



**L**OVELY  
**P**ROFESSIONAL  
**U**NIVERSITY

***LOVELY  
PROFESSIONAL  
UNIVERSITY***

LAB EVALUATION-3  
ON  
DATA STRUCTURE  
COURSE CODE: CAP-282

SUBMITTED TO

***Balraj Kumar***  
***HOD, Dept. of Computer Application***  
***Faculty of Research & Development***  
***Lovely Professional University***

SUBMITTED BY

***Amlan Kumar Sarkar***  
***Registration No:11900435***  
***Section: D1904***  
***Roll No: RD1904A01***  
***Lovely Professional University***

**Date of Submission: 04-15-2020**

## Set-ODD

1. Develop a menu driven program to implement **Binary Search Tree** using linked representation and perform the following operations on it:
  - Insertion Operation
  - Preorder traversal
  - Inorder traversal
  - Postorder traversal
2. Develop a menu driven program to implement **Linear Queue** using linked representation and perform the following operations on it:
  - Enqueue Operation
  - Dequeue Operation
  - Traversal Operation
3. Develop a program to implement **Insertion Sort** to sort an array of elements in descending order.

## Answer

### Solution 1:

#### Source Code:

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

struct mynode
{
    int item;
    struct mynode *left;
    struct mynode *right;
}*root = NULL,*temp=NULL;

void insert();
void create();
void search(struct mynode *p);
void inorder(struct mynode *p);
void preorder(struct mynode *p);
void postorder(struct mynode *p);
```

```

int flag=1;

void main()
{
    int choice;
    printf("\nBinary Search Tree Operations:");
    printf("\n1. Insert Element:\n");
    printf("2. Inorder Traversal\n");
    printf("3. Preorder Traversal\n");
    printf("4. Postorder Traversal\n");
    printf("5. Exit\n");
    while(1)
    {
        printf("\nEnter Choice from the list: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                insert();
                break;
            case 2:
                inorder(root);
                break;
            case 3:
                preorder(root);
                break;
            case 4:
                postorder(root);
                break;
            case 5:
                exit(0);
            default :
                printf("Please Enter Correct Choice: ");
                break;
        }
    }
}

void insert()
{
    create();
}

```

```

    if(root==NULL)
        root=temp;
    else
        search(root);
}

void create()
{
    int data;
    printf("Enter Inserted Data: ");
    scanf("%d",&data);
    root=(struct mynode*)malloc(1*sizeof(struct mynode));
    root->item=data;
    root->left=root->right=NULL;
}

void search(struct mynode *p)
{
    if((temp->item>p->item)&& (p->right!=NULL))
        search(p->right);
    else if((temp->item>p->item)&&(p->right==NULL))
        p->right=temp;
    else if((temp->item<p->item)&&(p->left!=NULL))
        search(p->left);
    else if((temp->item<p->item)&&(p->left==NULL))
        p->left=temp;
}

void inorder(struct mynode *p)
{
    if(root==NULL)
    {
        printf("Elements aren't available to display...");
        return;
    }
    if (p->left!=NULL)
        inorder(p->left);
    printf("%d->",p->item);
    if (p->right!=NULL)
        inorder(p->right);
}

```

```

}
void preorder(struct mynode *p)
{
    if (root==NULL)
    {
        printf("Elements aren't available to display...");
        return;
    }
    printf("%d->",p->item);
    if (p->left!=NULL)
        preorder(p->left);
    if (p->right!=NULL)
        preorder(p->right);
}
void postorder(struct mynode *p)
{
    if (root==NULL)
    {
        printf("Elements aren't available to display...");
        return;
    }
    if (p->left!=NULL)
        postorder(p->left);
    if (p->right!=NULL)
        postorder(p->right);
    printf("%d->",p->item);
}

```

Output:

Insert Operation:

The screenshot shows a C++ IDE with the file `BST_Operation.c` open. The code defines an `insert()` function that creates a new node and inserts it into a Binary Search Tree (BST). The output window on the right shows the program's execution, where the user repeatedly chooses to insert elements from a list.

