PRACTICAL NO. : 1

/*
To perform menu-driven code for searching techniques : -
1. Linear search
2. Binary search */

```
#include<stdio.h>
#include<conio.h>
int LINEAR_SEARCH(int k[20],int X,int N);
int BINARY_SEARCH(int k[2],int X,int N);
void line();
void dline();
void main()
{
 int i,k[20],X,x,N,index,ch;
 char c;
 do
 {
        clrscr();
        printf("\n");
        dline();
        printf("\n\t\t\t\tPRACTICAL NO.:1\n");
        dline();
        printf("\n\n\n\n\n\n");
        line();
        printf("\n\t\t\t\tSEARCHING TECHNIQUES\n");
        line();
        printf("\n\t\t\t\t1.Linear search");
        printf("\n\t\t\t\t2.Binary search");
        printf("\n\t\t\t\t3.Exit\n");
        line();
        printf("\t\t\t\tChoice :");
        scanf("%d",&ch);
        clrscr();
        switch(ch)
        {
         case 1: printf("\n\nLinear Search\n");
                        line();
                        printf("\n\nEnter size of array :");
```

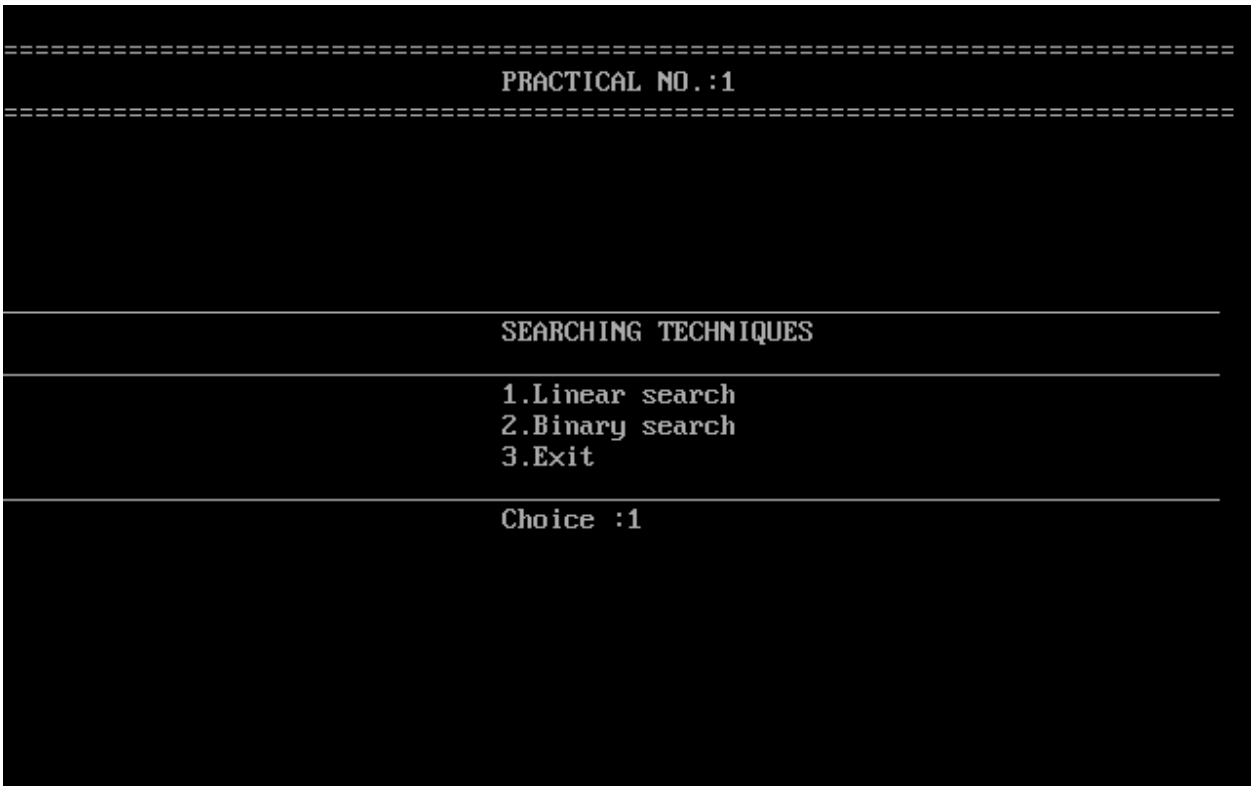
```
                scanf("%d",&N);
                printf("\nEnter elements...");
                for(i=1;i<=N;++i)
                { scanf("%d",&k[i]); }
                clrscr();
                printf("\n\nLinear Search\n");
                line();
                printf("\n");
                for(i=1;i<=N;++i)
                { printf("%d   ",k[i]); }
                printf("\n\nEnter element to be search...");
                scanf("%d",&X);
                index=LINEAR_SEARCH(k,X,N);
                if(index==0)
                        printf("\nElement not found.");
                else
                        printf("\nElement found at index :%d",index);
                printf("\n\n\n\n\n");
                break;
case 2: printf("\n\nBinary search\n");
                line();
                printf("\n\nEnter size of array :");
                scanf("%d",&N);
                printf("\nEnter elements...");
                for(i=1;i<=N;++i)
                { scanf("%d",&k[i]); }
                clrscr();
                printf("\n\nBinary search\n");
                line();
                printf("\n");
                for(i=1;i<=N;++i)
                { printf("%d   ",k[i]); }
                printf("\n\nEnter element to be search...");
                scanf("%d",&x);
                index=BINARY_SEARCH(k,x,N);
                if(index==0)
                        printf("\nElement not found.");
                else
                        printf("\nElement found at index :%d",index);
                printf("\n\n\n\n\n");
```

```
                        break;
        case 3: exit(0);
                        break;
        default:line();
                        printf("\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t Wrong
Entry...\n\n\n\n\n\n");
                        // printf("\n\n\n\n\n\n");
        }
        line();
        printf("\nReturn back(y) / Exit(Y)...");
        fflush(stdin);
        scanf("%c",&c);
 }while(c!='Y');
 getch();

}
```

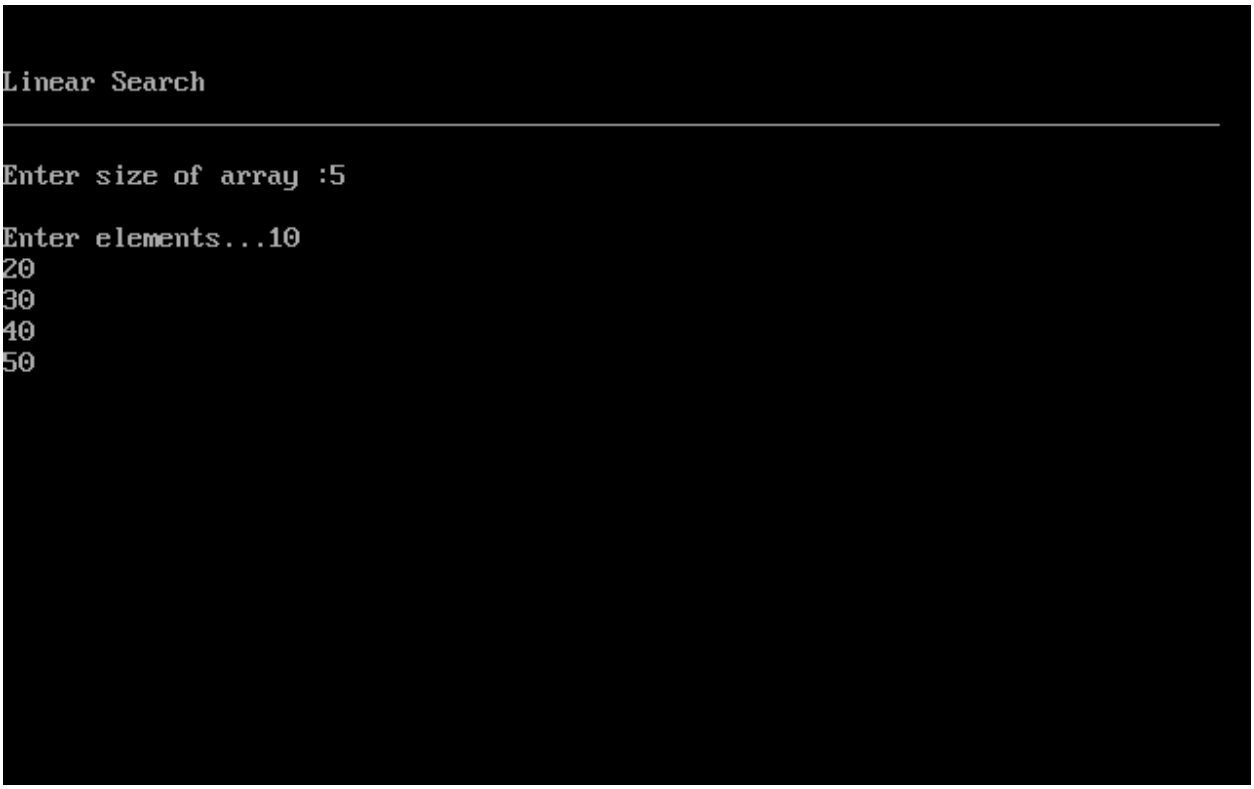
```
int LINEAR_SEARCH(int k[20],int X,int N)
{
 int I=1;
 k[N+1]=X;
 while(k[I]!=X)
 {
  I+=1;
 }
 if(I==N+1)
        {
                printf("\nUnsuccessful Search...");
                return(0);
        }
 else
        {
                printf("\nSuccessful Search...");
                return(I);
```

```
        }
}



int BINARY_SEARCH(int k[20],int X,int N)
{
 int Low=1,High=N,Mid;
 while(Low<=High)
 {
        Mid=(Low+High)/2;
        if(X<k[Mid])
                High=Mid-1;
        else
                {
                        if(X>k[Mid])
                                Low=Mid+1;
                        else
                                {
                                        printf("\nSuccessfu search...");
                                        return(Mid);
                                }
                }
 }
 printf("\nUnsuccessful search...");
 return(0);
}

void line()
{
        printf("_____");
}

void dline()
{
        printf("==========
==========");
}
```

OUTPUT : -


PRACTICAL NO.:1
SEARCHING TECHNIQUES
1.Linear search
2.Binary search
3.Exit
Choice :1

(1)


Linear Search
Enter size of array :5
Enter elements...10
20
30
40
50

(2)


Linear Search
10 20 30 40 50
Enter element to be search...1
Unsuccessful Search...
Element not found.
Return back(y) / Exit(Y)...y

(3)

```
================================================================================
                                PRACTICAL NO.:1
================================================================================




--------------------------------------------------------------------------------
                                SEARCHING TECHNIQUES
--------------------------------------------------------------------------------
                                1.Linear search
                                2.Binary search
                                3.Exit
--------------------------------------------------------------------------------
                                Choice :2_
```

(4)

```
Binary search
--------------------------------------------------------------------------------

Enter size of array :10

Enter elements...10
20
30
40
50
60
70
80
90
100
```

(5)

```
Binary search
--------------------------------------------------------------------------------
10   20   30   40   50   60   70   80   90   100

Enter element to be search...11

Unsuccessful search...
Element not found.




--------------------------------------------------------------------------------
Return back(y) / Exit(Y)...y_
```

(6)

```
================================================================================
                              PRACTICAL NO.:1
================================================================================




--------------------------------------------------------------------------------
                              SEARCHING TECHNIQUES
--------------------------------------------------------------------------------
                              1.Linear search
                              2.Binary search
                              3.Exit
--------------------------------------------------------------------------------
                              Choice :4 _
```

(7)

PRACTICAL NO. : 2

/* To perform menu-driven code for sorting techniques : -
1. Bubble sort
2. Selection sort***/**

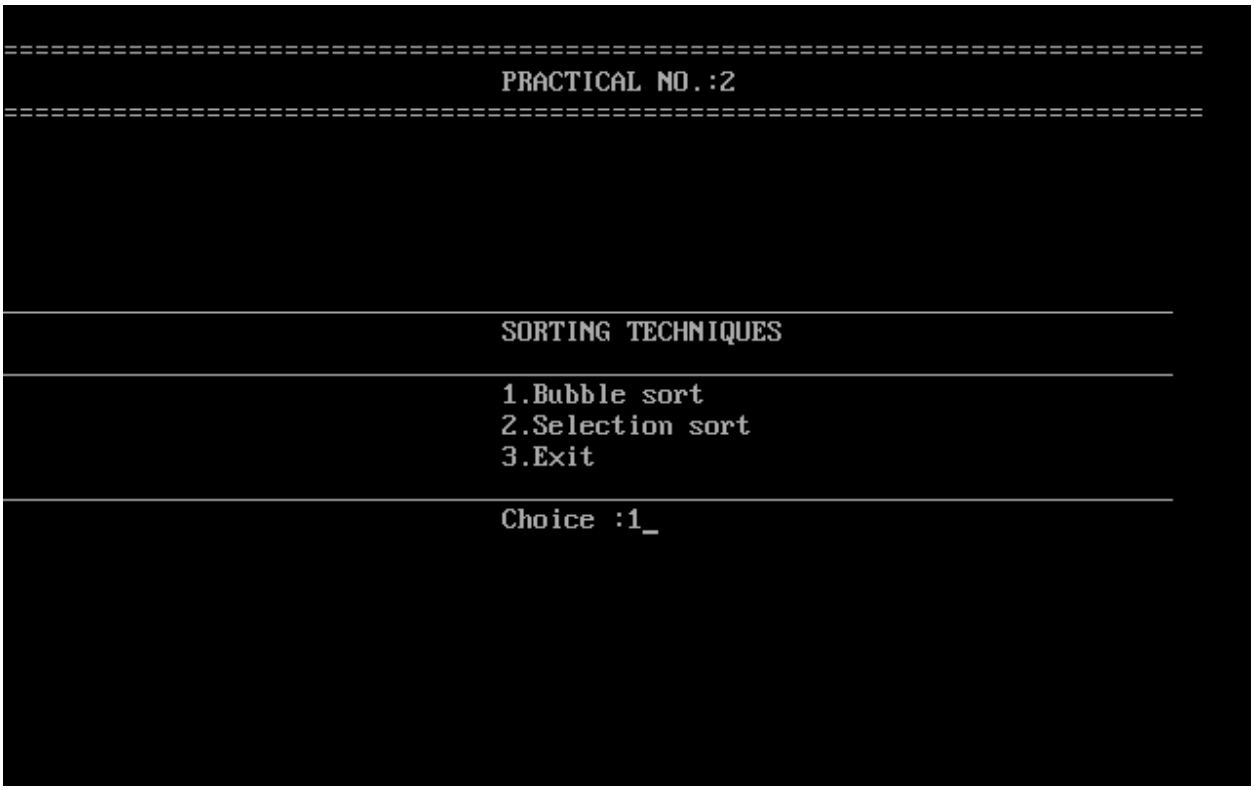
```
#include<stdio.h>
#include<conio.h>
void line();
void dline();
int BUBBLE_SORT(int k[20],int N);
int SELECTION_SORT(int k[20],int N);
void main()
{
 int i,k[20],N,ch;
 char c;
 do
 {
            clrscr();
            printf("\n");
            dline();
            printf("\n\t\t\t\tPRACTICAL NO.:2\n");
            dline();
            printf("\n\n\n\n\n\n");
            line();
            printf("\n\t\t\t\tSORTING TECHNIQUES\n");
            line();
            printf("\n\t\t\t\t1.Bubble sort");
            printf("\n\t\t\t\t2.Selection sort");
            printf("\n\t\t\t\t3.Exit\n");
            line();
            printf("\t\t\t\tChoice :");
            scanf("%d",&ch);
            clrscr();
            switch(ch)
            {
             case 1: printf("\n\nBubble sort\n");
```

```
                line();
                printf("\n\nEnter size of array :");
                scanf("%d",&N);
                printf("\nEnter elements...");
                for(i=1;i<=N;++i)
                { scanf("%d",&k[i]); }
                clrscr();
                printf("\n\nBubble sort\n");
                line();
                printf("\n");
                for(i=1;i<=N;++i)
                { printf("%d   ",k[i]); }
                BUBBLE_SORT(k,N);
                printf("\n\nAfter sorting...\n");
                for(i=1;i<=N;++i)
                { printf("%d   ",k[i]); }
                printf("\n\n\n\n\n");
                break;
case 2: printf("\n\nSelection sort\n");
                line();
                printf("\n\nEnter size of array :");
                scanf("%d",&N);
                printf("\nEnter elements...");
                for(i=1;i<=N;++i)
                { scanf("%d",&k[i]); }
                clrscr();
                printf("\n\nSelection sort\n");
                line();
                printf("\n");
                for(i=1;i<=N;++i)
                { printf("%d   ",k[i]); }
                SELECTION_SORT(k,N);
                printf("\n\nAfter sorting...\n");
                for(i=1;i<=N;++i)
                { printf("%d   ",k[i]); }
                printf("\n\n\n\n\n");
                break;
case 3: exit(0);
```

