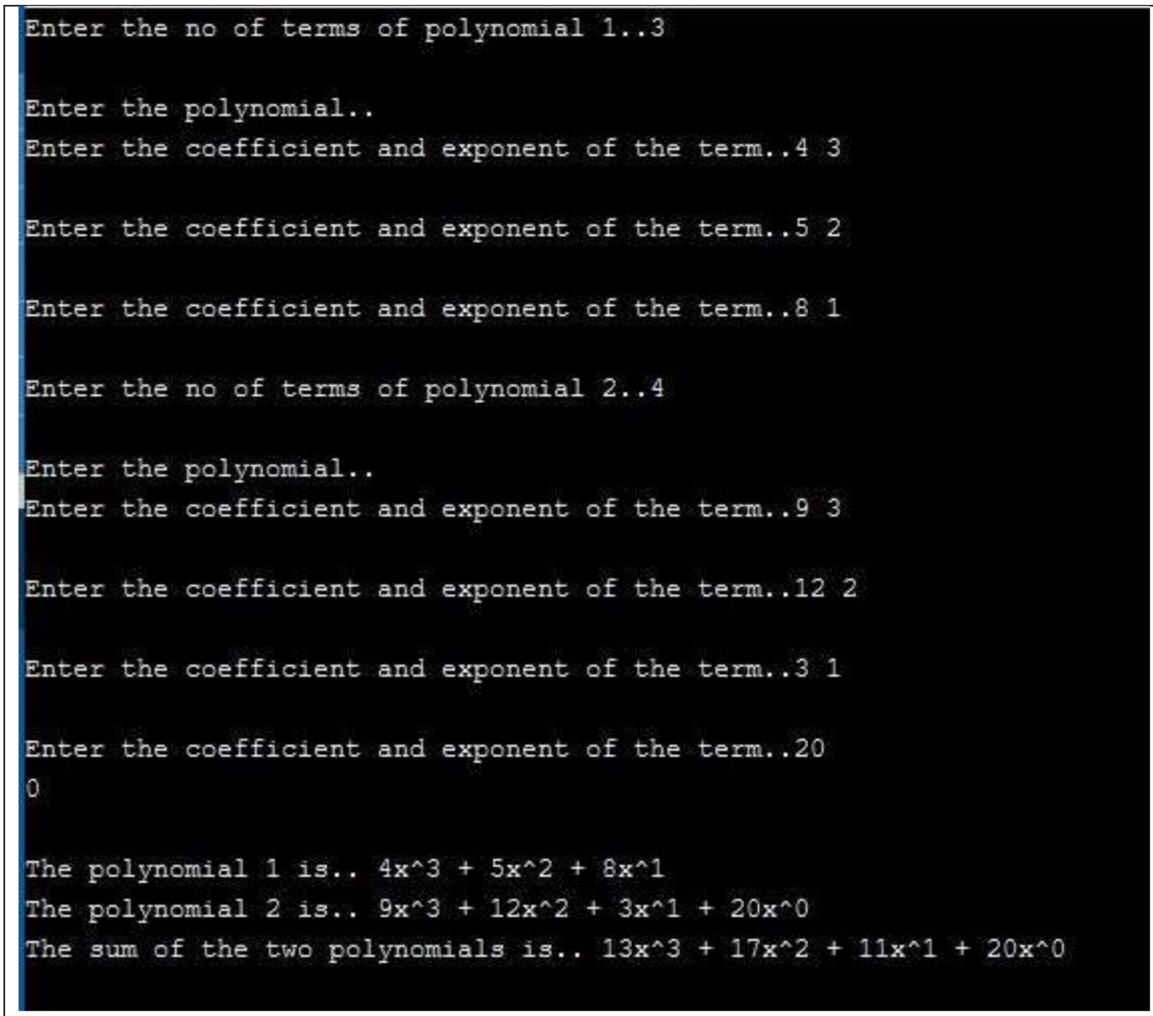
```
1. Pptr=PHEADER->LINK, Qptr=QHEADER->LINK
2. RHEADER= GetNode(NODE)
3. RHEADER->LINK=NULL, RHEADER->EXP=NULL, RHEADER->COEFF=NULL
4. Rptr=RHEADER
5. While(Pptr!=NULL) and (Qptr!=NULL)do
6. Case: Pptr->EXP=Qptr->EXP
7. new=GetNode(NODE)
8. Rptr->LINK=new,Rptr=new
9. Rptr->COEFF=Pptr->COEFF+Qptr->COEFF
10. Rptr->EXP=Pptr->EXP
11. Rptr->LINK=null
12. Pptr=Pptr->LINK=new,Qptr=Qptr->LINK
13. Case: Pptr->EXP>Qptr->EXP
14. new=GetNode(NODE)
15. Rptr->LINK=new, Rptr=new
16. Rptr->COEFF=Pptr->COEFF
17. Rptr->EXP=Pptr->EXP
18. Rptr->LINK=NULL
19. Pptr=Pptr->LINK
20. CASE: Pptr->EXP<Qptr->EXP
21. new=GetNode(NODE)
22. Rptr->LINK=new, Rptr=new
23. Rptr->COEFF=Qptr->COEFF
24. Rptr->EXP=Qptr->EXP
25. Rptr->LINK=NULL
26. Qptr=Qptr->LINK
27. Endwhile
28. If(Pptr!=NULL)and(Qptr=NULL)then
29. While (Pptr!=NULL)do
30. new=GetNode(NODE)
31. Rptr->LINK=new, Rptr=new
32. Rptr->COEFF=Pptr->COEFF
33. Rptr->EXP=Pptr->EXP
34. Rptr->LINK=NULL
35. Pptr=Pptr->LINK
```

```
36. Endwhile
37. Endif
38. If(Pptr=NULL)and(Qptr!=NULL)then
39. While (Qptr!=NULL)do
40. new=GetNode(NODE)
41. Rptr->LINK=new, Rptr=new
42. Rptr->COEFF=Qptr->COEFF
43. Rptr->EXP=Qptr->EXP
44. Rptr->LINK=NULL
45. Qptr=Qptr->LINK
46. Endwhile
47. Endif
48. Return(RHEADER)
49. Stop
```

Program

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

   int coef,expo;

   struct node* next;

};

struct node* insertpoly(struct node* thead,int c,int e);

struct node* append(struct node* thead,int c,int e);

struct node* polyaddition(struct node* p1thead,struct node* p2thead);

void display(struct node* thead);


void main()