```
53     }
54 }
55 void insert()
56 {
57     create();
58     if(root==NULL)
59         root=temp;
60     else
61         search(root);
62 }
63
64 void create()
65 {
66     int data;
67     printf("Enter Inserted Data: ");
68     scanf("%d", &data);
69     root=(struct mynode*)malloc(1*sizeof(struct mynode));
70     root->item=data;
71     root->left=root->right=NULL;
72 }
73
74 void search(struct mynode *p)
75 {
76     if((temp->item>p->item) && (p->right!=NULL))
77         search(p->right);
78     else if((temp->item>p->item) && (p->right==NULL))
79         p->right=temp;
```

Binary Search Tree Operations:  
1. Insert Element:  
2. Inorder Traversal  
3. Preorder Traversal  
4. Postorder Traversal  
5. Exit  
Enter Choice from the list: 1  
Enter Inserted Data: 50  
Enter Choice from the list: 1  
Enter Inserted Data: 20  
Enter Choice from the list: 1  
Enter Inserted Data: 60  
Enter Choice from the list: 1  
Enter Inserted Data: 5  
Enter Choice from the list: 1  
Enter Inserted Data: 25  
Enter Choice from the list: 1  
Enter Inserted Data: 1  
Enter Choice from the list: 1  
Enter Inserted Data: 100  
Enter Choice from the list: 1  
Enter Inserted Data: 85  
Enter Choice from the list: 1  
Enter Inserted Data: 55  
Enter Choice from the list: 1  
Enter Inserted Data: 45  
Enter Choice from the list: 1  
Enter Inserted Data: 35  
Enter Choice from the list:

Traversing(Preorder,Inorder,Postorder) Operation:

The screenshot shows the same C++ IDE with the file `BST_Operation.c` open. The code defines `inorder()`, `preorder()`, and `postorder()` functions for traversing the BST. The output window on the right shows the program's execution, where the user repeatedly chooses to traverse the tree.

```
86 void inorder(struct mynode *p)
87 {
88     if(root==NULL)
89     {
90         printf("Elements are not available to display..");
91         return;
92     }
93     if (p->left!=NULL)
94         inorder(p->left);
95     printf("%d->", p->item);
96     if (p->right!=NULL)
97         inorder(p->right);
98 }
99 void preorder(struct mynode *p)
100 {
101     if (root==NULL)
102     {
103         printf("Elements are not available to display..");
104         return;
105     }
106     printf("%d->", p->item);
107     if (p->left!=NULL)
108         preorder(p->left);
109     if (p->right!=NULL)
110         preorder(p->right);
111 }
112 void postorder(struct mynode *p)
113 {
114     if (root==NULL)
115     {
```

Enter Choice from the list: 1  
Enter Inserted Data: 50  
Enter Choice from the list: 1  
Enter Inserted Data: 20  
Enter Choice from the list: 1  
Enter Inserted Data: 60  
Enter Choice from the list: 1  
Enter Inserted Data: 5  
Enter Choice from the list: 1  
Enter Inserted Data: 25  
Enter Choice from the list: 1  
Enter Inserted Data: 1  
Enter Choice from the list: 1  
Enter Inserted Data: 100  
Enter Choice from the list: 1  
Enter Inserted Data: 85  
Enter Choice from the list: 1  
Enter Inserted Data: 55  
Enter Choice from the list: 1  
Enter Inserted Data: 45  
Enter Choice from the list: 1  
Enter Inserted Data: 35  
Enter Choice from the list: 2  
1 -> 5 -> 20 -> 25 -> 35 -> 45 -> 50 -> 55 -> 60 -> 85 -> 100 -> Enter Choice from the list: 3  
50 -> 20 -> 5 -> 1 -> 25 -> 45 -> 35 -> 60 -> 55 -> 100 -> 85 -> Enter Choice from the list: 4  
1 -> 5 -> 35 -> 45 -> 25 -> 20 -> 55 -> 85 -> 100 -> 60 -> 50 -> Enter Choice from the list: 5  
Process returned 0 (0x0) execution time : 270.184 s  
Press any key to continue.

## Solution 2:

### Source Code:

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

struct Node
{
    int item;
    struct Node *next;
}*front = NULL,*rear = NULL;

void enqueue(int);
void dequeue();
void traverse();

int main()
{
    int choice=1, value;
    printf("\nLinear Queue Operation:");
    while(choice){
        printf("\nPress from the list:");
        printf("\n1.Enqueue 2.Dequeue 3.Traverse 4.Exit\n");
        printf("Enter your choice: ");
        scanf("%d",&choice);

        switch(choice){
            case 1: printf("Enter the value to be added: ");
                    scanf("%d", &value);
                    enqueue(value);
                    break;
            case 2:
                    dequeue();
                    break;
            case 3:
                    traverse();
                    break;
            case 4:
                    exit(0);
```

```

        default:
            printf("\nPlaese choose the correct input...\n");
        }
    }
    return 0;
}

void enqueue(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->item = value;
    newNode -> next = NULL;
    if(front == NULL)
        front = rear = newNode;
    else{
        rear -> next = newNode;
        rear = newNode;
    }
    printf("Added the Element.\n");
}

void dequeue()
{
    if(front == NULL)
        printf("Queue is Empty.\n");
    else{
        struct Node *temp = front;
        front = front -> next;
        printf("\nDeleted element: %d", temp->item);
        free(temp);
    }
}

void traverse()
{
    if(front == NULL)
        printf("\nQueue is Empty!!!\n");
    else{
        struct Node *temp = front;
        while(temp->next != NULL){
            printf("%d--->",temp->item);
            temp = temp -> next;
        }
    }
}