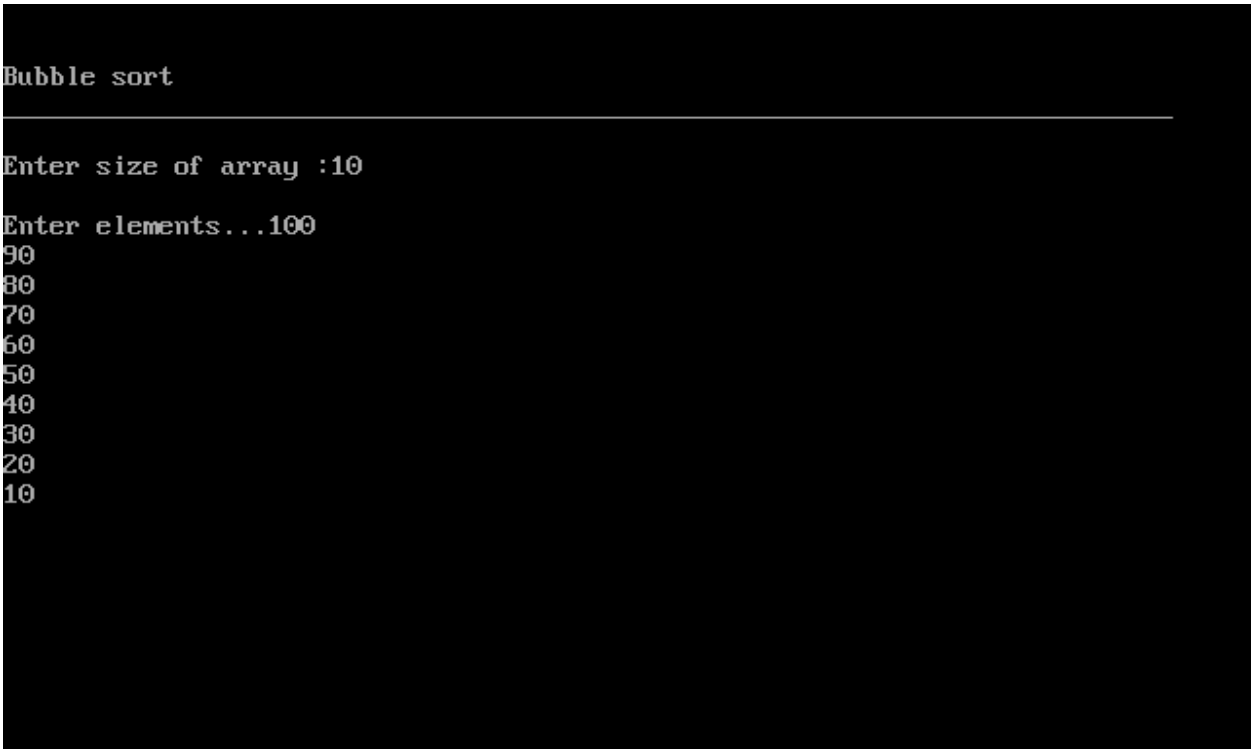
```
                        break;
            default:line();
                        printf("\n\n\n\n\n\n\n\n\n\n\t\t\t\t Wrong Entry...\n\n\n\n\n");
                        // printf("\n\n\n\n\n");
            }
            line();
            printf("\nReturn back(y) / Exit(Y)...");
            fflush(stdin);
            scanf("%c",&c);
 }while(c!='Y');
 getch();

}

int BUBBLE_SORT(int k[20],int N)
{
 int Last,Pass,Exchs,i,tmp;
 Last=N;
 for(Pass=1;Pass<=N-1;++Pass)
 {
        Exchs=0;
        for(i=1;i<=Last-1;++i)
        {
                if(k[i]>k[i+1])
                {
                        tmp=k[i];
                        k[i]=k[i+1];
                        k[i+1]=tmp;
                        Exchs+=1;
                }
        }
        if(Exchs==0)
                return(0);
        else
                Last-=1;
 }
 return(0);
}
```

```
int SELECTION_SORT(int k[20],int N)
{
 int PASS,MIN_INDEX,I,t;
 for(PASS=1;PASS<=N-1;++PASS)
 {
        MIN_INDEX=PASS;
        for(I=PASS+1;I<=N;++I)
        {
                if(k[I]<k[MIN_INDEX])
                        MIN_INDEX=I;
                if(MIN_INDEX!=PASS)
                {
                        t=k[PASS];
                        k[PASS]=k[MIN_INDEX];
                        k[MIN_INDEX]=t;
                }
        }
 }
 return(0);
}

void line()
{

printf("_____
___");
}

void dline()
{

printf("==========================================================================
=====");
}
```

OUTPUT : -


PRACTICAL NO.:2
SORTING TECHNIQUES
1.Bubble sort
2.Selection sort
3.Exit
Choice :1_

(1)


Bubble sort
Enter size of array :10
Enter elements...100
90
80
70
60
50
40
30
20
10

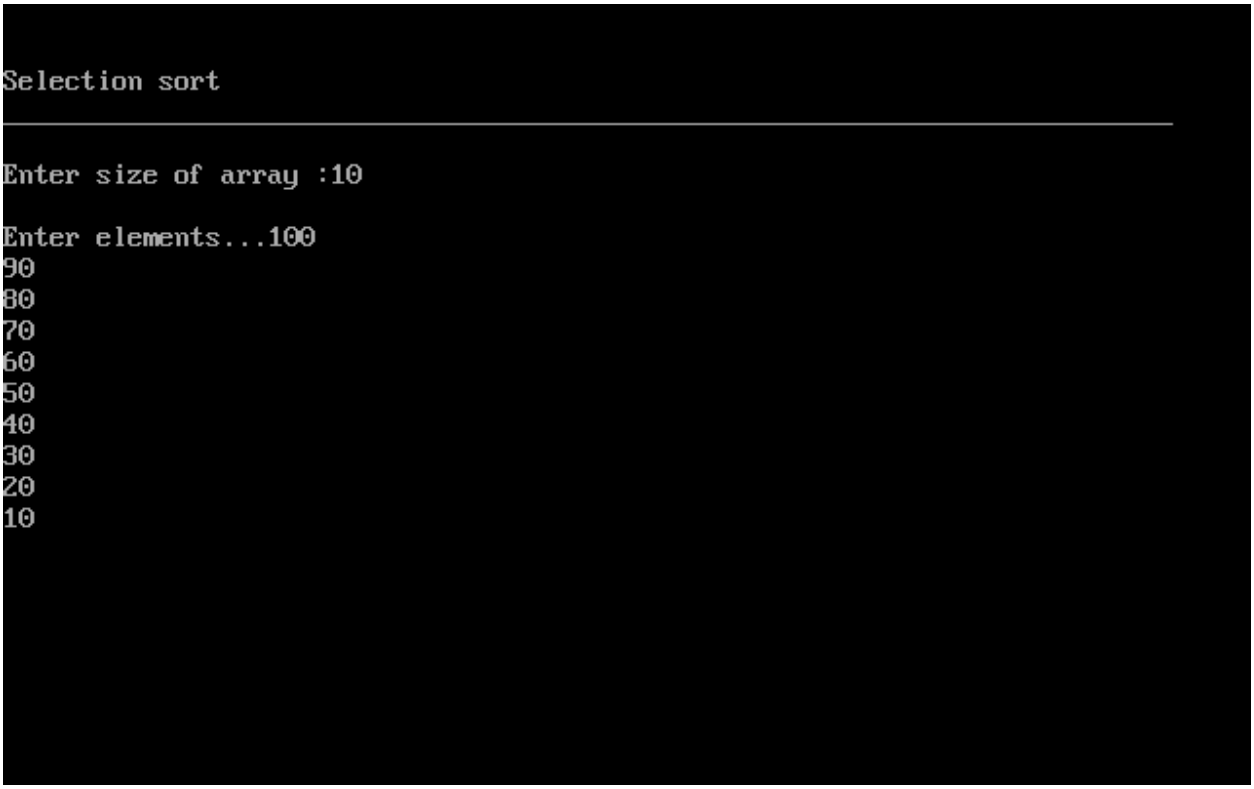
(2)

```
Bubble sort
--------------------------------------------------------------------------------
100   90   80   70   60   50   40   30   20   10

After sorting...
10   20   30   40   50   60   70   80   90   100




--------------------------------------------------------------------------------
Return back(y) / Exit(Y)...y
```

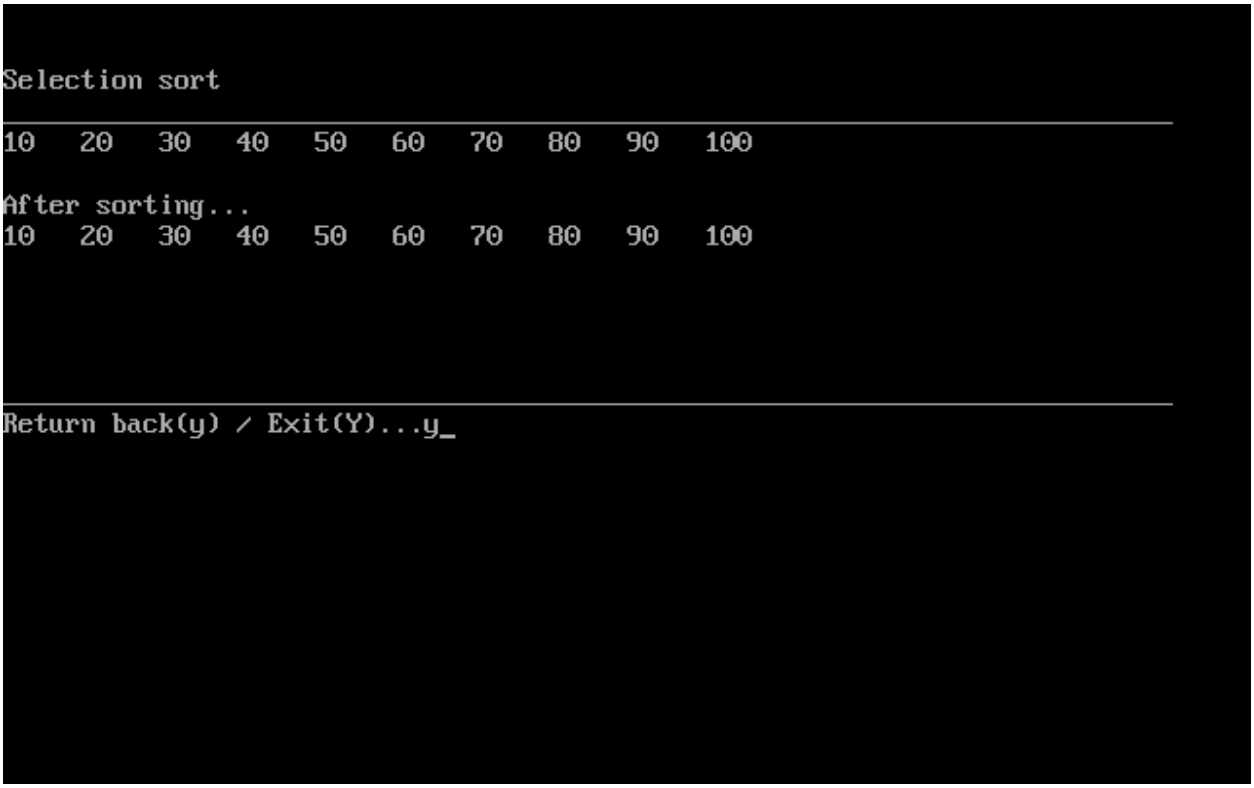
(3)

```
================================================================================
                                PRACTICAL NO.:2
================================================================================




--------------------------------------------------------------------------------
                                SORTING TECHNIQUES
--------------------------------------------------------------------------------
                                1.Bubble sort
                                2.Selection sort
                                3.Exit
--------------------------------------------------------------------------------
                                Choice :2_
```

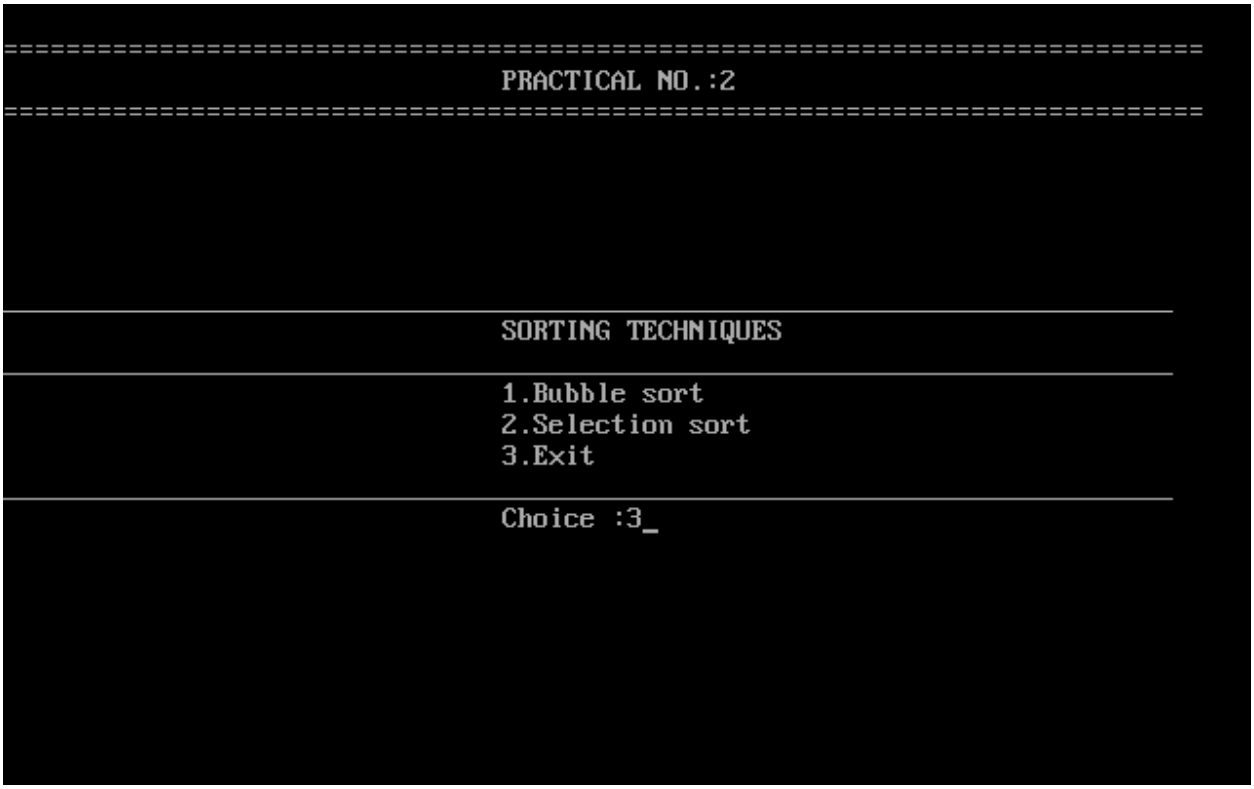
(4)


Selection sort
Enter size of array :10
Enter elements...100
90
80
70
60
50
40
30
20
10

(5)


Selection sort
10 20 30 40 50 60 70 80 90 100
After sorting...
10 20 30 40 50 60 70 80 90 100
Return back(y) / Exit(Y)...y

(6)


PRACTICAL NO.:2
SORTING TECHNIQUES
1.Bubble sort
2.Selection sort
3.Exit
Choice :3_

(7)