{

   int a,b,n,i;

   struct node* p1head,* p2head,* p3head;

   p1head=p2head=NULL;

   printf("Enter the no of terms of polynomial 1..");

   scanf("%d",&n);

   printf("\nEnter the polynomial..");

   for(i=0;i<n;i++){
       printf("\nEnter the coefficient and exponent of the term..");
       scanf("%d%d",&a,&b);
       p1head=insertpoly(p1head,a,b);
   }
   printf("\nEnter the no of terms of polynomial 2..");
   scanf("%d",&n);
   printf("\nEnter the polynomial..");
   for(i=0;i<n;i++){
       printf("\nEnter the coefficient and exponent of the term..");
       scanf("%d%d",&a,&b);
       p2head=insertpoly(p2head,a,b);
   }
   p3head=polyaddition(p1head,p2head);
   printf("\nThe polynomial 1 is..");
   display(p1head);
   printf("\nThe polynomial 2 is..");
   display(p2head);
   printf("\nThe sum of the two polynomials is..");
   display(p3head);
}
struct node* append(struct node* thead,int c,int e)
{
   struct node* newnode = (struct node*)malloc(sizeof(struct node));
   newnode->coef=c;
   newnode->expo=e;
   if(thead==NULL){
   newnode->next=NULL;
   return newnode;
   }
   struct node* trav=thead;
   while(trav->next!=NULL)
       trav=trav->next;
   trav->next=newnode;
   newnode->next=NULL;
   return thead;
}

struct node* insertpoly(struct node* thead,int c,int e)
{
   struct node* newnode=(struct node*)malloc(sizeof(struct node));
   newnode->coef=c;
   newnode->expo=e;
   if(thead==NULL){
       newnode->next=NULL;
       return newnode;
   }
   struct node* prev,* curr;
   prev=curr=thead;
   while(curr!=NULL && curr->expo>e){
       prev=curr;
       curr=curr->next;
   }
   if(curr==thead){
       newnode->next=curr;
       return newnode;
   }
   else if(curr==NULL){
       prev->next=newnode;
       newnode->next=NULL;
   }
   else{
       newnode->next=curr;
       prev->next=newnode;
   }
   return thead;
}

struct node* polyaddition(struct node* p1thead,struct node* p2thead)
{
   struct node* ans=NULL;
   struct node* t1,* t2;
   t1=p1thead;
   t2=p2thead;
   while(t1!=NULL && t2!=NULL){
       if(t1->expo > t2->expo){
           ans=append(ans,t1->coef,t1->expo);
           t1=t1->next;
       }
       else if(t1->expo < t2->expo){
           ans=append(ans,t2->coef,t2->expo);
           t2=t2->next;
       }
       else{
           ans=append(ans,(t1->coef)+(t2->coef),t1->expo);
           t1=t1->next;
           t2=t2->next;
       }
   }

   while(t1!=NULL){
       ans=append(ans,t1->coef,t1->expo);
       t1=t1->next;
   }

   while(t2!=NULL){
       ans=append(ans,t2->coef,t2->expo);
       t2=t2->next;
   }
   return ans;
}

void display(struct node* thead)
{
   struct node* temp=thead;
   if(temp==NULL){
       printf("\nEmpty..");
   }
   else{
       while(temp->next!=NULL){
           printf(" %dx^%d +",temp->coef,temp->expo);
           temp=temp->next;
       }
      printf(" %dx^%d ",temp->coef,temp->expo);
   }
}
```

Output



```
Enter the no of terms of polynomial 1..3

Enter the polynomial..
Enter the coefficient and exponent of the term..4 3

Enter the coefficient and exponent of the term..5 2

Enter the coefficient and exponent of the term..8 1

Enter the no of terms of polynomial 2..4

Enter the polynomial..
Enter the coefficient and exponent of the term..9 3

Enter the coefficient and exponent of the term..12 2

Enter the coefficient and exponent of the term..3 1

Enter the coefficient and exponent of the term..20
0

The polynomial 1 is.. 4x^3 + 5x^2 + 8x^1
The polynomial 2 is.. 9x^3 + 12x^2 + 3x^1 + 20x^0
The sum of the two polynomials is.. 13x^3 + 17x^2 + 11x^1 + 20x^0
```

Result: The program was executed and output obtained successfully and was verified

BINARY TREE

File Name	binarytree.c	Compiler	online gdb compiler
Experiment No	9	Data Structure	Array &Linked List
Date	27.11.2020		
Aim	To implement binary tree using array using C language		

Algorithm

```
1. Set int lvl=-1

2. Create int arr[16]

3. Create a structure BST having data and two self-referencing variable which link to its childs and create
a variable ‘node’ for the structure

int search_Seq(int key)

1. For int i=0 till t

Void insertArray(int key, int item)

1. Set int loc = search_Seq(key)

2. If loc==0, print “no such key element found”

3. Otherwise if arr[2*loc]==0 or arr[2*loc+1]==0,

       A. Set a label choice here

       B. Ask user to which node he wants to insert

       C. If user chooses left child,

            a. If arr[2*loc]==0, arr[2*loc] = item

            b. Else, print “Insertion not possible at left child” D. If user chooses right child,

            c. If arr[2*loc+1]==0, arr[2*loc+1] = item

          d. Else, print “Insertion not possible at right child”

      E. If user input wrong choice, print wrong choice and goto choice (Step 3A)
```

```
 4. Else, print “Item cannot be inserted”
Void deleteArray(int item)
1. Set flag=0;
2. Set int loc = search_Seq(item)
3. If loc==0, print “no such key element found”
 4. Otherwise if arr[2*loc]==0 and arr[2*loc+1]==0, set falg=1 and arr[loc]=0
5. Else, print “The node to be deleted is not leaf node”
6. If flag==0, print “No deletion”
7. Otherwise, print “Node deleted”
Int get_right_child(int index)
1. If arr[index]!=0 and (2*index)+1 < size of arr, return (2*index +1)
2. Return -1
Int get_right_child(int index)
1. If arr[index]!=0 and (2*index) < size of arr, return (2*index)
2. Return -1
Void preorderArray(int indx)
    1. If indx>0 and arr[indx]!=0
        A. Print arr[indx]
        B. preorderArray(get_left_child(indx))
        C. preorderArray(get_right_child(indx))
 void postorderArray(int index)
    1. If indx>0 and arr[indx]!=0
       A. preorderArray(get_left_child(indx))
       B. preorderArray(get_right_child(indx))
    C. Print arr[indx]
void inorderArray(int index)
1. If indx>0 and arr[indx]!=0
    A. preorderArray(get_left_child(indx))
    B. Print arr[indx]
    C. preorderArray(get_right_child(indx))
void arrayTree()
1. Ask the user for the operation to perform
2. For insertion,
        A. If lvl=-1, Ask the user to enter an element for root node and store it in arr[1]. Then increment the value          of lvl by 1
        B. Otherwise, Ask the user for location to enter the item and item. Invoke insertArray() and pass location     and item as parameters
        C. Ask the user, if he wants to continue operation. If he wants, goto Step 1
3. For deletion,
   A. Ask the user for the item to be deleted
   B. Invoke deleteArray(item)
   C. Ask the user if he wants to continue operations. If he wants , goto Step 1 4.
For traversal :
    A. Ask the user for the type of traversal
    B. B. For inorder traversal, invoke inorderTraversal(1)
    C. C. For preorder traversal, invoke preorderTraversal(1)
    D. D. For postorder traversal, invoke postorderTraversal(1);
    E. E. For wrong choice, repeat Step 4
    F. F. Ask the user if he wants to continue operation. If he wants, goto Step 1
 Node * create()
 1. Ask the user to enter value
2. Create a new node using malloc for structure and store its address to pointer temp
3. Set the entered value as the data for temp
4. Return temp
 Void insert()
Void insert(node *root, node *temp)
 1. If data of temp is less than data of root
    A. If left child of root is not NULL, invoke insert(root->left, temp)
    B. Otherwise set root-> left as temp
2. If data of temp is greater than data of root
     A. If right child is not NULL, inovoke(root->right, temp)
     B. Otherwise set root->right as temp Node * Searchpar(node *root, int val)
1. If data of root is not equal to val,
    A. Set ptr1 = root->left and ptr2 = root->right
    B. If ptr1 is not NULL, invoke Searchpar(ptr1,val)
    C. Otherwise, set parent = NULL
    D. if ptr2 is not NULL, invoke Searchpar(ptr2, val)
   E. Otherwise, set parent = NULL
2. Otherwise return parent
int delete_tree(node * root, int val)
1. Set flag = 0, and ptr = root
2. While ptr is not NULL and flag = 0
     A. If val is less than ptr->data, set par=ptr and ptr = ptr->left
     B. If val is greater than ptr->data, set par = ptr and ptr = ptr->right
    C. If ptr->data == val, set flag = 1
3. If flag==0, print item doesn’t exist
4. If ptr->left and ptr->right = NULL,
    A. If par->left = ptr , set par->left = NULL
    B. Otherwise, set par->right = NULL
Void preorder(node * root)
1. If root is not NULL,
    A. Print root->data
    B. Invoke preorder(root->left)
    C. Invoke preorder(root->right)
Void inorder(node * root)
1. If root is not NULL,
    A. Invoke preorder(root->left)
    B. Print root->data
    C. Invoke preorder(root->right)
Void postorder(node * root)
1. If root is not NULL,
    A. Invoke preorder(root->left)
    B. Invoke preorder(root->right)
    C. Print root->data
Void main()
1. Ask the type of binary tree representation user wants to use
2. If he choose array, invoke arrayTree()
3. If he choose linkedlist, invoke listTree()
```

4. For wrong choice, goto Step 1

Program