```



```

    }
    printf("%d--->NULL\n",temp->item);
}
}

```

## Output-1:

The screenshot shows a code editor with the following C code for a linear queue. The code includes headers for stdio and conio, defines a Node structure with an integer item and a pointer to the next Node, and implements enqueue, dequeue, and traverse functions. The main function uses a menu-driven interface to perform operations.

```

1  #include<stdio.h>
2  #include<conio.h>
3
4  struct Node
5  {
6      int item;
7      struct Node *next;
8  }*front = NULL,*rear = NULL;
9
10 void enqueue(int);
11 void dequeue();
12 void traverse();
13
14 int main()
15 {
16     int choice=1, value;
17     printf("\nLinear Queue Operation:");
18     while(choice){
19         printf("\nPress from the list:");
20         printf("\n1.Enqueue 2.Dequeue 3.Traverse 4.Exit\n");
21         printf("Enter your choice: ");
22         scanf("%d",&choice);
23
24         switch(choice){
25             case 1: printf("Enter the value to be added: ");
26                     scanf("%d", &value);
27                     enqueue(value);
28                     break;
29             case 2:
30                 dequeue();

```

The output window shows the following sequence of operations:

```

Added the Element.
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 1
Enter the value to be added: 20
Added the Element.
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 1
Enter the value to be added: 30
Added the Element.
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 2
Deleted element: 10
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 3
20--->30--->NULL
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice:

```

## Output-2:

The screenshot shows a code editor with the following C code for a linear queue. The code includes headers for stdio and conio, defines a Node structure with an integer item and a pointer to the next Node, and implements enqueue, dequeue, and traverse functions. The main function uses a menu-driven interface to perform operations.

```

41     return 0;
42 }
43 void enqueue(int value)
44 {
45     struct Node *newNode;
46     newNode = (struct Node*)malloc(sizeof(struct Node));
47     newNode->item = value;
48     newNode->next = NULL;
49     if(front == NULL)
50         front = rear = newNode;
51     else{
52         rear->next = newNode;
53         rear = newNode;
54     }
55     printf("Added the Element.\n");
56 }
57 void dequeue()
58 {
59     if(front == NULL)
60         printf("Queue is Empty.\n");
61     else{
62         struct Node *temp = front;
63         front = front->next;
64         printf("\nDeleted element: %d", temp->item);
65         free(temp);
66     }
67 }
68 void traverse()
69 {
70     if(front == NULL)

```

The output window shows the following sequence of operations:

```

Linear Queue Operation:
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 1
Enter the value to be added: 50
Added the Element.
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 1
Enter the value to be added: 20
Added the Element.
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 1
Enter the value to be added: 70
Added the Element.
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 1
Enter the value to be added: 5
Added the Element.
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 2
Deleted element: 50
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 3
20--->70--->5--->NULL
Press from the list:
1.Enqueue 2.Dequeue 3.Traverse 4.Exit
Enter your choice: 4
Process returned 0 (0x0)   execution time : 68.370 s
Press any key to continue.

```

### Solution 3:

#### Source Code:

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

int main()
{
    int n,array[1000],c,d,temp;

    printf("Enter number of Elements:\n");
    scanf("%d",&n);

    printf("Enter %d Integers are:\n",n);
    for (c =0;c <n;c++)
    {
        printf("Elements are:%d : ",c);
        scanf("%d",&array[c]);
    }