PRACTICAL NO. : 3

/* To perform menu-driven code for sorting techniques : -
1. Merge sort
2. Quick sort***/**

```
#include <stdio.h>
#include <stdlib.h>
void QuickSort(int A[], int low, int high);
int partition(int A[], int low, int high);
void mergeSort(int A[], int low, int high);
void dline()
{
 int i;
 for(i=1;i<=80;++i)
        printf("=");
}
void line()
{
 int i;
 for(i=1;i<=80;++i)
        printf("_");
}

void main()
{
        int size,i, A[50], ch;
        char c;
        do
        {
                clrscr();
```

```
printf("\n");
dline();
printf("\t\t\t\tPRACTICAL NO.:3\n");
dline();
printf("\n\n\n\n");
line();
printf("\t\t\t\tSorting Techniques\n ");
line();
printf("\n\t\t\t\t\t1.Quick sort");
printf("\n\t\t\t\t\t2.Merge sort");
printf("\n\t\t\t\t\t3.Exit\n");
line();
printf("\n\n\t\t\t\t\tChoice : ");
scanf("%d", &ch);
clrscr();
switch(ch)
{
        case 1: printf("\nEnter size of array :");
                        scanf("%d", &size);
                        printf("\nEnter elements...");
                        for(i = 0; i < size; i++)
                        {
                                scanf("%d", &A[i]);
                        }
                        QuickSort(A, 0, size-1);
                        break;
        case 2: printf("\nEnter size of array :");
                        scanf("%d", &size);
                        printf("\nEnter elements...");
                        for(i = 0; i < size; i++)
                        {
                                scanf("%d", &A[i]);
                        }
```

```
                        mergeSort(A, 0, size - 1);
                        break;
                case 3: exit(0);
                default:printf("\nInvalid Entry");
            }

            printf("\nYour Array is:\n");
            for (i = 0; i < size; i++)
            {
                printf("%d ",A[i]);
            }

            printf("\nReturn back(y) / Exit(Y)...");fflush(stdin);
            scanf("%c", &c);
        } while(c!='Y');
        getch();
}

void QuickSort(int A[], int low, int high)
{
        int keyloc;
        if (low < high)
        {
            keyloc = partition(A, low, high);
            QuickSort(A, low, keyloc - 1);
            QuickSort(A, keyloc + 1, high);
        }

}

int partition(int A[], int low, int high)
{
        int i, j, key, temp;
```

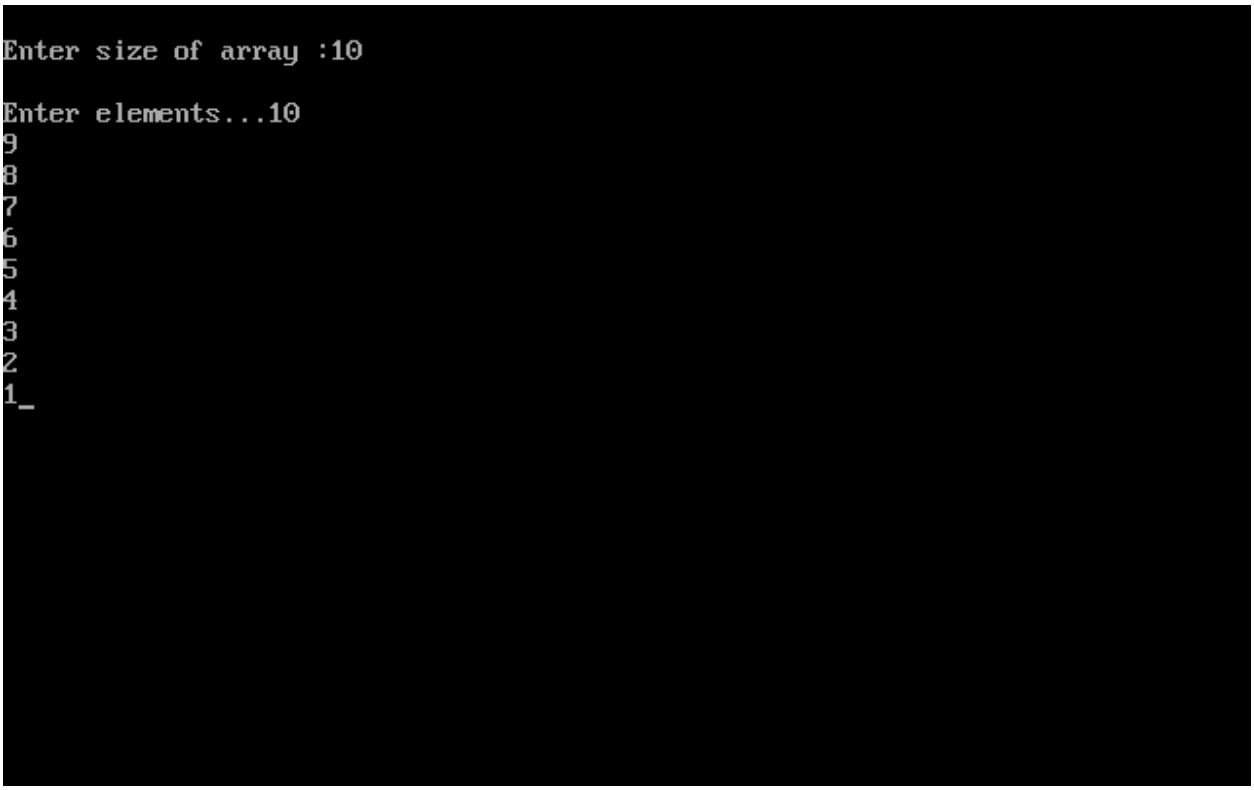
```
        key = A[low];
        i = low + 1;
        j = high;

        do
        {
                while (A[i] <= key)
                i++;

                while (A[j] > key)
                j--;

                if (i < j)
                {
                        temp = A[i];
                        A[i] = A[j];
                        A[j] = temp;
                }


        } while (i < j);

        temp = A[low];
        A[low] = A[j];
        A[j] = temp;

        return j;
}

void merge(int A[], int mid, int low, int high)
{
        int i, j, k, B[100];
        i = low;
```

```
j = mid + 1;
k = low;

while (i <= mid && j <= high)
{
        if (A[i] < A[j])
        {
                B[k] = A[i];
                i++;
                k++;
        }
        else
        {
                B[k] = A[j];
                j++;
                k++;
        }
}
while (i <= mid)
{
        B[k] = A[i];
        k++;
        i++;
}
while (j <= high)
{
        B[k] = A[j];
        k++;
        j++;
}
for (i = low; i <= high; i++)
{
        A[i] = B[i];
```

```
        }

}

void mergeSort(int A[], int low, int high){
        int mid;
        if(low<high)
        {
                mid = low + (high - low) /2;
                mergeSort(A, low, mid);
                mergeSort(A, mid+1, high);
                merge(A, mid, low, high);
        }
}
```

OUTPUT : -

(1)



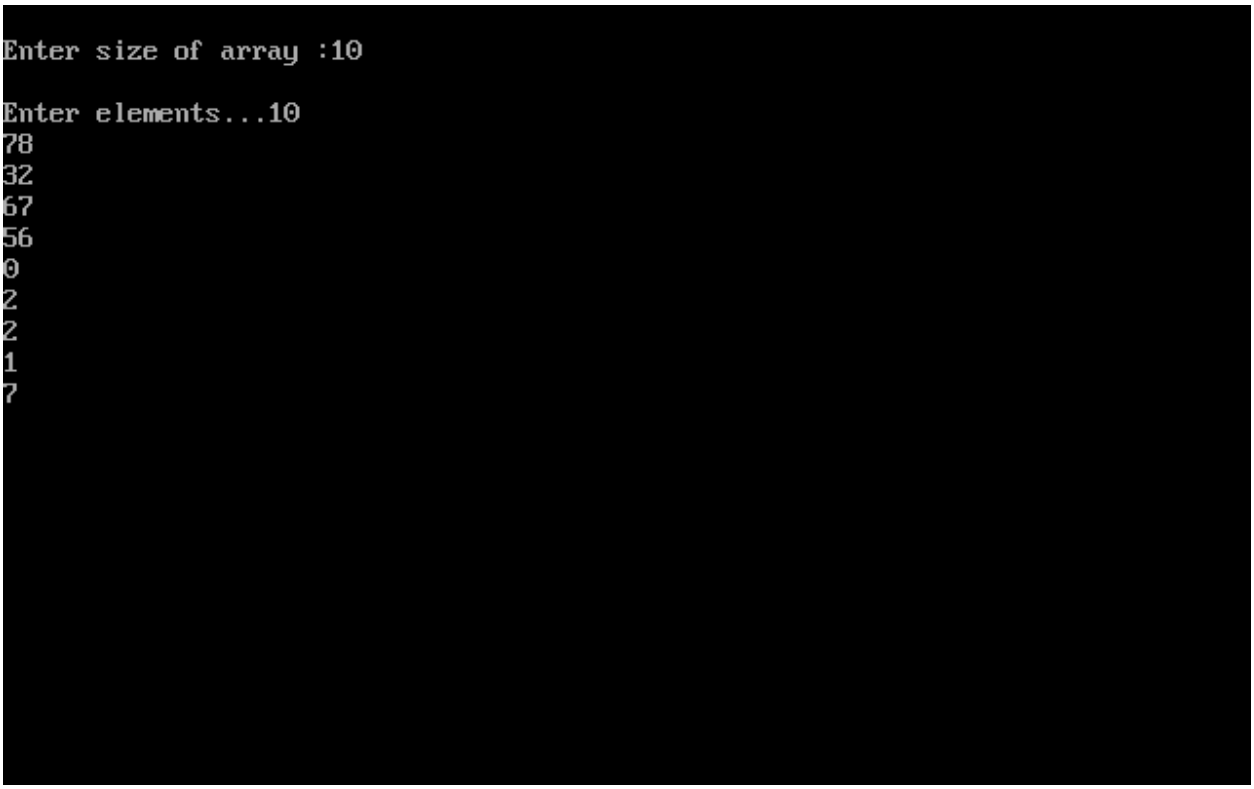
(2)

```
Enter size of array :10

Enter elements...10
9
8
7
6
5
4
3
2
1

Your Array is:
1 2 3 4 5 6 7 8 9 10
Return back(y) / Exit(Y)...y_
```

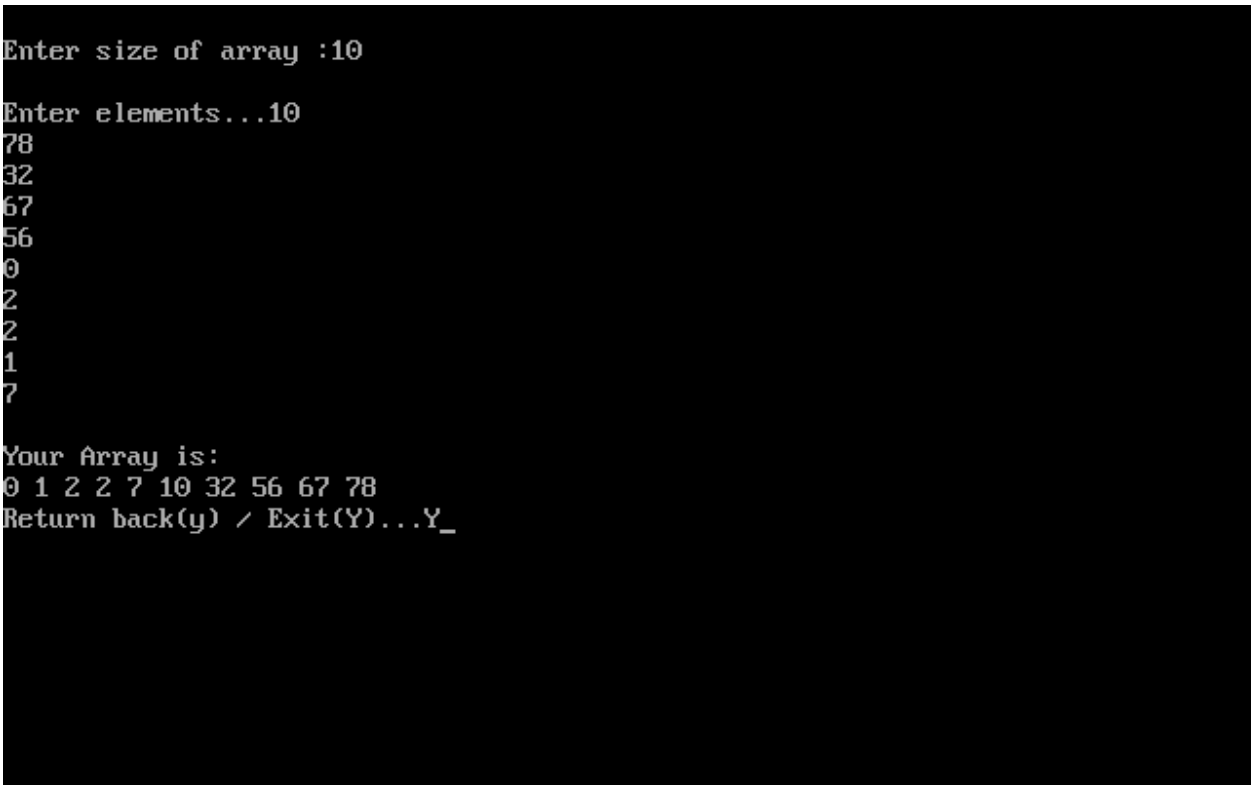
(3)

```
================================================================================
                                PRACTICAL NO.:3
================================================================================

--------------------------------------------------------------------------------
                                Sorting Techniques
 ------------------------------------------------------------------------------
_
                                1.Quick sort
                                2.Merge sort
                                3.Exit
--------------------------------------------------------------------------------

                                Choice : 2_
```

(4)


Enter size of array :10
Enter elements...10
78
32
67
56
0
2
2
1
7

(5)


Enter size of array :10
Enter elements...10
78
32
67
56
0
2
2
1
7
Your Array is:
0 1 2 2 7 10 32 56 67 78
Return back(y) / Exit(Y)...Y_

(6)

PRACTICAL N0. : 4

/* To perform menu-driven code for Stack operations : -
1. Push
2. Pop
3. Peep
4. Change/Update***/**

```
#include<stdio.h>
#include<conio.h>
#define size 4
void Push(int stack[size],int x);
void display(int stack[size]);
int Pop(int stack[size]);
void change(int stack[size],int i,int x);
int Peep(int stack[size],int i);
void dline()
```