```
#include <stdio.h>

#include <stdlib.h>

int lvl=-1;

int arr[16];

typedef struct BST

{

        int data;

        struct BST *left;

        struct BST *right;

}node;

//-------------------------------------------------------------------------------------

int search_Seq(int key)

{

        int i;

        for(i=1;i<=15;i++)

        {

                if(arr[i]==key)

                        return i;

        }

        return 0;

}
```

```
void insertArray(int key, int item)
{
        int loc,op;
        loc = search_Seq(key);
        if(loc==0)
                printf("No such key element found\n");
        else if((arr[2*loc] == 0) || (arr[(2*loc)+1]==0))
        {
                choice:
                printf("Which child node do u want to enter data to ? \n 1 - Left child \t\t 2 - Right
child\n");
                scanf("%d",&op);
                if(op==1)
                {
                        if(arr[2*loc]==0)
                        {
                                arr[2*loc]=item;
                                lvl++;
                        }
                        else
                                printf("Insertion not possible at left child\n");
                }
                else if(op==2)
                {
                        if(arr[(2*loc)+1]==0)
                        {
                                arr[(2*loc)+1]=item;
                                lvl++;
                        }
                        else
                                printf("Insertion not possible at right child\n");
                }
                else
                {
                        printf("Wrong choice\n");
                        goto choice;
                }
        }
        else
                printf("Item cannot be inserted as leaf node\n");
}
void deleteArray(int item)
{
        int flag=0;
        int loc =  search_Seq(item);
        if(loc==0)
                printf("No such key element found\n");
        else if((arr[2*loc] == 0) && (arr[(2*loc)+1]==0))
        {
                flag=1;
                arr[loc]=0;
        }
        else
                printf("The node to be deleted is not leaf node\n");
        if(flag==0)
                printf("No deletion\n");
        else
                printf("Deleted\n");
}
int get_right_child(int index)
{
   if(arr[index]!=0 && ((2*index)+1)<=15)
       return (2*index)+1;
   return -1;
}
int get_left_child(int index)
{
   if(arr[index]!=0 && (2*index)<=15)
       return 2*index;
   return -1;
}
```

```
void preorderArray(int indx)
{
	if(indx>0 && arr[indx]!=0)
	{
		printf("%d\t",arr[indx]);
		preorderArray(get_left_child(indx));
		preorderArray(get_right_child(indx));
	}
}
void postorderArray(int indx)
{
	if(indx>0 && arr[indx]!=0)
	{
		postorderArray(get_left_child(indx));
		postorderArray(get_right_child(indx));
		printf("%d\t",arr[indx]);
	}
}
void inorderArray(int indx)
{
	if(indx>0 && arr[indx]!=0)
	{
		inorderArray(get_left_child(indx));
		printf("%d\t",arr[indx]);
```

```
                inorderArray(get_right_child(indx));
        }
}
void arrayTree()
{
        int op,item,key,i;
        char op2='n';
        arr[0]=0;
        choice:
        printf("Which operation do u want to perform?\n");
        printf("1 - Insertion \t\t 2 - Deletion\t\t 3 - Traversal\n");
        scanf("%d",&op);
        switch(op)
        {
                case 1:
                        if(lvl==-1)
                        {
                                printf("Enter the item to insert as root node\n");
                                scanf("%d",&item);
                                arr[1]=item;
                                lvl++;
                        }
                        else
                        {
```

```
                printf("Enter the key after which it is to be inserted\n");
                scanf("%d",&key);
                printf("Enter the item to be insert\n");
                scanf("%d",&item);
                insertArray(key,item);
            }



            printf("Continue operations ? (y/n)");
            getchar();
            scanf("%c",&op2);
            if(op2=='y')
                goto choice;
            break;
        case 2:
            printf("Enter the item to be deleted\n");
            scanf("%d",&item);
            deleteArray(item);
            printf("Continue operations ? (y/n)");
            getchar();
            scanf("%c",&op2);
            if(op2=='y')
                goto choice;
            break;
```

```
case 3:

        traverse:

        printf("\nChoose a traversal :\n1 - Inorder\t2 - Preorder\t3 - Postorder\n");

        scanf("%d",&op);

        if(op==1)

                inorderArray(1);

        else if(op==2)

                preorderArray(1);

        else if(op==3)

                postorderArray(1);

        else

        {

                printf("Wrong choice\n");

                goto traverse;

        }

        printf("\nContinue operations ? (y/n)\n");

        getchar();

        scanf("%c",&op2);

        if(op2=='y')

                goto choice;

        break;

default:

        printf("Choose a proper option\n");

        goto choice;
```

```
	}
}
//----------------------------------------------------------------------------------------------------



node *create();
void insert(node *,node *);
void preorder(node *);
 void inorder(node *);
 void postorder(node *);
 node * Searchpar(node *,int );
void listTree()
{
	int option,val;
	char ch;
	node *root=NULL,*temp;

	do
	{
		temp=create();
		if(root==NULL)
			root=temp;
		else
			insert(root,temp);
```