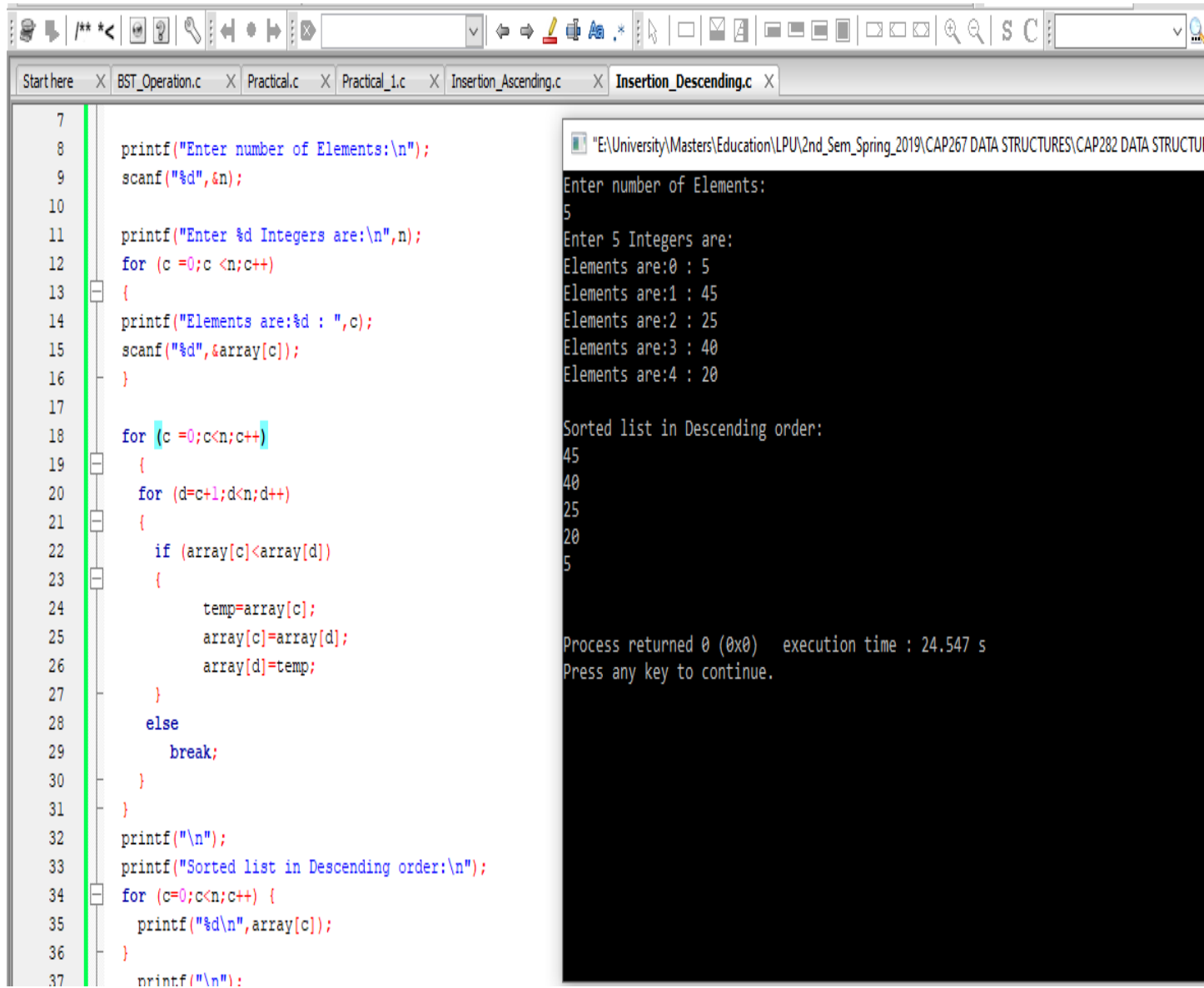
    for (c =0;c<n;c++)
    {
        for (d=c+1;d<n;d++)
        {
            if (array[c]<array[d])
            {
                temp=array[c];
                array[c]=array[d];
                array[d]=temp;
            }
            else
                break;
        }
    }
    printf("\n");
    printf("Sorted list in Descending order:\n");
    for (c=0;c<n;c++) {
        printf("%d\n",array[c]);
    }
```

```

    printf("\n");
return 0;
}

```

## Output:



```

7
8     printf("Enter number of Elements:\n");
9     scanf("%d",&n);
10
11     printf("Enter %d Integers are:\n",n);
12     for (c =0;c <n;c++)
13     {
14         printf("Elements are:%d : ",c);
15         scanf("%d",&array[c]);
16     }
17
18     for (c =0;c <n;c++)
19     {
20         for (d=c+1;d <n;d++)
21         {
22             if (array[c]<array[d])
23             {
24                 temp=array[c];
25                 array[c]=array[d];
26                 array[d]=temp;
27             }
28             else
29                 break;
30         }
31     }
32     printf("\n");
33     printf("Sorted list in Descending order:\n");
34     for (c=0;c <n;c++) {
35         printf("%d\n",array[c]);
36     }
37     printf("\n");

```

```

"E:\University\Masters\Education\LPU\2nd_Sem_Spring_2019\CAP267 DATA STRUCTURES\CAP282 DATA STRUCTURES\Insertion_Descending.c"
Enter number of Elements:
5
Enter 5 Integers are:
Elements are:0 : 5
Elements are:1 : 45
Elements are:2 : 25
Elements are:3 : 40
Elements are:4 : 20

Sorted list in Descending order:
45
40
25
20
5

Process returned 0 (0x0)   execution time : 24.547 s
Press any key to continue.

```