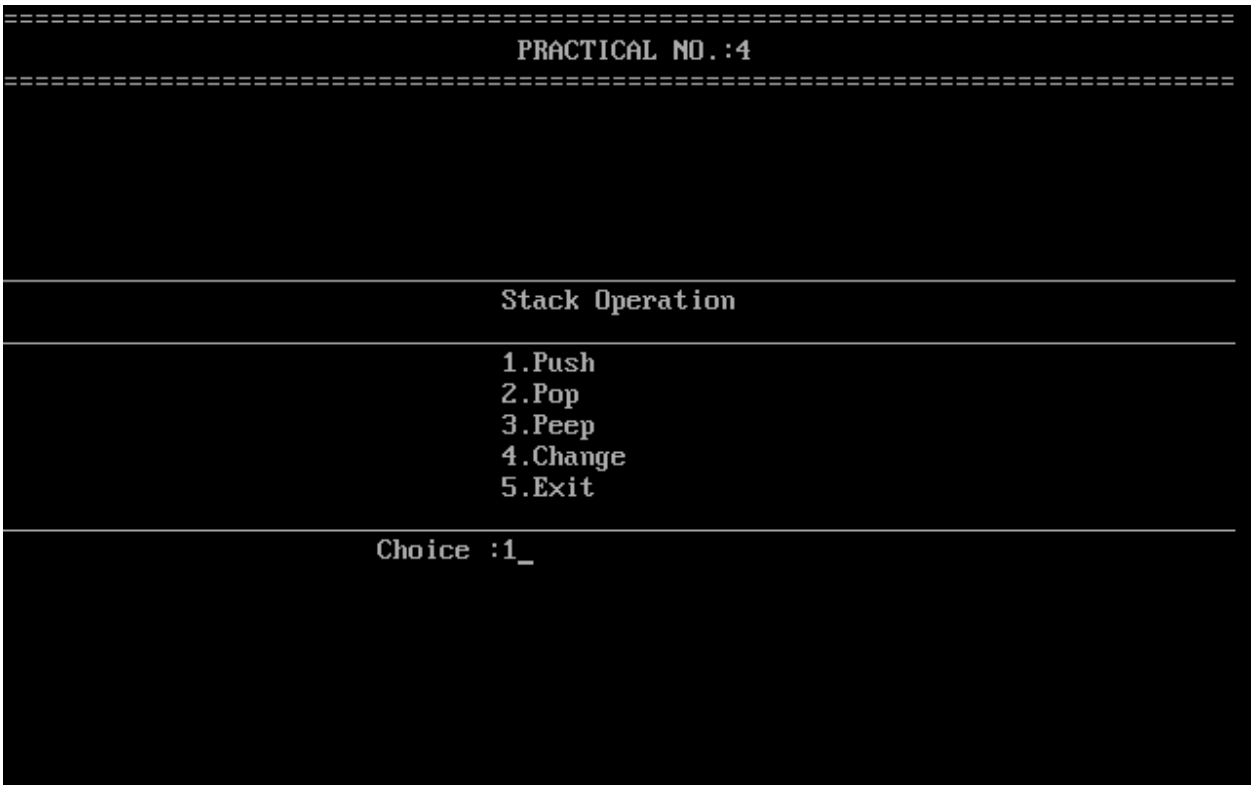
```
{

printf("============================================================
============================");
}
void line()
{

printf("____________________________________________________________________
_____________________");
}
int top=-1;
void main()
{
 int c,stack[size],x,del,i,in;
 char ch;
 do
   {
                clrscr();
                dline();
                printf("\n\t\t\t\t PRACTICAL NO.:4\n");
                dline();
                printf("\n\n\n\n\n\n");
                line();
                printf("\n\t\t\t\tStack Operation\n");
                line();
                printf("\n\t\t\t\t1.Push");
                printf("\n\t\t\t\t2.Pop");
                printf("\n\t\t\t\t3.Peep");
                printf("\n\t\t\t\t4.Change");
                printf("\n\t\t\t\t5.Exit\n");
                line();
                printf("\n\t\t\tChoice :");
                scanf("%d",&c);
                clrscr();
                switch(c)
                {
                        case 1: printf("\nStack Operation : Push\n");
                                        line();
                                        printf("\n Enter element in the stack...");
```

```
                        scanf("%d",&x);
                        Push(stack,x);
                    //
printf("\ntop=%d(MF)\nstack[top]=%d(MF)",top,stack[top]);
                        display(stack);
                        printf("\n");
                        line();
                        break;
                case 2: printf("\nStack Operation : Pop\n");
                        line();
                        del=Pop(stack);
                        if(del==-1)
                                printf("\nStack underflow");
                        else
                                {
                                        printf("\nDeleted element :%d",del);
                                        if(top!=-1)
                                                display(stack);
                                        else
                                                printf("\nNow the stack is
empty");
                                }
                        printf("\n");
                        line();
                        break;
                case 3: printf("\nStack Operation : Peep\n");
                        line();
                        display(stack);printf("\n\n");
                        printf("\ni = ");
                        scanf("%d",&i);
                        in=Peep(stack,i);
                        printf("\n Element to access : %d",in);
                        printf("\n");
                        line();
                        break;
                case 4: printf("\nStack Operation : Change/Update\n");
                        line();
                        display(stack);
                        printf("\ni=");
                        scanf("%d",&i);
```

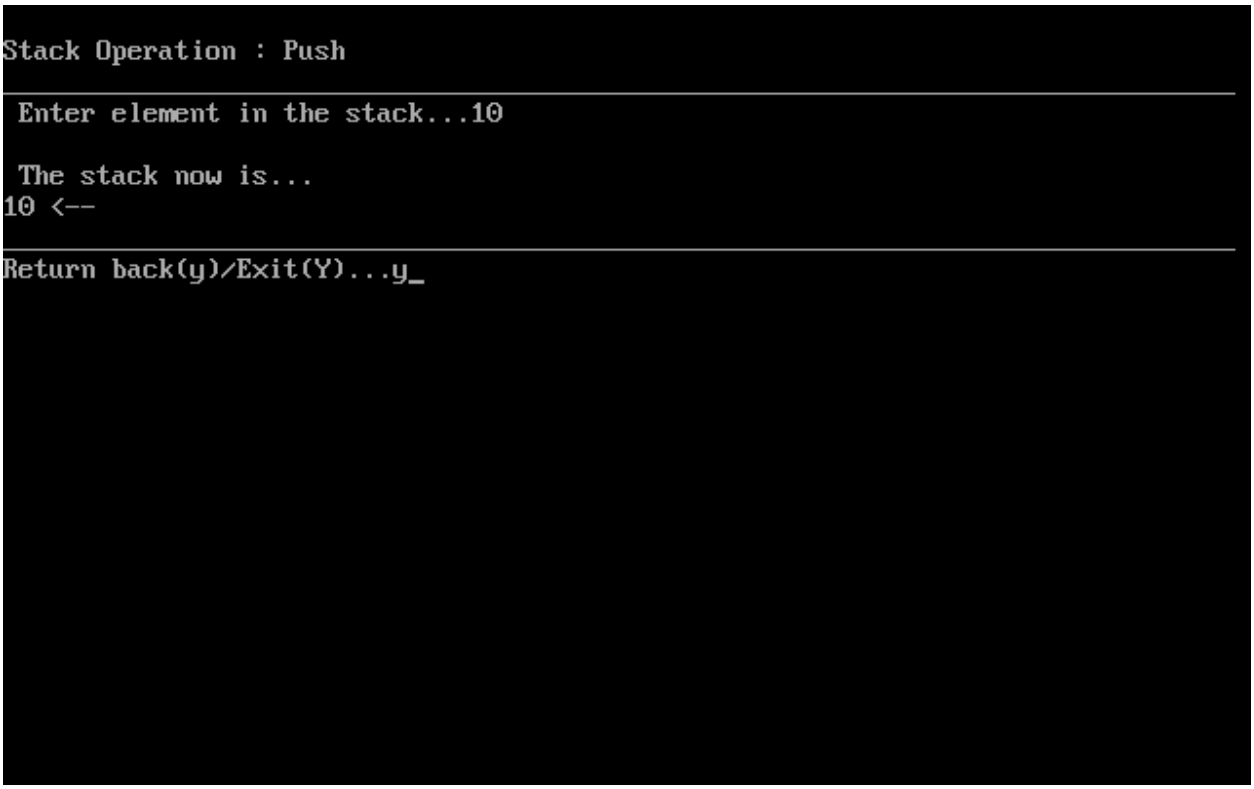
```
                                        printf("\n Enter element :");
                                        scanf("%d",&x);
                                        change(stack,i,x);
                                        display(stack);
                                        printf("\n");
                                        line();
                                         break;
                        case 5: exit(0);

                                default: printf("\n Invalid Entry...");
                }
                printf("\nReturn back(y)/Exit(Y)...");
                fflush(stdin);
                scanf("%c",&ch);
  }while(ch!='Y');
 getch();
}
//---------------------------------------------------------------------------
void Push(int stack[size],int x)
{
 if( (top+1)>=size)
   {    printf("\n Stack Overflow!!!");
        printf("\ntop=%d(FD)",top);
   }
 else
        {
                ++top;
                stack[top]=x;
        //      printf("\nstack[top]=%d",stack[top]);
        }
}
//---------------------------------------------------------------------------
void display(int stack[size])
{
 int i;
 // printf("\ntop=%d(FD)",top);
 printf("\n The stack now is...\n");
 printf("%d <--",stack[top]);
 for(i=top-1;i>=0;--i)
        {
```

```
                printf("\n%d",stack[i]);
        }
}

int Pop(int stack[size])
{
 if(top==-1)
        return(top);
 else
        {
                top-=1;
                return(stack[top+1]);
        }
}

void change(int stack[size],int i,int x)
{
        if((top-i+1)<=-1)
                printf("\nStack underflow on change");
         else
                {       printf("\ntop-i+1=%d-%d+1",top,i);
                        stack[top-i+1]=x;
                }

}

int Peep(int stack[size],int i)
{
 if( (top-i+1)<=-1)
        printf("\nStack Underflow on Peep");
 else
        return(stack[top-i+1]);
}
```

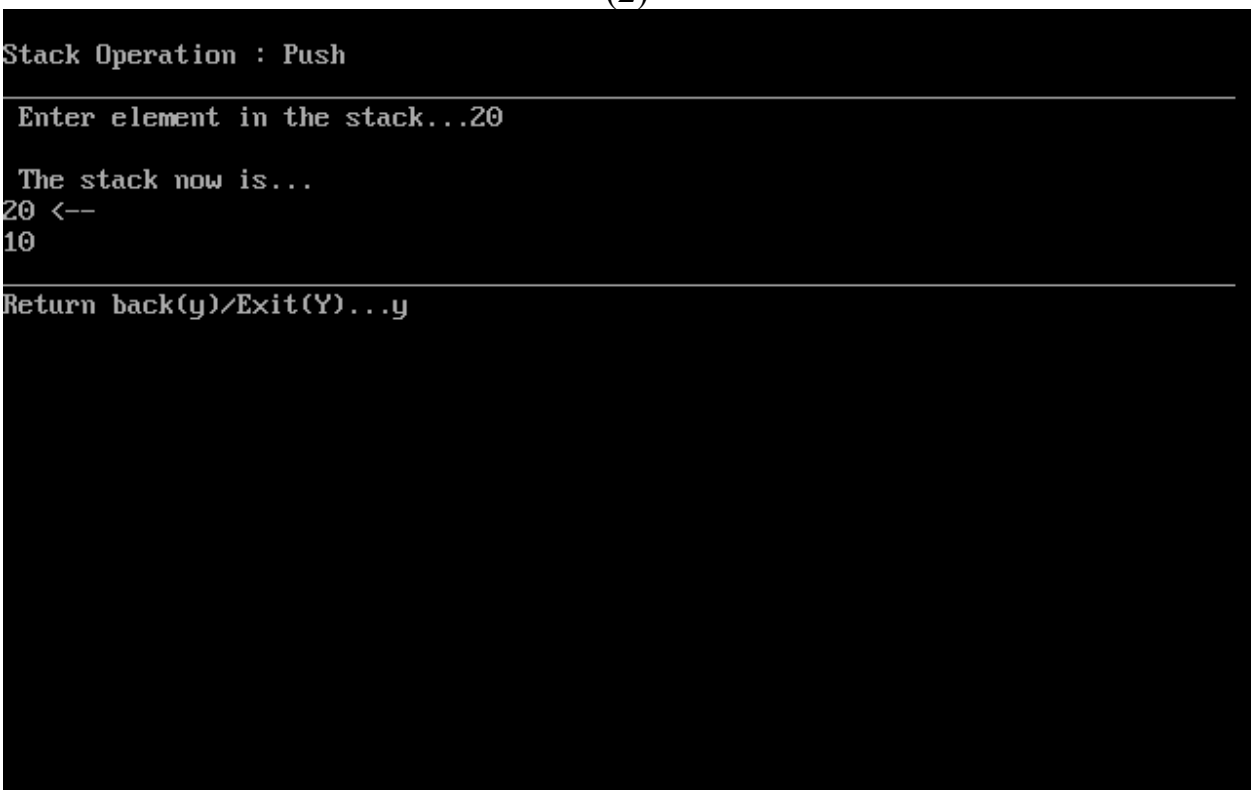
OUTPUT : -


PRACTICAL NO.:4
Stack Operation
1.Push
2.Pop
3.Peep
4.Change
5.Exit
Choice :1_

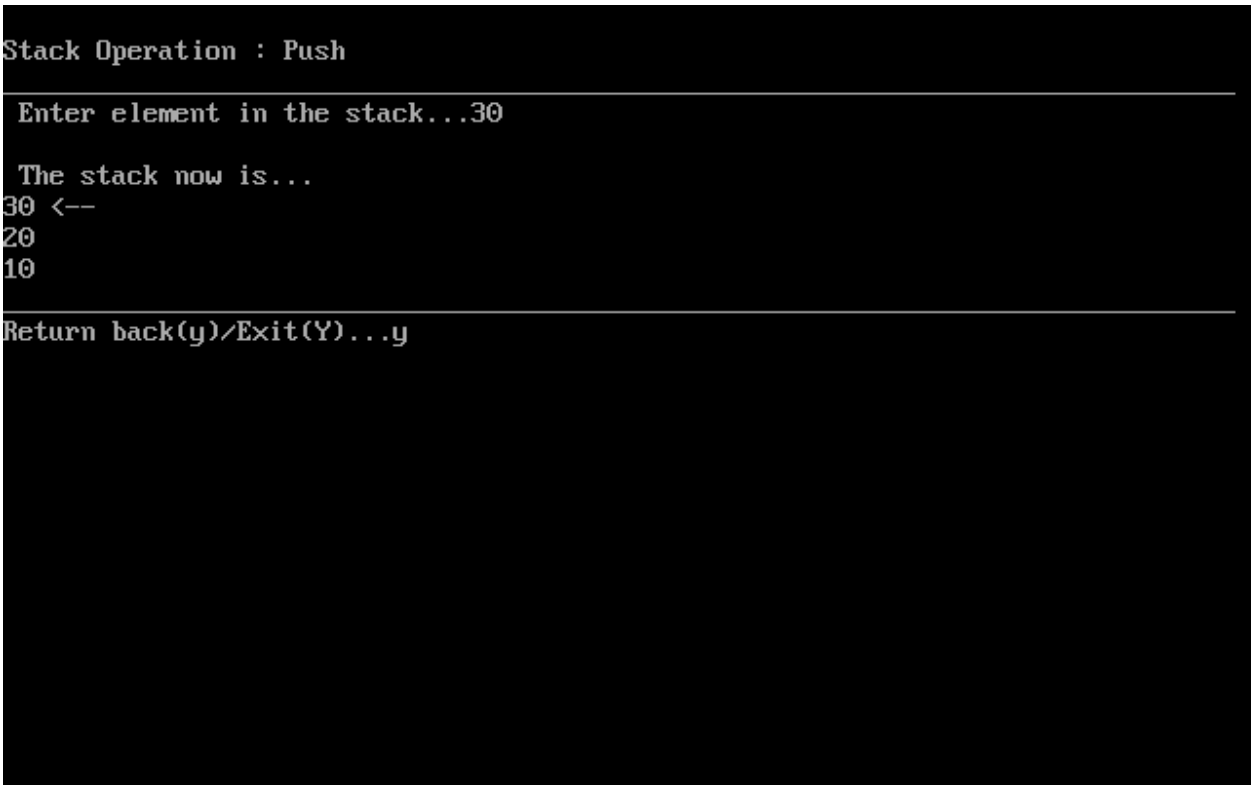
(1)


Stack Operation : Push
Enter element in the stack...10
The stack now is...
10 <--
Return back(y)/Exit(Y)...y

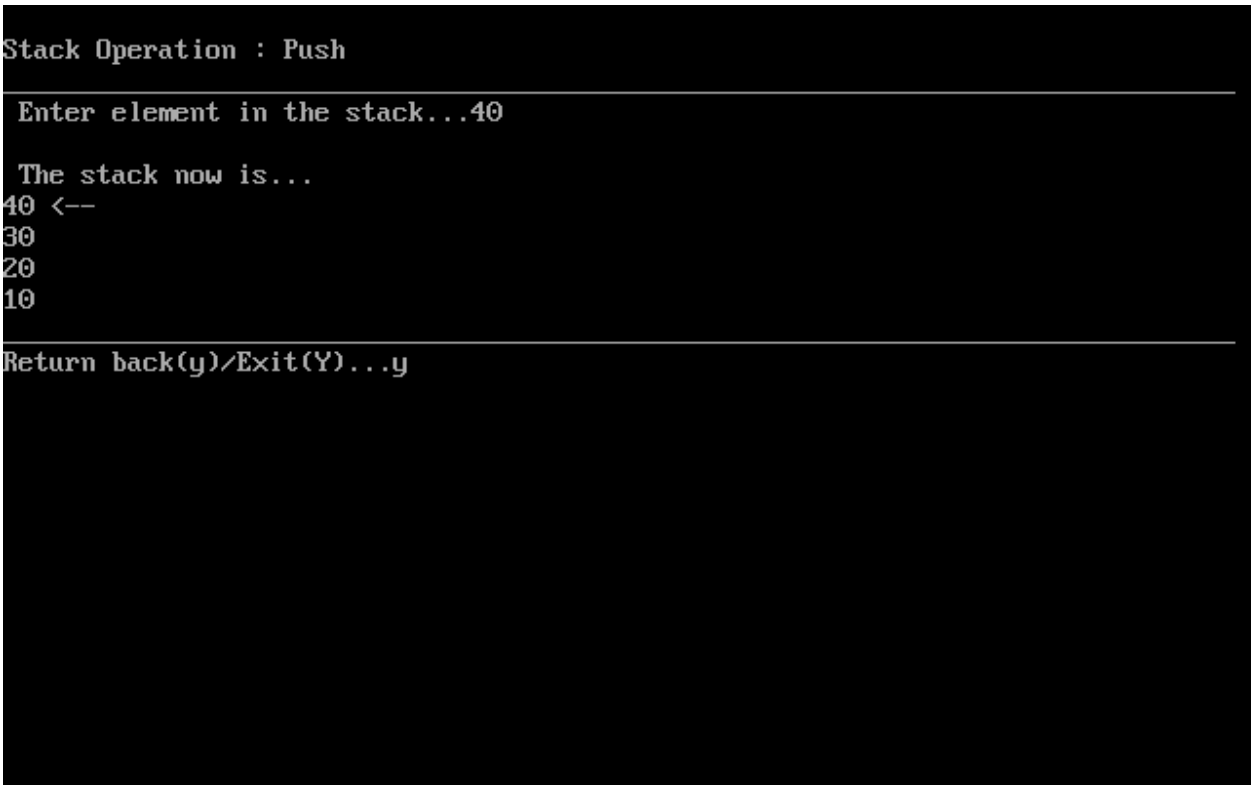
(2)