```
                printf("\nDo you want to enter more(y/n)?");
                getchar();
                scanf("%c",&ch);
        }while(ch=='y'|ch=='Y');
        do
        {
        printf("1-preorder\t2:inorder\t3:postorder\t4:Deletion \n ");
        scanf("%d",&option);
        if (option==1)
        preorder(root);
        else if(option==2)
        inorder(root);
        else if(option==3)
        postorder(root);
        else
        {
           printf("Enter the element you want to delete:");
           scanf("%d",&val);
           delete_tree(root,val);
        }
  printf("\nPerform operations? (y/n)?");
                getchar();
                scanf("%c",&ch);
```

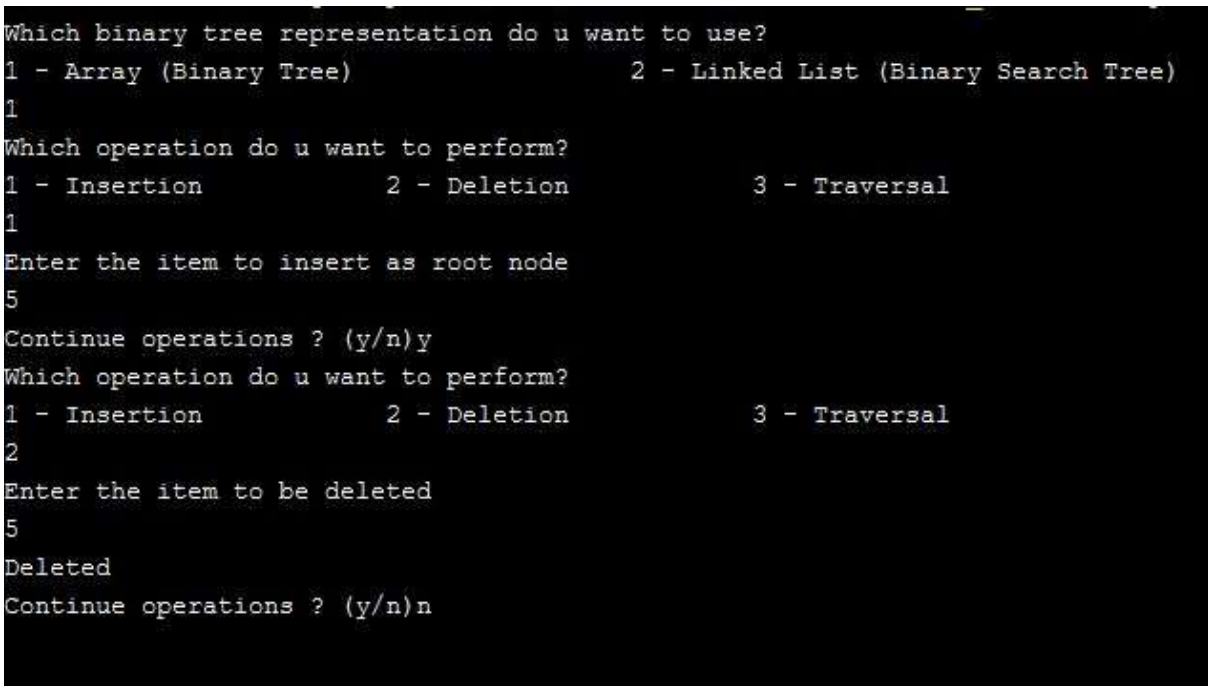
```
	}while(ch=='y'|ch=='Y');

}



node *create()

{

	node *temp;

	printf("Enter the node values for binary search tree:");

	temp=(node*)malloc(sizeof(node));

	scanf("%d",&temp->data);

	temp->left=temp->right=NULL;

	return temp;

}



void insert(node *root,node *temp)

{

	if(temp->data<root->data)

	{

		if(root->left!=NULL)

			insert(root->left,temp);

		else

			root->left=temp;

	}



	if(temp->data>root->data)
```

```
        {
                if(root->right!=NULL)
                        insert(root->right,temp);
                else
                        root->right=temp;
        }
}


node * Searchpar(node *root,int val)
{
   node *parent,*ptr1,*ptr2;
   parent=root;
   if(root->data!=val)
   {
      ptr1=root->left;
      ptr2=root->right;
      if(ptr1!=NULL)
      {
         Searchpar(ptr1,val);
      }
      else
      {
         parent=NULL;
      }
```

```
        if(ptr2!=NULL)
        {
            Searchpar(ptr2,val);
        }
        else
        {
            parent=NULL;
        }
    }
    else
    {
        return parent;
    }
}
int delete_tree(node *root,int val)
{
    node *ptr,*par,*ptr1,*ptr2;
    int flag=0;
    ptr=root;
    while(ptr!=NULL&&flag==0)
    {
        if(val<ptr->data)
        {
            par=ptr;
            ptr=ptr->left;
        }
        else if(val>ptr->data)
        {
            par=ptr;
            ptr=ptr->right;
        }
        else if(ptr->data==val)
        {
            flag=1;
        }
    }
    if(flag==0)
    {
        printf("Item doesn't exist.");
    }
    if(ptr->left==NULL&&ptr->right==NULL)
    {
        if(par->left==ptr)
        {
            par->left=NULL;
        }
        else
        {
            par->right=NULL;
        }
    }
}
void preorder(node *root)
{
            if(root!=NULL)
            {
                        printf("%d ",root->data);
                        preorder(root->left);
                        preorder(root->right);
            }
}
 void inorder(node *root)
{
            if(root!=NULL)
            {
                        inorder(root->left);
                        printf("%d ",root->data);
                        inorder(root->right);
            }
}
void postorder(node *root)
{
	if(root!=NULL)
	{
		postorder(root->left);
		postorder(root->right);
		printf("%d ",root->data);
	}
}
//-------------------------------------------------------------------------------------
void main()
{
	int op;
	choice:
	printf("Which binary tree representation do u want to use?\n");
	printf("1 - Array (Binary Tree) \t\t 2 - Linked List (Binary Search Tree)\n");
	scanf("%d",&op);
	if(op==1)
		arrayTree();
	else if(op==2)
		listTree();
	else
	{
		printf("Choose 1 or 2\n");
		goto choice;
	}
```

```
}
```

Output



```
Which binary tree representation do u want to use?
1 - Array (Binary Tree)                  2 - Linked List (Binary Search Tree)
1
Which operation do u want to perform?
1 - Insertion            2 - Deletion            3 - Traversal
1
Enter the item to insert as root node
5
Continue operations ? (y/n)y
Which operation do u want to perform?
1 - Insertion            2 - Deletion            3 - Traversal
2
Enter the item to be deleted
5
Deleted
Continue operations ? (y/n)n
```

Result: The program was executed and output obtained successfully and was verified

GRAPHS

File Name	graph.c	Compiler	online gdb compiler
Experiment No	10	Data Structure	Array
Date	4.12.2020		
Aim	To implement graphs using array using C language		

Algorithm

Algorithm DFS_LL

Input :V ,the starting vertex.

Output: A list VISIT giving the order of visit of vertices during traversal.

Data Structure: Linked structure of graph, GPTR is the pointer to the graph.