Stack Operation : Push
Enter element in the stack...20
The stack now is...
20 <--
10
Return back(y)/Exit(Y)...y

(3)


Stack Operation : Push
Enter element in the stack...30
The stack now is...
30 <--
20
10
Return back(y)/Exit(Y)...y

(4)


Stack Operation : Push
Enter element in the stack...40
The stack now is...
40 <--
30
20
10
Return back(y)/Exit(Y)...y

(5)

```
Stack Operation : Push
--------------------------------------------------------------------------------
 Enter element in the stack...50

 Stack Overflow!!!
top=3(FD)
 The stack now is...
40 <--
30
20
10
--------------------------------------------------------------------------------
Return back(y)/Exit(Y)...y
```

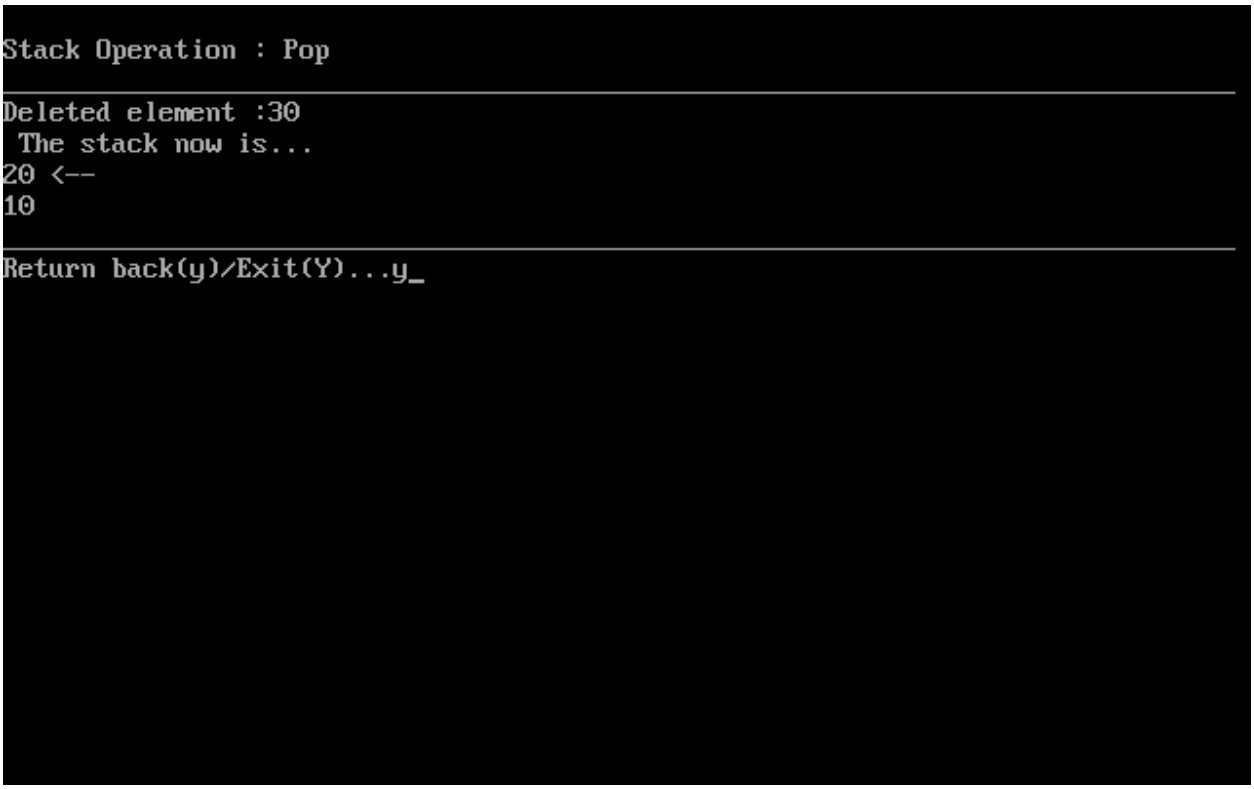
(6)

```
================================================================================
                                PRACTICAL NO.:4
================================================================================

--------------------------------------------------------------------------------
                              Stack Operation
--------------------------------------------------------------------------------
                              1.Push
                              2.Pop
                              3.Peep
                              4.Change
                              5.Exit
--------------------------------------------------------------------------------
                         Choice :2_
```

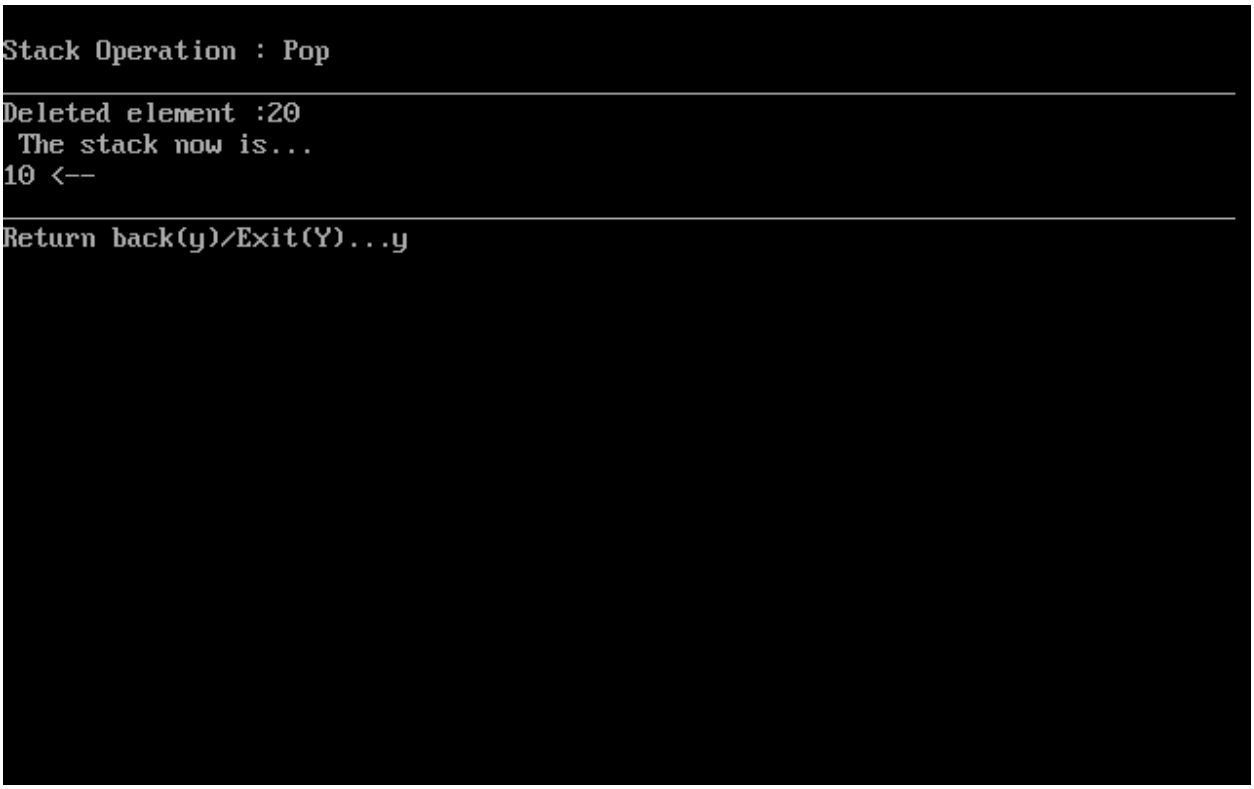
(7)


Stack Operation : Pop
Deleted element :40
The stack now is...
30 <--
20
10
Return back(y)/Exit(Y)...y_

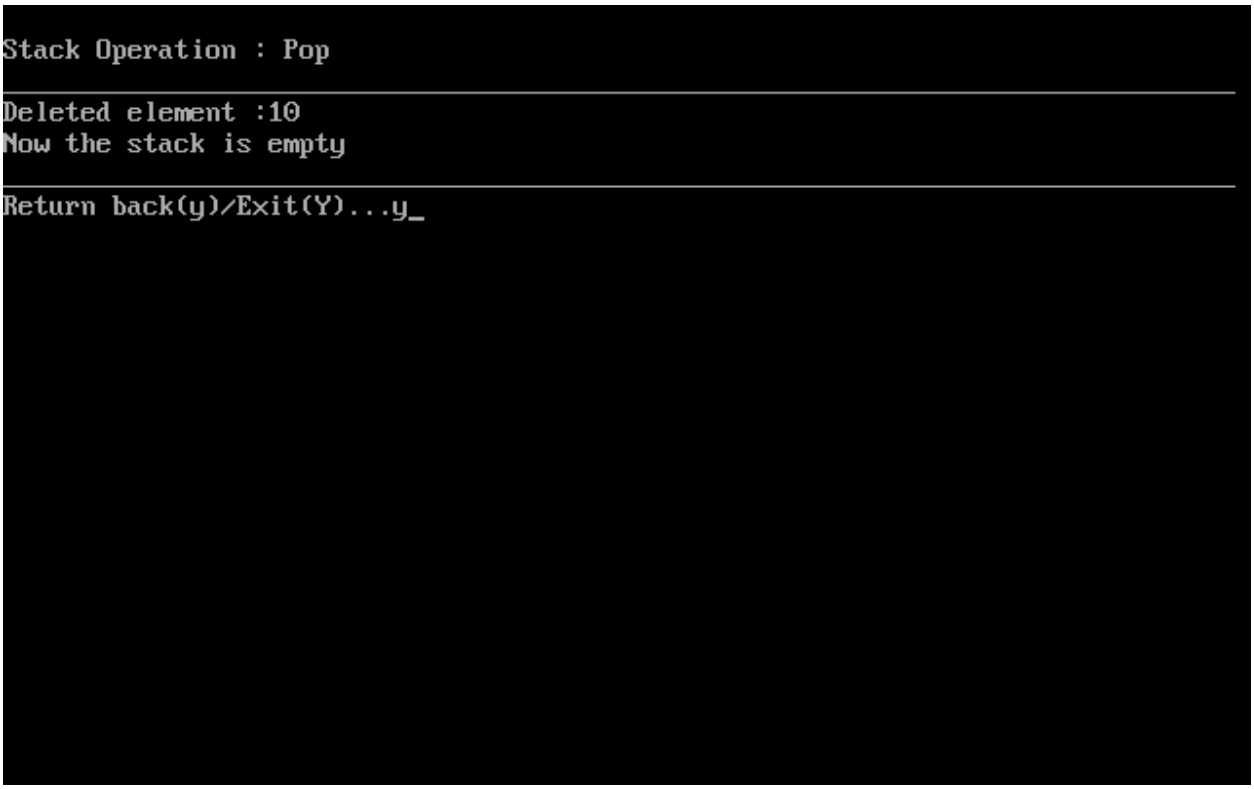
(8)


Stack Operation : Pop
Deleted element :30
The stack now is...
20 <--
10
Return back(y)/Exit(Y)...y_

(9)


Stack Operation : Pop
Deleted element :20
The stack now is...
10 <--
Return back(y)/Exit(Y)...y

(10)


Stack Operation : Pop
Deleted element :10
Now the stack is empty
Return back(y)/Exit(Y)...y

(11)

```
Stack Operation : Pop
--------------------------------------------------------------------------------
Stack underflow
--------------------------------------------------------------------------------
Return back(y)/Exit(Y)...y
```

(12)

```
================================================================================
                                PRACTICAL NO.:4
================================================================================

--------------------------------------------------------------------------------
                              Stack Operation
--------------------------------------------------------------------------------
                              1.Push
                              2.Pop
                              3.Peep
                              4.Change
                              5.Exit
--------------------------------------------------------------------------------
                         Choice :3
```

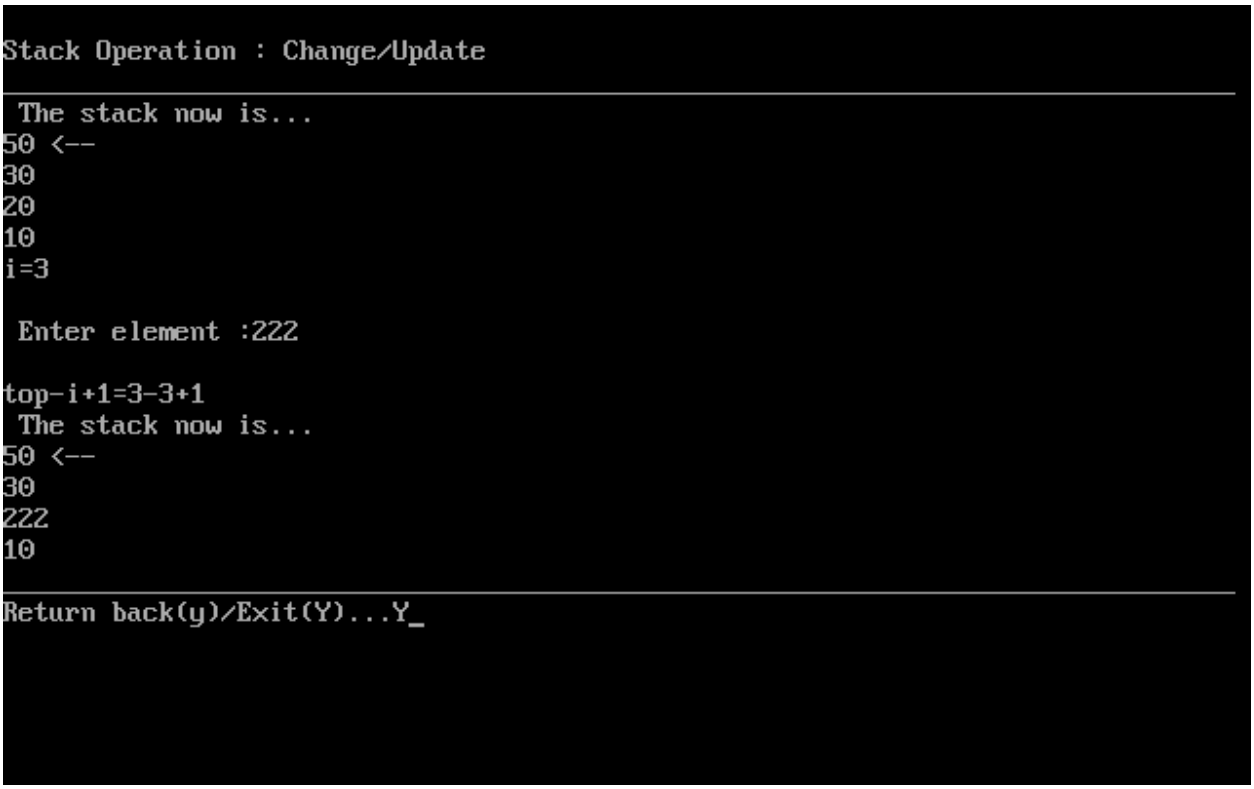
(13)

```
Stack Operation : Peep
--------------------------------------------------------------------------------
 The stack now is...
40 <--
30
20
10


i = 1

 Element to access : 40
--------------------------------------------------------------------------------
Return back(y)/Exit(Y)...y_
```

(14)

```
================================================================================
                                PRACTICAL NO.:4
================================================================================









--------------------------------------------------------------------------------
                            Stack Operation
--------------------------------------------------------------------------------
                            1.Push
                            2.Pop
                            3.Peep
                            4.Change
                            5.Exit
--------------------------------------------------------------------------------
                        Choice :4
```

(15)


Stack Operation : Change/Update
The stack now is...
50 <--
30
20
10
i=3
Enter element :222
top-i+1=3-3+1
The stack now is...
50 <--
30
222
10
Return back(y)/Exit(Y)...Y_

(16)

PRACTICAL NO. : 5

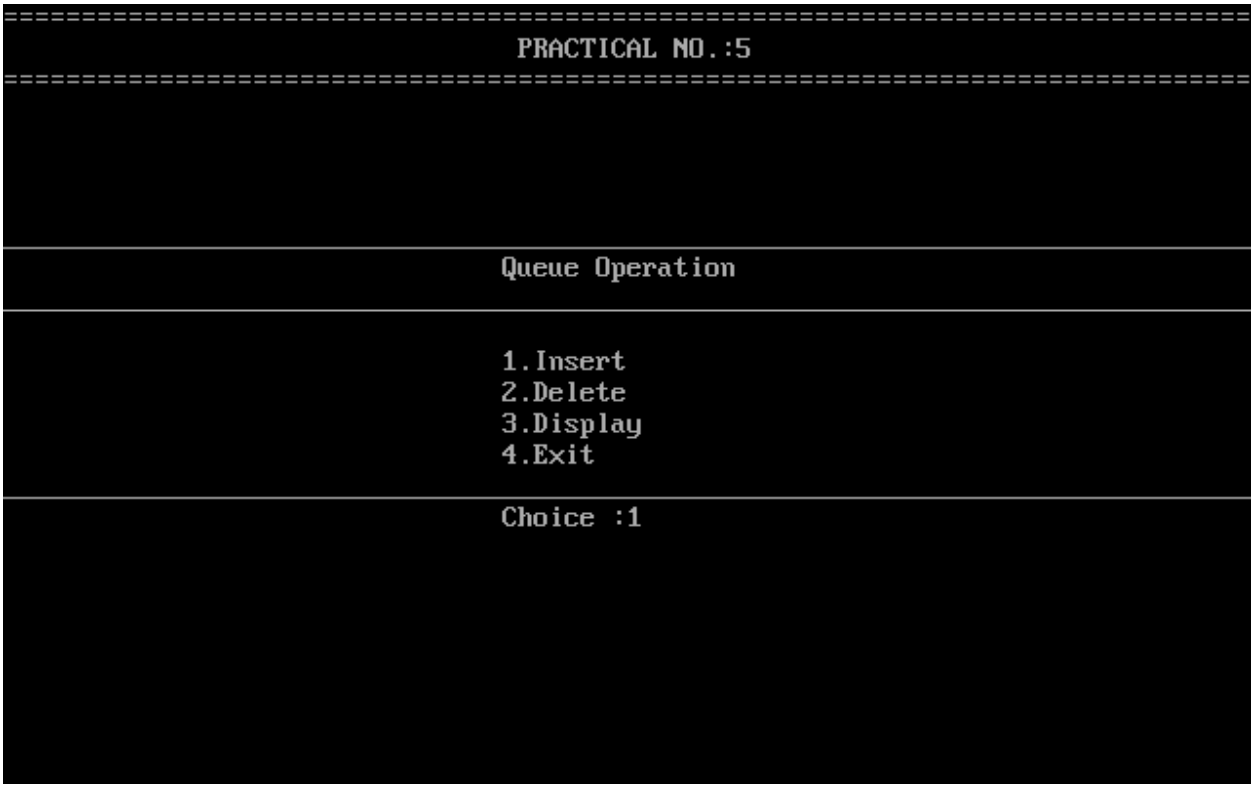
```
/* To perform menu-driven code for Queue operations : -
1. Insertion
2. Deletion
3. Display*/

#include<stdio.h>
#include<conio.h>
#define size 4
void insert(int queue[size],int x);
void qd(int queue[size]);
void display(int queue[size]);
int f=-1,r=-1,del;
void dline()
{
 int i;
 for(i=1;i<=80;++i)
        printf("=");
}
void line()
{
 int i;
 for(i=1;i<=80;++i)
        printf("_");

//
printf("_____
_____");
}

void main()
{
 int c,queue[size],x,del;
 char ch;
 do
   {
                clrscr();
```

```
dline();
printf("\t\t\t\t PRACTICAL NO.:5\n");
dline();
printf("\n\n\n\n");
line();
printf("\t\t\t\tQueue Operation\n");
line();
printf("\n\t\t\t\t\t1.Insert");
printf("\n\t\t\t\t\t2.Delete");
printf("\n\t\t\t\t\t3.Display");
printf("\n\t\t\t\t\t4.Exit\n");
line();
printf("\t\t\t\tChoice :");
scanf("%d",&c);
clrscr();
switch(c)
{
        case 1: printf("\nQueue Operation : Insertion\n");
                        line();
                        printf("\n Enter element in the queue...");
                        scanf("%d",&x);
                        insert(queue,x);
                        printf("\n");
                        line();
                        break;
        case 2: printf("\nQueue Operation : Deletion\n");
                        line();
                        qd(queue);
                        printf("\n");
                        line();
                        break;
        case 3: printf("\nQueue Operation : Display\n");
                        line();
                        display(queue);printf("\n\n");
                        printf("\n");
                        line();
                        break;
        case 4: exit(0);

        default: printf("\n Invalid Entry...");
```

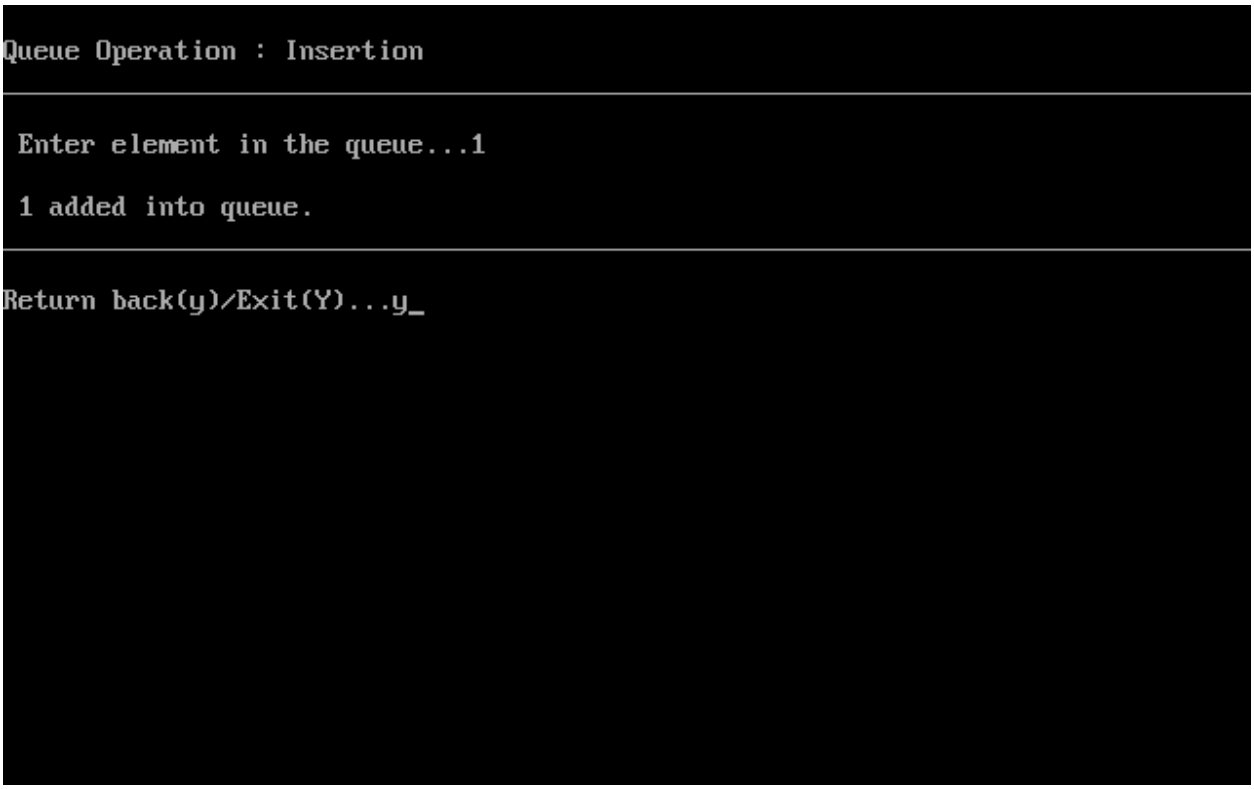
```
                }
                printf("\nReturn back(y)/Exit(Y)...");
                fflush(stdin);
                scanf("%c",&ch);
  }while(ch!='Y');
 getch();
}

void insert(int queue[size],int x)
{
 if((r+1)>=size)    // size =4        r=-1   (-1+1)>=4    0>=4
        printf("\nOverflow");
 else
        {
                r+=1;  //r=-1 + 1 = 0
                queue[r]=x;
                if(f==-1)
                        f+=1;        // f = -1+1 = 0
                printf("\n %d added into queue.",x);
        }
}

void display(int queue[size])
{
 int i;
 if(f==-1)
        printf("\nQueue is empty");
 else
        {
                printf("\n%d <--",queue[f]);
                for(i=f+1;i<=r;++i)
                {
                        printf("\n%d",queue[i]);
                }
        }
 printf("\n\n\nCurrent status -");
 if( (f==-1) && (r==-1))
        printf("\nInserted element : Nil\t\t Deleted element : Nil");
 else
        {
```

```
                if(f==0)
                        printf("\nInserted element : %d\t\t Deleted element :
Nil",queue[r]);
                else
                        printf("\nInserted element : %d\t\t Deleted element :
%d",queue[r],del);

        }
}
void qd(int queue[size])
{
 if(f==-1)
        printf("\nUnderflow");
 else
        {
          printf("\nDeleted element :%d",queue[f]);
          del=queue[f];
          if((f+1)>=r)  // f+1 = (4+1) = 5>=4   \
                {       f=-1;  r=-1;  }
                else
                        f+=1;    //f=1
        }
}
```

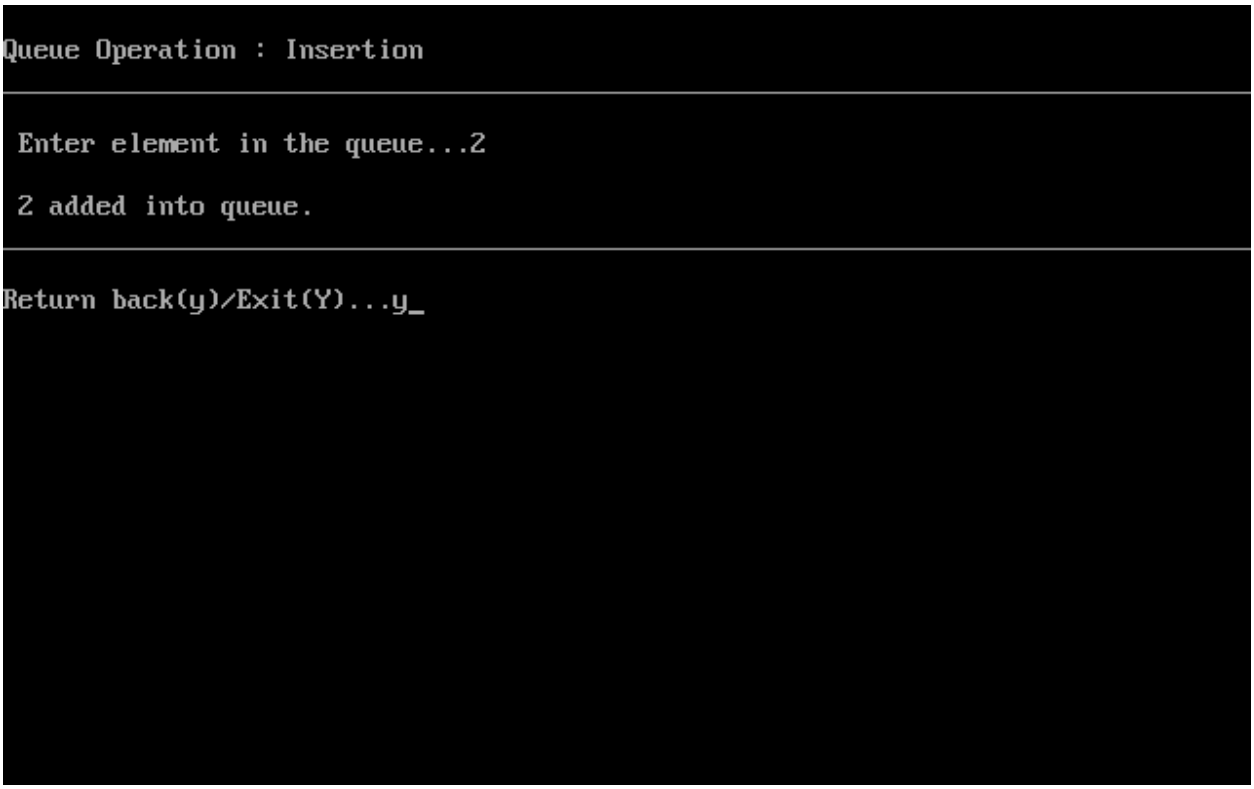
OUTPUT : -


PRACTICAL NO.:5
Queue Operation
1.Insert
2.Delete
3.Display
4.Exit
Choice :1

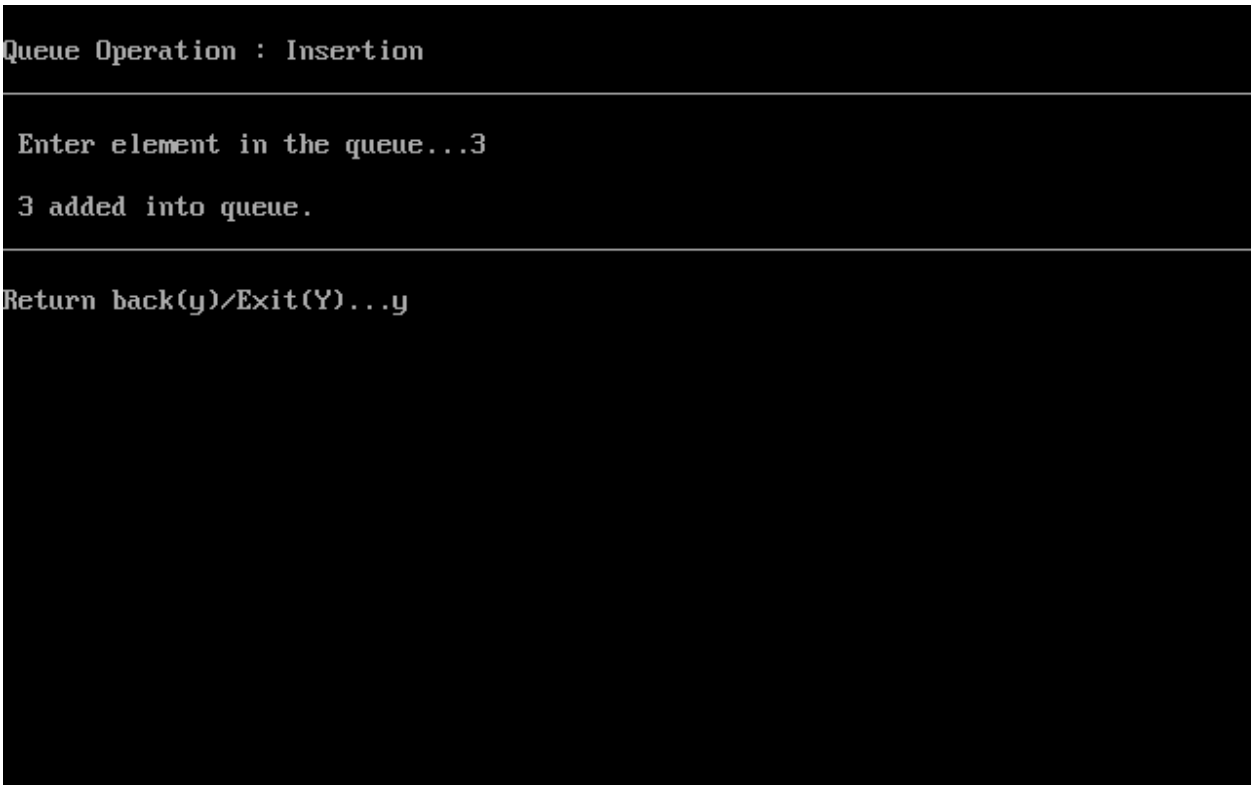
(1)


Queue Operation : Insertion
Enter element in the queue...1
1 added into queue.
Return back(y)/Exit(Y)...y_