```
1. If  GPTR =NULL then
2.   Print “Graph is empty”
3.   Exit
4. EndIf
5. u=V                                   //Start from V
6. OPEN.PUSH(u)                          // Push the starting vertex into OPEN
7. While (OPEN.TOP≠NULL) do              // Till the stack is not empty
8.     u=OPEN.POP()                      //Pop the top element from OPEN
9.     If(Search_SL(VISIT,u)=FALSE)  then   // If u is not in VISIT
10.     InsertEnd_SL(VISIT,u)               // Store u in VISIT
11.     ptr = GPTR[u]                     // To push all the adjacent vertices of
                                            u into OPEN
12.    While(ptr→LINK≠NULL) do
13.       vptr=ptr→LINK
14.        OPEN.PUSH(vptr→LABEL)
15.         EndWhile
16.       EndIf
17.      EndWhile
18.      Return(VISIT)
19.    Stop.
```

Algorithm BFS

```
1.    Enqueue the starting vertex in to the queue QUEUE
2.    While QUEUE not empty
1.    Dequeue  a vertex V
2.    If Vis not in the VISIT
1.    Visit the vertex V
2.    Store V in VISIT_LIST
3.    Enqueue all adjacent vertex of V on to QUEUE
3.    EndIf
3.    EndWhile
Stop
```

Program

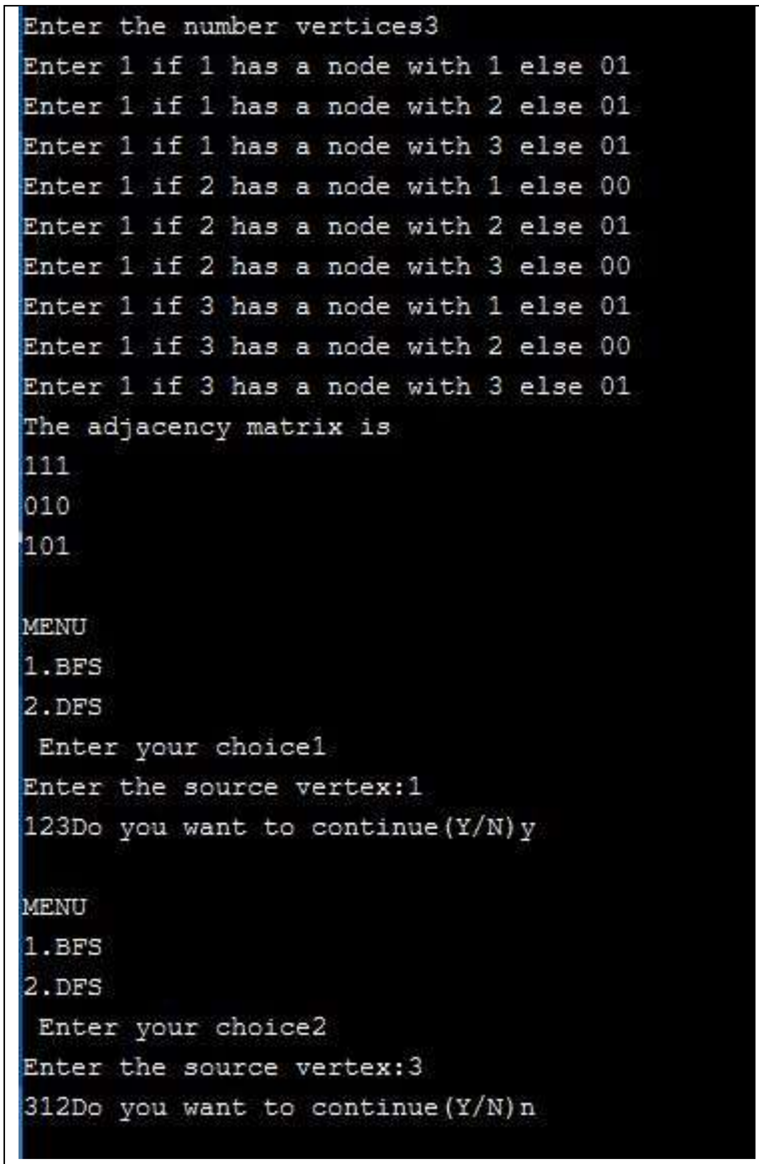
```
#include<stdio.h>
int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();
void main()
{
   int n,i,s,ch,j;
   char c,dummy;
   printf("Enter the number vertices");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
```

```
   for(j=1;j<=n;j++)
   {
      printf("Enter 1 if %d has a node with %d else 0",i,j);
      scanf("%d",&a[i][j]);
   }
}
printf("The adjacency matrix is\n");
for(i=1;i<=n;i++)
{
   for(j=1;j<=n;j++)
   {
      printf("%d",a[i][j]);
   }
   printf("\n");
}
do
{
   for(i=1;i<=n;i++)
   vis[i]=0;
   printf("\nMENU");
   printf("\n1.BFS\n2.DFS");
   printf("\n Enter your choice");
   scanf("%d",&ch);
   printf("Enter the source vertex:");
   scanf("%d",&s);
   switch(ch)
   {
      case 1:bfs(s,n);
      break;
      case 2:
            dfs(s,n);
            break;
        }
        printf("Do you want to continue(Y/N)");
        scanf("%c",&dummy);
        scanf("%c",&c);
    }while((c=='y')||(c=='Y'));
}
void bfs(int s,int n)
{
    int p,i;
    add(s);
    vis[s]=1;
    p=delete();
    if(p!=0)
    printf("%d",p);
    while(p!=0)
    {
        for(i=1;i<=n;i++)
        if((a[p][i]!=0)&&(vis[i]==0))
        {
            add(i);
            vis[i]=1;
        }
        p=delete();
        if(p!=0)
        printf("%d",p);
    }
    for(i=1;i<=n;i++)
    if(vis[i]==0)
```

```
   bfs(i,n);
}
void add(int item)
{
   if(rear==19)
   printf("Queue is full");
   else
   {
      if(rear==-1)
      {
         q[++rear]=item;
         front++;
      }
      else
      q[++rear]=item;
   }
}
int delete()
{
   int k;
   if((front>rear)||(front==-1))
   return(0);
   else
   {
      k=q[front++];
      return(k);
   }
}
void dfs(int s,int n)
{
    int i,k;
    push(s);
    vis[s]=1;
    k=pop();
    if(k!=0)
    printf("%d",k);
    while(k!=0)
    {
        for(i=1;i<=n;i++)
        if((a[k][i]!=0)&&(vis[i]==0))
        {
            push(i);
            vis[i]=1;
        }
        k=pop();
        if(k!=0)
        printf("%d",k);
    }
    for(i=1;i<=n;i++)
    if(vis[i]==0)
    dfs(i,n);
}
void push(int item)
{
    if(top==19)
    printf("Stack overflow");
    else
    stack[++top]=item;
}
int pop()
```

```
{
   int k;
   if(top==-1)
   return(0);
   else
   {
      k=stack[top--];
      return(k);
   }
}
```

Output



```
Enter the number vertices3
Enter 1 if 1 has a node with 1 else 01
Enter 1 if 1 has a node with 2 else 01
Enter 1 if 1 has a node with 3 else 01
Enter 1 if 2 has a node with 1 else 00
Enter 1 if 2 has a node with 2 else 01
Enter 1 if 2 has a node with 3 else 00
Enter 1 if 3 has a node with 1 else 01
Enter 1 if 3 has a node with 2 else 00
Enter 1 if 3 has a node with 3 else 01
The adjacency matrix is
111
010
101

MENU
1.BFS
2.DFS
 Enter your choice1
Enter the source vertex:1
123Do you want to continue(Y/N)y

MENU
1.BFS
2.DFS
 Enter your choice2
Enter the source vertex:3
312Do you want to continue(Y/N)n
```

Result: The program was executed and output obtained successfully and was verified

HASH TABLE

File Name	hashtable.c	Compiler	online gdb compiler
Experiment No	11	Data Structure	Array
Date	4.12.2020		
Aim	To implement hashing using c language.		

Algorithm

Algorithm HashLinearProbe

Input: K is the value of search. INSERT is a flag for the insertion operation.

Output: Return the location if it is found in the hash table else if INSERT is TRUE put K in the table if table has no overflown otherwise return NULL.

Data Structure: A hash table H of size HSIZE in the form of an array.

Steps:

```
1.flag=FALSE                                   //flag for continuation of looping.
   2.index=HashFunction(K)                     //Calculate the hash address using a hash function
3.If(K=H[index]) then                          //If there is a hit
4.  Return(index)
5.  Exit                                       //End of the execution
6.Else
7.      i=index+1                              //Set to the new location
8.      While(i≠ index) and (not flag) do
9.        If((H[i]=NULL) or (H[i]<0))  then    // If the cell is free
10.           If(INSERT) then                  //  True option for insertion
11.             H[i]=K                         //Put the key value into the hash table
12.             flag=TRUE
```

```
13.             EndIf
14.         Else                                         // Cell is occupied
15.             If (H[i]=K) then                         // Match
16.                 flag=TRUE
17.                 Return(i)
18.                 Exit                                  // End of the execution
19.             Else                                        //No match
20.                 i=i MOD h+1                             // Closed looping
21.             EndIf
22.         EndIf
23.     EndWhile
24.     If(flag=FALSE) and (i=index) then      //No match and reach to the starting point
25.         Print " The table is overflow"
26.     EndIf
27.   EndIf
28. Stop
```

Program

```
#include<stdio.h>
#include<stdlib.h>
struct data
{
 int key;
 int value;

};

struct data *array;

int capacity = 10;

int size = 0;

/* this function gives a unique hash code to the given key */

int hashcode(int key)

{

 return (key % capacity);

}

/* it returns prime number just greater than array capacity */

int get_prime(int n)

{

 if (n % 2 == 0)

 {

 n++;

 }

 for (; !if_prime(n); n += 2);

 return n;

}

/* to check if given input (i.e n) is prime or not */

int if_prime(int n)

{

 int i;
 if ( n == 1 || n == 0)

 {

 return 0;

 }

 for (i = 2; i < n; i++)

 {

 if (n % i == 0)

 {

 return 0;

 }

 }

 return 1;

}

void init_array()

{

 int i;

 capacity = get_prime(capacity);

 array = (struct data*) malloc(capacity * sizeof(struct data));

 for (i = 0; i < capacity; i++)

 {

 array[i].key = 0;

 array[i].value = 0;

 }

}
```

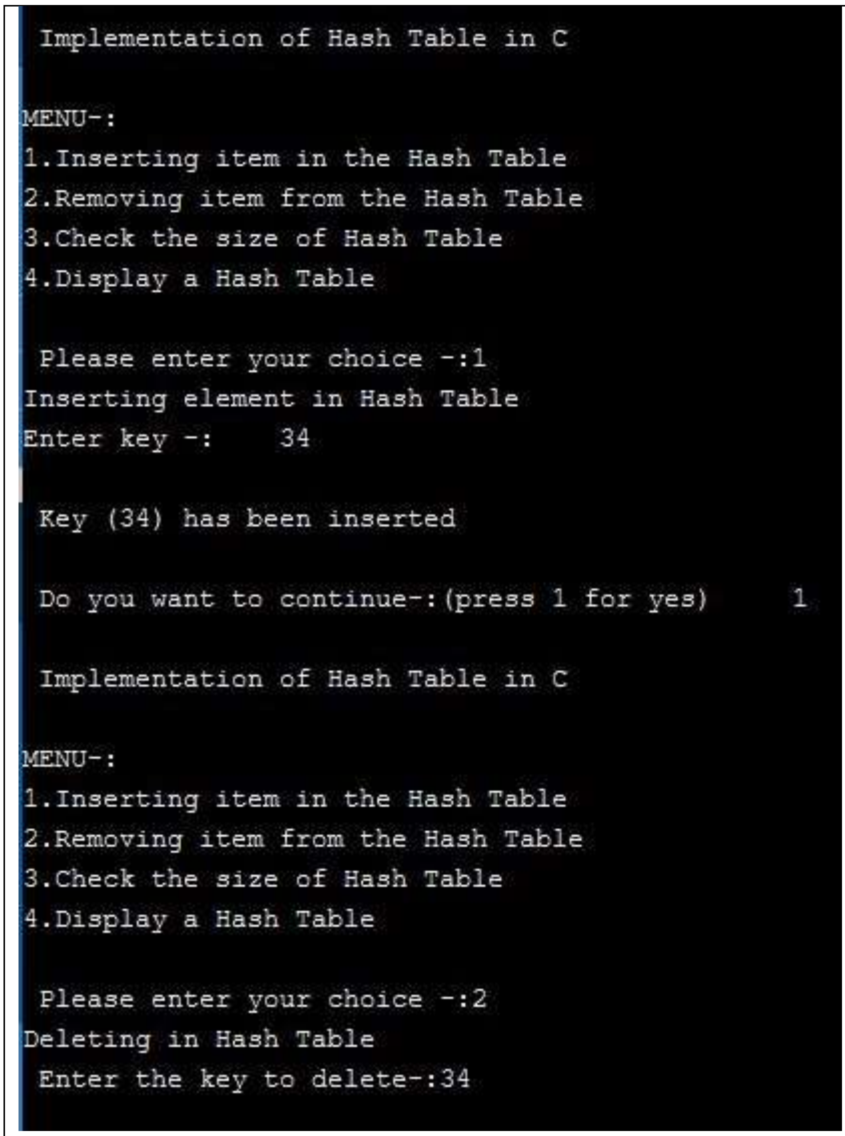
```
/* to insert a key in the hash table */

void insert(int key)

{

 int index = hashcode(key);

 if (array[index].value == 0)

 {

 /* key not present, insert it */

 array[index].key = key;

 array[index].value = 1;

 size++;

 printf("\n Key (%d) has been inserted \n", key);

 }

 else if(array[index].key == key)

 {

 /* updating already existing key */

 printf("\n Key (%d) already present, hence updating its value \n", key);

 array[index].value += 1;

 }

 else

 {

 /* key cannot be insert as the index is already containing some other key */

 printf("\n ELEMENT CANNOT BE INSERTED \n");

 }

}
```

```
/* to remove a key from hash table */

void remove_element(int key)

{

 int index = hashcode(key);

 if(array[index].value == 0)

 {

 printf("\n This key does not exist \n");

 }

 else {

 array[index].key = 0;

 array[index].value = 0;

 size--;

 printf("\n Key (%d) has been removed \n", key);

 }

}

/* to display all the elements of a hash table */

void display()

{

 int i;

 for (i = 0; i < capacity; i++)