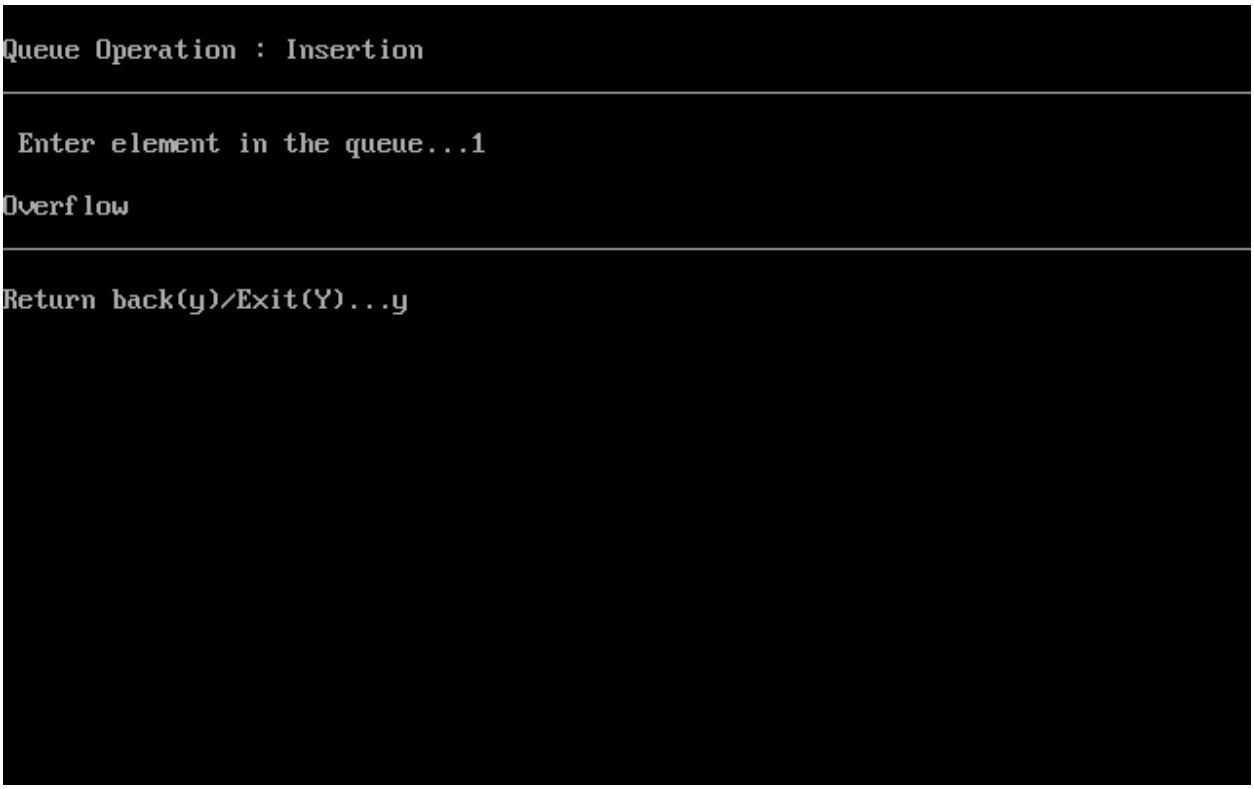
(2)


Queue Operation : Insertion
Enter element in the queue...2
2 added into queue.
Return back(y)/Exit(Y)...y_

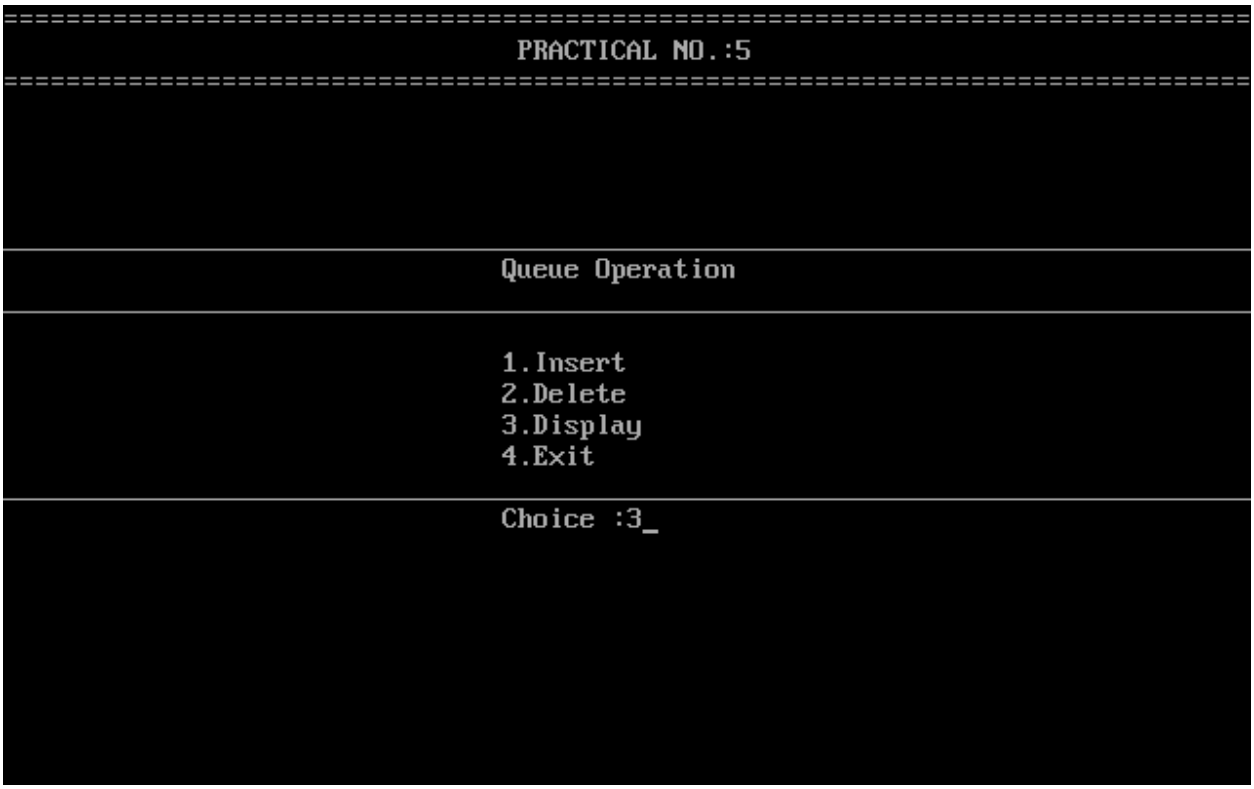
(3)


Queue Operation : Insertion
Enter element in the queue...3
3 added into queue.
Return back(y)/Exit(Y)...y

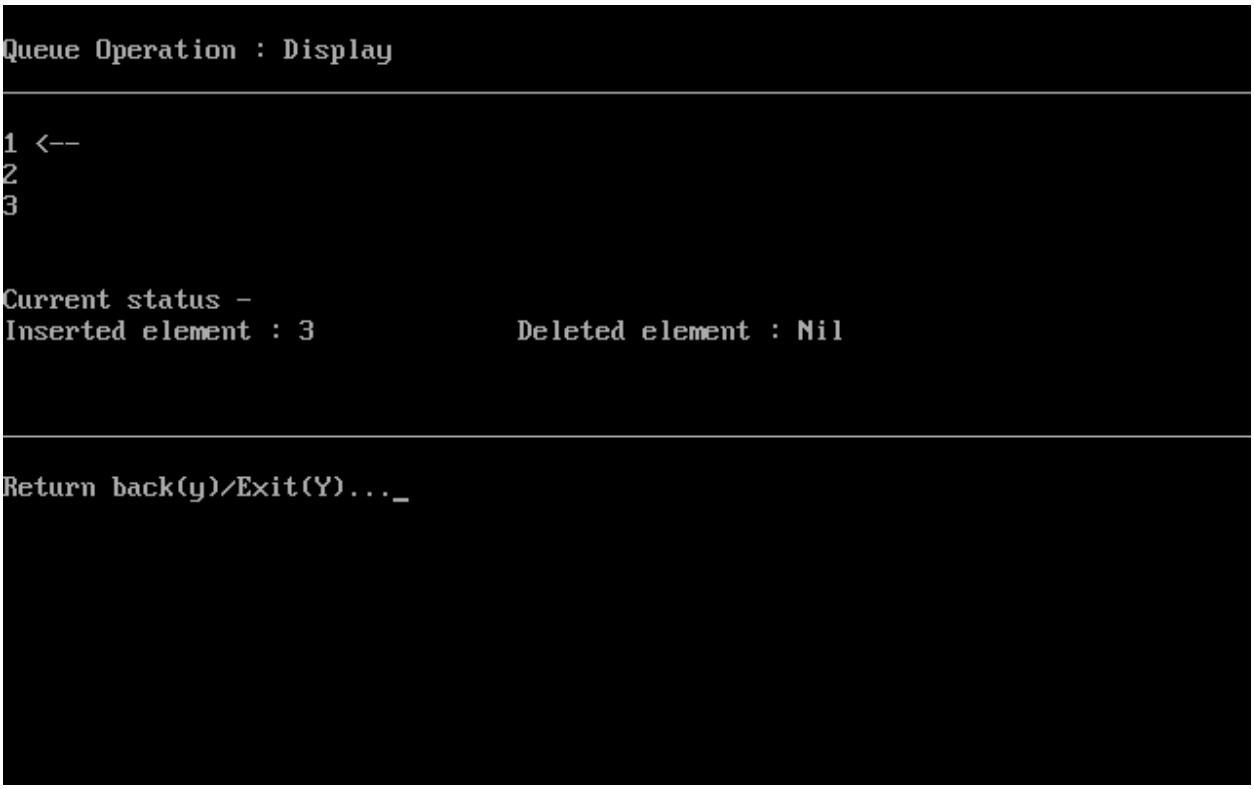
(4)


Queue Operation : Insertion
Enter element in the queue...1
Overflow
Return back(y)/Exit(Y)...y

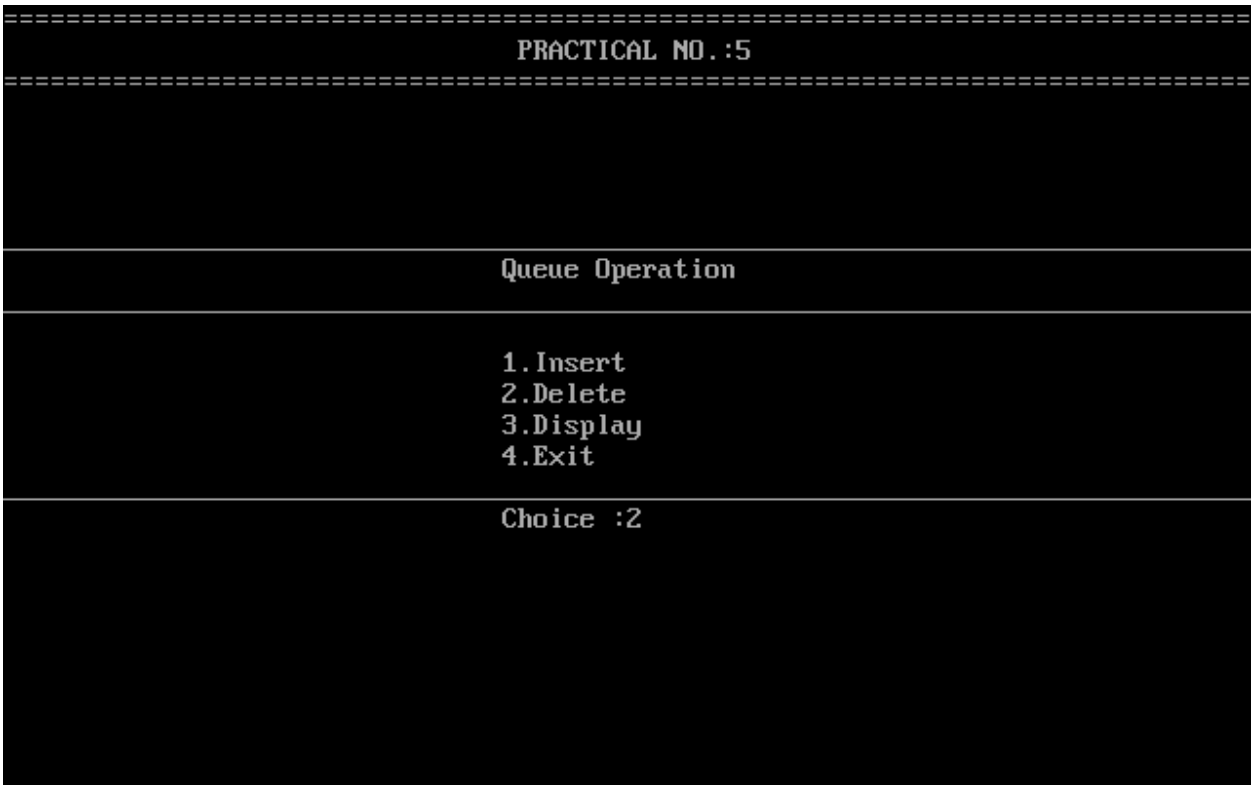
(5)


PRACTICAL NO.:5
Queue Operation
1.Insert
2.Delete
3.Display
4.Exit
Choice :3_

(6)


Queue Operation : Display
1 <--
2
3
Current status -
Inserted element : 3
Deleted element : Nil
Return back(y)/Exit(Y)..._

(7)


PRACTICAL NO.:5
Queue Operation
1.Insert
2.Delete
3.Display
4.Exit
Choice :2

(8)


Queue Operation : Deletion
Deleted element :1
Return back(y)/Exit(Y)...y_

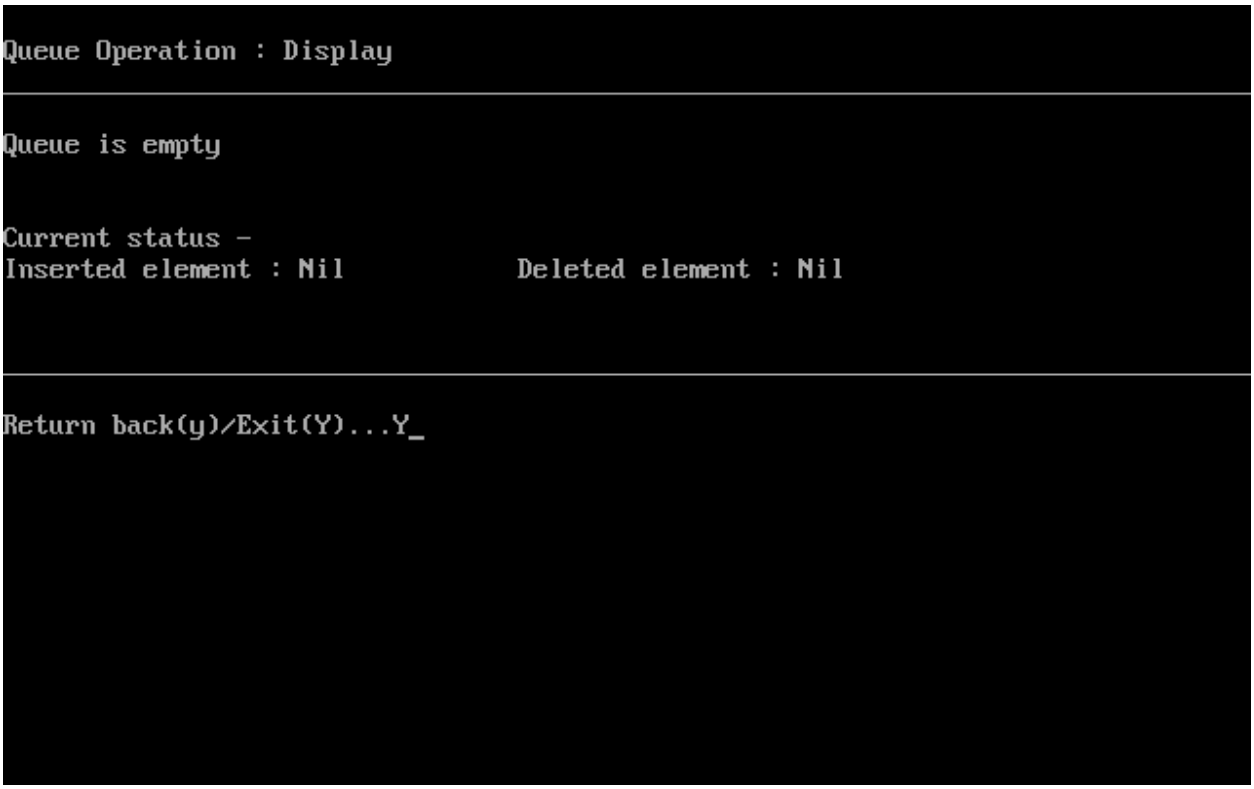
(9)