 {

 if (array[i].value == 0)

 {

 printf("\n Array[%d] has no elements \n",i);
 }
 else
 {
 printf("\n array[%d] has elements -:\n key(%d) and value(%d) \t", i, array[i].key,
array[i].value);
 }
 }
}
int size_of_hashtable()
{
 return size;
}
void main()
{
 int choice, key, value, n, c;
 init_array();
 do
 {
 c=0;
 printf("\n Implementation of Hash Table in C \n\n");
 printf("MENU-: \n1.Inserting item in the Hash Table"
 "\n2.Removing item from the Hash Table"
 "\n3.Check the size of Hash Table"
 "\n4.Display a Hash Table"
```

```
"\n\n Please enter your choice -:");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Inserting element in Hash Table\n");

printf("Enter key -:\t");

scanf("%d", &key);

insert(key);

break;

case 2:

printf("Deleting in Hash Table \n Enter the key to delete-:");

scanf("%d", &key);

remove_element(key);

break;

case 3:

n = size_of_hashtable();

printf("Size of Hash Table is-:%d\n", n);

break;

case 4:

display();

break;

default:

printf("Wrong Input\n");
```

```
 }

 printf("\n Do you want to continue-:(press 1 for yes)\t");

 scanf("%d", &c);

 }

 while(c == 1);

 getch();

}
```

Output



```
 Implementation of Hash Table in C

MENU-:
1.Inserting item in the Hash Table
2.Removing item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice -:1
Inserting element in Hash Table
Enter key -:    34

 Key (34) has been inserted

 Do you want to continue-:(press 1 for yes)     1

 Implementation of Hash Table in C

MENU-:
1.Inserting item in the Hash Table
2.Removing item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice -:2
Deleting in Hash Table
 Enter the key to delete-:34
```

Result: The program was executed and output obtained successfully and was verified

HEAP SORT

File Name	heapsort.c	Compiler	online gdb compiler
Experiment No	12	Data Structure	Array
Date	11.12.2020		
Aim	To implement heapsort and find and element using binary search techinque using c language.		

Algorithm

```
Heapify(A, i)
    l ← left(i)
    r ← right(i)
    if l <= heapsize[A] and A[l] > A[i]
       then largest ←l
       else largest ← i
    if r <= heapsize[A] and A[r] > A[largest]
       then largest ← r
    if largest != i
       then swap A[i] ←→ A[largest]
          Heapify(A, largest)
 Buildheap(A)
    heapsize[A] ←length[A]
    for i ←|length[A]/2  //down to 1
       do Heapify(A, i)
 Heapsort(A)
    Buildheap(A)
    for i ← length[A] //down to 2
       do swap A[1] ←→ A[i]
       heapsize[A] ← heapsize[A] - 1
       Heapify(A, 1)
```

BinarySearch

Input: Array with size , N and search element namely key

Output: Return location of search element if found else return -1

Data Structure: Array.

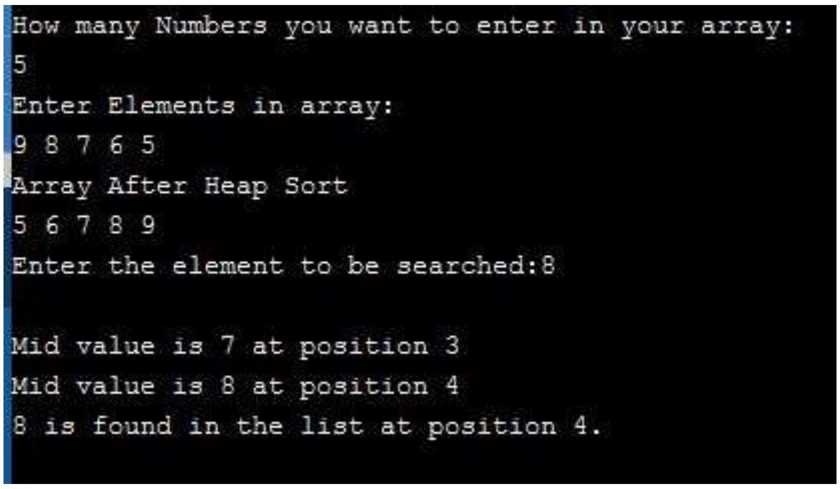
```
Binary_search (A, N, key)

1.Set left=0, right=N-1

2.Repeat steps 3 and 4 while left<=right

3.mid= (left + right)/ 2

4.If A[mid] < key then

            set left=mid+1

  else if A[mid] > key

            set right=mid-1

       else

            return mid //found at location-mid

   end while

6.Return  //not found
```

Program

```
#include<stdio.h>
#include <conio.h>
void heapify_function(int arr[])
{
    int i,n;
    n=arr[0];
    for(i=n/2;i>=1;i--)
    adjust(arr,i);
}
```

```
void adjust(int arr[],int i)
{
   int j,temp,n,k=1;
   n=arr[0];
   int l=2*i;
   int r=2*i+1;
   while(2*i<=n && k==1)
   {
     j=2*i;
     if(j+1<=n && arr[j+1] > arr[j])
        j=j+1;

     if( arr[j] < arr[i])
        k=0;
     else
     {
        temp=arr[i];
        arr[i]=arr[j];
        arr[j]=temp;
        i=j;
      }
    }
}

void main()
{
int arr[100],n,temp,last,i,key,mid,left,right;
printf("How many Numbers you want to enter in your array: \n");
scanf("%d",&n);
printf("Enter Elements in array:\n");
for(i=1;i<=n;i++)
scanf("%d",&arr[i]);
arr[0]=n;
heapify_function(arr);
while(arr[0] > 1)
{
   last=arr[0];
   temp=arr[1];
   arr[1]=arr[last];
   arr[last]=temp;
   arr[0]--;
   adjust(arr,1);
}

printf("Array After Heap Sort\n");
for(i=1;i<=n;i++)
printf("%d ",arr[i]);
getch();
   printf("\nEnter the element to be searched:");
   scanf("%d",&key);
   left=1;
   right=n;
   while(left<=right)
   {
      mid=(left+right)/2;
      printf("\nMid value is %d at position %d",arr[mid],mid);
      if(arr[mid]<key)
      {
          left=mid+1;
      }
      else if(arr[mid]>key)
      {
          right=mid-1;
      }
      else
      {
          printf("\n%d is found in the list at position %d.",key,mid);
          break;
      }
      mid=(left+right)/2;
   }
   if(left>right)
   {
      printf("\n%d is not found in the list.");
   }
}
```

Output



```
How many Numbers you want to enter in your array:
5
Enter Elements in array:
9 8 7 6 5
Array After Heap Sort
5 6 7 8 9
Enter the element to be searched:8

Mid value is 7 at position 3
Mid value is 8 at position 4
8 is found in the list at position 4.
```

Result: The program was executed and output obtained successfully and was verified

MEMORY MANAGEMENT

File Name	memory.c	Compiler	online gdb compiler
Experiment No	13	Data Structure	Array
Date	18.12.2020		
Aim	To implement first fit, best fit and worst fit memory management using c language.		

Algorithm

Input:Let nb be the size of blocks and nf be the size of files.

Output:Files are allocated to the memory according to its size.

FIRST FIT