Queue Operation : Deletion
Deleted element :2
Return back(y)/Exit(Y)...y

(10)


Queue Operation : Deletion
Underflow
Return back(y)/Exit(Y)...y_

(11)


Queue Operation : Display
Queue is empty
Current status -
Inserted element : Nil
Deleted element : Nil
Return back(y)/Exit(Y)...Y_

(12)

PRACTICAL NO. : 6

/* To perform menu-driven code for basic operation of Linked-list : -
1. Insertion(at the beginning, middle and end)
2. Deletion
3. Display*/

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct Node
{
    int info;
    struct Node *link;
} *first = NULL , *avail;
typedef struct Node node;

node *create_node()
{
    avail = (node *)malloc(sizeof(node));
    printf("\n Avail =%u", avail);
    // printf("\n Avail->link =%u", avail->link);
}
void insert_beg(int x);
void insert_middle(int x);
void insert_end(int x);
void delete ();
void display();
void dline();
void line();
int count = 0;
int main()
{
    int ch, n, x;
    char c;
    printf("\n How many node(s) you want to create...");
    scanf("%d", &n);
    create_node();
    do
    {
        printf("\n");
        dline();
        printf("\n\t\t\t\t\t PRACTICAL NO. : 6\n");
```

```
        dline();
        printf("\n\t\t\t    Single Linked-list Operation\n");
        line();
        printf("\n\t\t\t\t1.Insertion");
        printf("\n\t\t\t\t\t(11)-At the begining");
        printf("\n\t\t\t\t\t(12)-At the middle");
        printf("\n\t\t\t\t\t(13)-At the end");
        printf("\n\t\t\t\t2.Deletion");
        printf("\n\t\t\t\t3.Display");
        printf("\n\n\t\t\t\t4.Exit\n");
        line();
        printf("\t\t\t\tChoice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\n Please enter a valid insertion mode...!");
            break;
        case 11:
            while (count < n)
            {
                count += 1;
                printf("\n Enter information of node : ");
                scanf("%d", &x);
                insert_beg(x);
                break;
            }
            if (count == n)
            {
                avail = NULL;
                // printf("\n Availability stack underflow...!");
            }
            break;
        case 12:
            while (count <= n)
            {
                count += 1;
                printf("\n Enter information of node : ");
                scanf("%d", &x);
                insert_middle(x);
                break;
            }
            if (count == n)
            {
                avail = NULL;
```

```
                // printf("\n Availability stack underflow...!");
            }
            break;
        case 13:
            while (count <= n)
            {
                count += 1;
                printf("\n Enter information of node : ");
                scanf("%d", &x);
                insert_end(x);
                break;
            }
            if (count == n)
            {
                avail = NULL;
                // printf("\n Availability stack underflow...!");
            }
            break;
        case 2:
            delete ();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\n Wrong Entry");
        }
        printf("\n Return back / Exit(Y)...");
        fflush(stdin);
        scanf("%c", &c);
    } while (c != 'Y');
    return 0;
}

void insert_beg(int x)
{
    node *new;
    if (avail == NULL)
        printf("\n Availability stack underflow...!");
    else
    {
        new = avail;
        // printf("\n Avail =%u", avail);
```

```
        // printf("\n new =%u", new);
        avail = avail->link;
        new->info = x;
        new->link = NULL;
        if ( first  != NULL ){
            new->link = first;}
        first = new;
        printf("\n %d added", x);
        // getch();
        // printf("\n new->link =%u", new->link);
        // printf("\n first =%u", first);
        // printf("\n Avail->link =%u", avail->link);
        // printf("\n Avail =%u", avail);
        // printf("\n new =%u", new);
        // getch();
    }
}

void insert_middle(int x)
{
    node *New, *save,*pred;

    if (avail == NULL)
        printf("\n Availability stack underflow");
    else
    {
        New = avail;
        avail = avail->link;

        New->info = x;
        if(first == NULL)
        {
            printf("\n Linked list is Empty...!");
            count -= 1;
        }
        else
        {
            if ((New->info) <= (first->info))
            {
                New->link = first;
                first = New;
            }
            else
            {
```

```
                save = first;
                // printf("\n save=%u",save);
                // printf("\n save->link=%u",save->link);
                // printf("\n save->info=%d",save->info);
                while ((save->link != NULL) && (save->info <= New->info))
                {
                    pred =save;
                    save = save->link;
                }
                New->link = save;
                pred->link = New;
            }
        printf("\n%d added", x);
        }
        // printf("\nfirst =%u", first);
        // getch();
    }
}

void insert_end(int x)
{
    node *save, *new;
    if (avail == NULL)
        printf("\n Availability stack underflow...!");
    else
    {
        new = avail;
        avail = avail->link;
        new->info = x;
        new->link = NULL;
        if (first == NULL)
        {    printf("\n Linked-list is Empty");
            count -= 1;
        }
        else
        {
            save = first;
            while ((save->link) != NULL)
            {
                save = save->link;
            }
            save->link = new;
        printf("\n%d added", x);
        }
        // getch();
```

```
    }
}

void delete ()
{
    node *temp, *pred;
    int X;
    // X = (node *)malloc(sizeof(node));

    if (first == NULL)
    {
        printf("\n Linked-List Underflow on deletion...!");
    }
    else
    {
        printf("\n Enterd Linked-List...\n");
        display();
        printf("\n Enter node information to delete : ");
        scanf("%d",&X);
        // X->link = NULL;
        temp = first ;
        while ( (temp->info != X) && (temp->link != NULL) )
        {
            pred = temp;
            temp = temp->link;
        }
        // printf("\n X =%d",X);
        if( (temp->info != X)  ){
            printf("\n Entered node not found");
        }
        else{
            if(X == first->info){
                first = first->link;
            }
            else{
                pred->link = temp->link;
            }
            printf("\n Node deleted...!");
            // printf("\n temp->link=%u",temp->link);
            count -= 1;
             temp->link = avail;
            avail = temp;
        }
```

```
    }
}

void display()
{
    node *temp = first;
    if(temp == NULL){
        printf("\n Linked-List is 'Empty' on display\n");
    }

    while (temp != NULL)
    {
        printf("-->%d", temp->info);
        temp = temp->link;
    }
}

void dline()
{
    int i;
    for (i = 1; i <= 120; ++i)
        printf("=");
}
void line()
{
    int i;
    for (i = 1; i <= 120; ++i)
        printf("_");
}
```

OUTPUT : -

```
How many node(s) you want to create...4

================================================================================================================
                                PRACTICAL NO. : 6
================================================================================================================
                            Single Linked-list Operation
----------------------------------------------------------------------------------------------------------------
                                1.Insertion
                                        (11)-At the begining
                                        (12)-At the middle
                                        (13)-At the end
                                2.Deletion
                                3.Display

                                4.Exit
----------------------------------------------------------------------------------------------------------------
                    Choice : 11
```

(1)

```
Enter information of node : 10

10 added
Return back / Exit(Y)...y
```

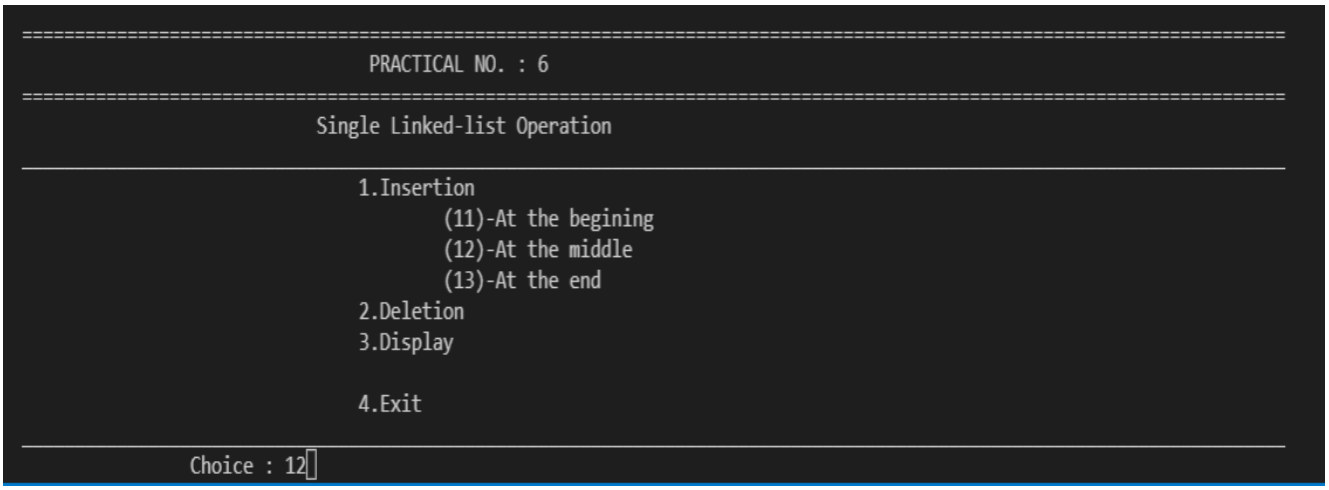
(2)

```
================================================================================================================
                                PRACTICAL NO. : 6
================================================================================================================
                            Single Linked-list Operation
----------------------------------------------------------------------------------------------------------------
                                1.Insertion
                                        (11)-At the begining
                                        (12)-At the middle
                                        (13)-At the end
                                2.Deletion
                                3.Display

                                4.Exit
----------------------------------------------------------------------------------------------------------------
                        Choice : 13
```

(3)

```
Enter information of node : 15

15 added
Return back / Exit(Y)...y
```

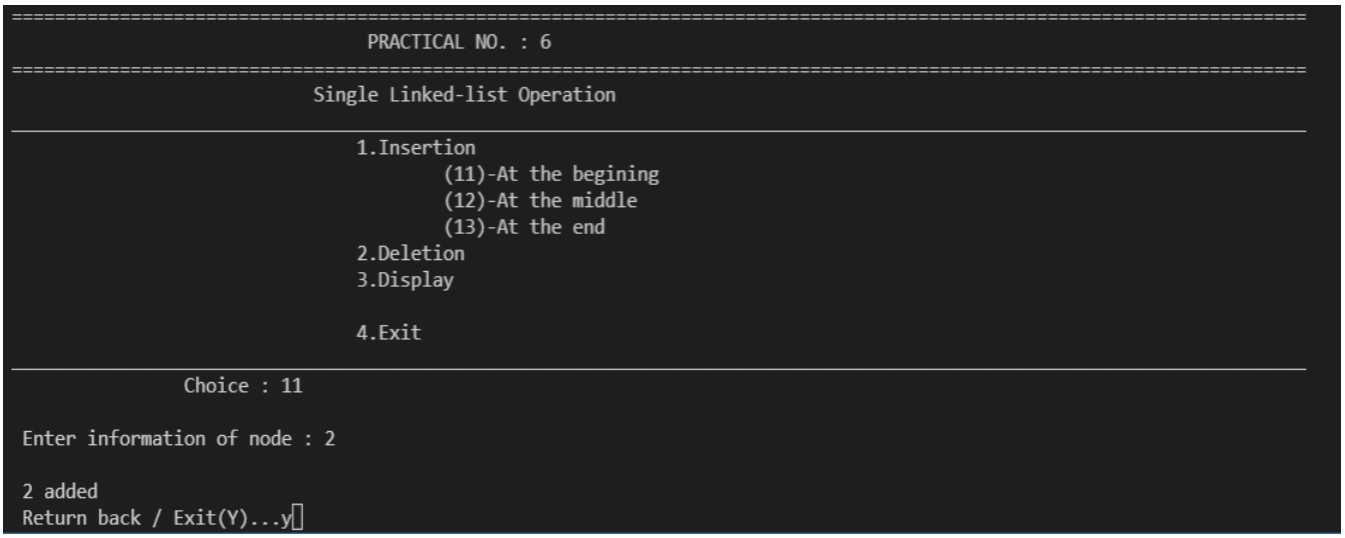
(4)


PRACTICAL NO. : 6
Single Linked-list Operation
1.Insertion
(11)-At the begining
(12)-At the middle
(13)-At the end
2.Deletion
3.Display
4.Exit
Choice : 12

(5)


Enter information of node : 12
12 added
Return back / Exit(Y)...y

(6)


PRACTICAL NO. : 6
Single Linked-list Operation
1.Insertion
(11)-At the begining
(12)-At the middle
(13)-At the end
2.Deletion
3.Display
4.Exit
Choice : 11
Enter information of node : 2
2 added
Return back / Exit(Y)...y

(7)

```
================================================================================
                            PRACTICAL NO. : 6
================================================================================
                        Single Linked-list Operation
--------------------------------------------------------------------------------
                            1.Insertion
                                    (11)-At the begining
                                    (12)-At the middle
                                    (13)-At the end
                            2.Deletion
                            3.Display

                            4.Exit
--------------------------------------------------------------------------------
                Choice : 12

Enter information of node : 1

Availability stack underflow
Return back / Exit(Y)...y
```

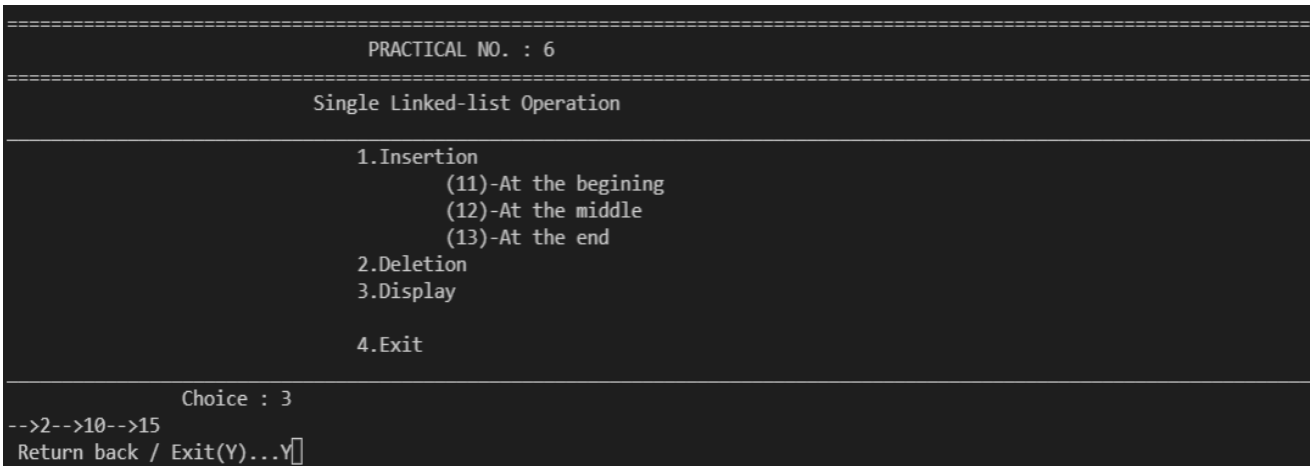
(8)

```
================================================================================
                            PRACTICAL NO. : 6
================================================================================
                        Single Linked-list Operation
--------------------------------------------------------------------------------
                            1.Insertion
                                    (11)-At the begining
                                    (12)-At the middle
                                    (13)-At the end
                            2.Deletion
                            3.Display

                            4.Exit
--------------------------------------------------------------------------------
                Choice : 3
-->2-->10-->12-->15
 Return back / Exit(Y)...y
```

(9)

```
================================================================================
                            PRACTICAL NO. : 6
================================================================================
                        Single Linked-list Operation
--------------------------------------------------------------------------------
                            1.Insertion
                                    (11)-At the begining
                                    (12)-At the middle
                                    (13)-At the end
                            2.Deletion
                            3.Display

                            4.Exit
--------------------------------------------------------------------------------
                Choice : 2

 Enterd Linked-List...
-->2-->10-->12-->15
 Enter node information to delete : 12

 Node deleted...!
 Return back / Exit(Y)...y
```

(10)


PRACTICAL NO. : 6
Single Linked-list Operation
1.Insertion
(11)-At the begining
(12)-At the middle
(13)-At the end
2.Deletion
3.Display
4.Exit
Choice : 3
-->2-->10-->15
Return back / Exit(Y)...Y

(11)