```
1. Repeat the steps for(i=1;i<=nf;i++)
 1.1.Repeat the steps for j<=nb
 1.1.1. if(bf[j]!=1) then
 temp=b[j]-f[i]
 if(temp>=0) then
 ff[i]=j
 1.2. frag[i]=temp
 1.3. bf[ff[i]]=1
```

BEST FIT

```
 1.Repeat the steps for(i=1;i<=nf;i++)
 1.1.Repeat the steps for(j=1;j<=nb;j++)
 1.1.1 if(bf[j]!=1) then
 1.1.1.1 temp=b[j]-f[i]
 1.1.1.2 if(temp>=0) then
 1.1.1.3 if(lowest>temp) then
 1.1.1.4ff[i]=j

 1.1.1.5lowest=temp
 1.2 frag[i]=lowest
 1.3bf[ff[i]]=1
 1.4lowest=10000
```

WORST FIT

```
1.Repeat the steps for(i=1;i<=nf;i++)
```

```
1.1.Repeat the steps for(j=1;j<=nb;j++)
1.1.1 if(bf[j]!=1) then //if bf[j] is not allocated
1.1.1.1.temp=b[j]-f[i]
1.1.1.2. if(temp>=0) then
1.1.1.3.if(highest<temp) then
a. ff[i]=j
b. highest=temp
1.2. frag[i]=highest
1.3.bf[ff[i]]=1
1.4.highest=0
```

Program

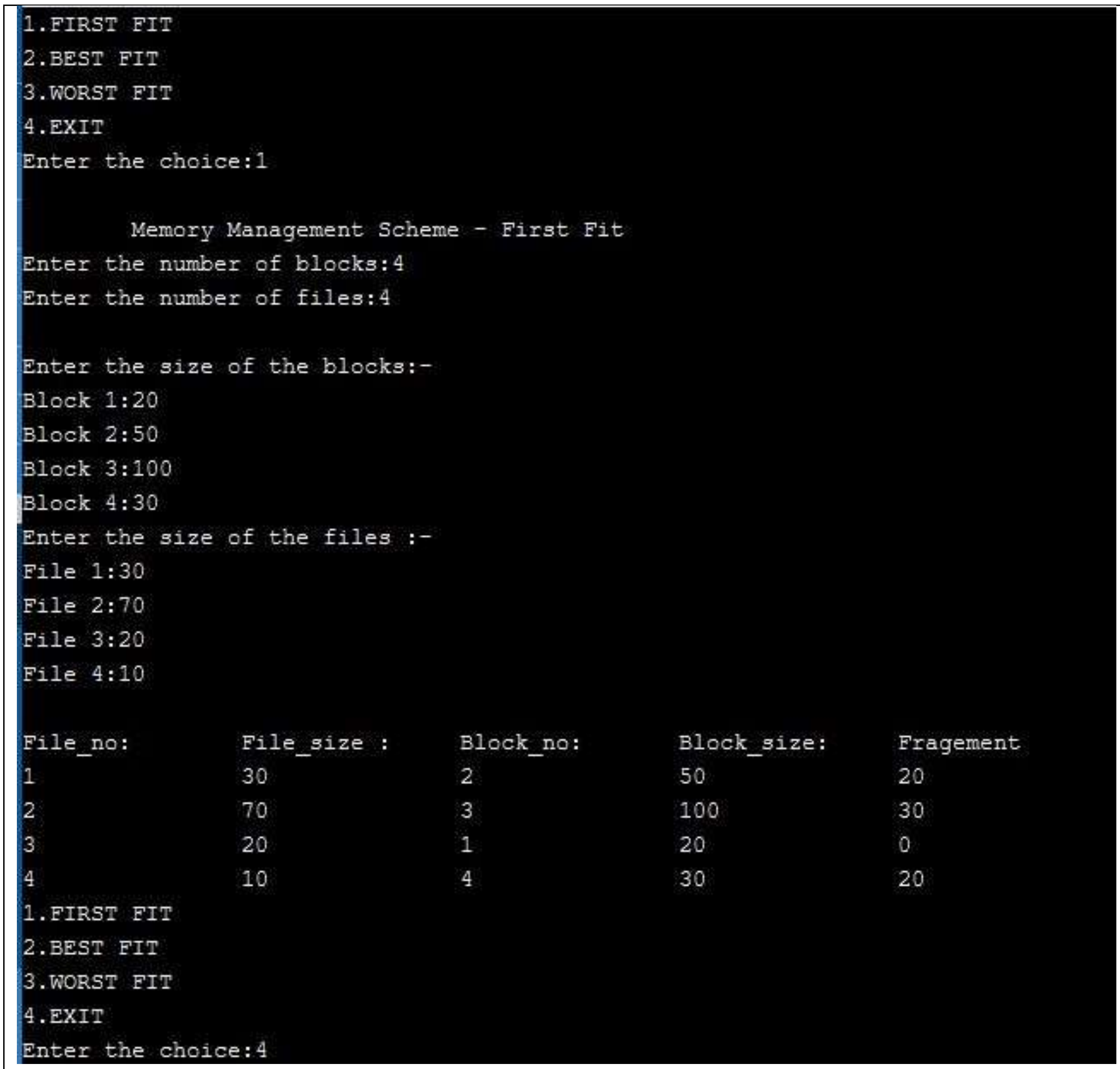
```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
 int ch;
 int frag[max],b[max],f[max],i,j,nb,nf,lowest=10000,highest=0,temp;
 static int bf[max],ff[max];
 do{
 printf("\n1.FIRST FIT\n2.BEST FIT\n3.WORST FIT\n4.EXIT");
 printf("\nEnter the choice:");
 scanf("%d",&ch);

 switch(ch)
 {
 case 1:
 {
 printf("\n\tMemory Management Scheme - First Fit");
 printf("\nEnter the number of blocks:");
 scanf("%d",&nb);
 printf("Enter the number of files:");
 scanf("%d",&nf);
 printf("\nEnter the size of the blocks:-\n");
```

```
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
//getch();
break;
}
case 2:
{
```

```
printf("\n\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;

lowest=temp;
}
}
}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
```

```
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
break;
}
case 3:
{
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
 ff[i]=j;
 highest=temp;
 }
 }
 }
 frag[i]=highest;
 bf[ff[i]]=1;
 highest=0;
 }
 printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
 for(i=1;i<=nf;i++)
 printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
 getch();
 break;
 }
 }
 }
 while(ch!=4);
 {
 return 0;
 }
}
```

Output



```
1.FIRST FIT
2.BEST FIT
3.WORST FIT
4.EXIT
Enter the choice:1

        Memory Management Scheme - First Fit
Enter the number of blocks:4
Enter the number of files:4

Enter the size of the blocks:-
Block 1:20
Block 2:50
Block 3:100
Block 4:30
Enter the size of the files :-
File 1:30
File 2:70
File 3:20
File 4:10

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               30              2               50              20
2               70              3               100             30
3               20              1               20              0
4               10              4               30              20
1.FIRST FIT
2.BEST FIT
3.WORST FIT
4.EXIT
Enter the choice:4
```

Result: The program was executed and output obtained successfully and was verified