DAY-4 (Data-Structures) Lab Session

EXP - 1

TITLE : C Program to Implement Binary Search Tree

PROGRAM :

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
        int data;
        struct node *right,*left;
};
struct node *root;
struct node *insert(struct node *root,int e)
{
        if(root==NULL)
        {
                struct node *temp;
                temp=(struct node*)malloc(sizeof(struct node));
                temp->data=e;
                temp->left=NULL;
                temp->right=NULL;
                return temp;
        }
        else if(e<root->data)
        {
                root->left=insert(root->left,e);
        }
        else if(e>root->data)
        {
                root->right=insert(root->right,e);
        }
        return root;
}
struct node *successor(struct node *root)
{
        struct node *count;
```

```
	count=root;
	while(count->left!=NULL)
	{
		count=count->left;
	}
	return count;
}
void inorder(struct node *root)
{
	if(root==NULL)
	{
		return;
	}
	else
	{
		inorder(root->left);
		printf("%d ",root->data);
		inorder(root->right);
	}
}
struct node *del(struct node *root,int e)
{
	if(root==NULL)
	{
		printf("Element Not Found");
	}
	else if(e<root->data)
	{root->left=del(root->left,e);
	}
	else if(e>root->data)
	{
		root->right=del(root->right,e);
	}
	else
	{
		if(root->left==NULL)
```

```
                else if(root->right==NULL)
                {
                        root=root->left;
                }
                else
                {
                        struct node *temp;
                        temp=successor(root->right);
                        root->data=temp->data;
                        root=del(root->right,temp->data);
                }
        }
        return root;
}
int main()
{
        int a,b,c,e;
        while(true)
        {
        printf("\n1.INSERT\n2.DELETE\n");
        printf("Enter the Choice:");
        scanf("%d",&c);
        switch(c)
        {
                case 1:
                        printf("Enter The Element:");
                        scanf("%d",&e);
                        root=insert(root,e);
                        break;
                case 2:
                        printf("Enter the Data to delete:");
                        scanf("%d",&e);
                        root=del(root,e);
                        break;
                case 3:
```

```
            inorder(root);
        }
        }
}
```

INPUT AND OUTPUT :

```
1.INSERT
2.DELETE
3.DISPLAY
Enter the Choice:1
Enter The Element:2

1.INSERT
2.DELETE
3.DISPLAY
Enter the Choice:1
Enter The Element:4

1.INSERT
2.DELETE
3.DISPLAY
Enter the Choice:1
Enter The Element:5

1.INSERT
2.DELETE
3.DISPLAY
Enter the Choice:3
2 4 5
1.INSERT
2.DELETE
3.DISPLAY
Enter the Choice:
```

RESULT :

The C Program for Implementing Binary Search Tree is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-4 (Data-Structures) Lab Session

EXP - 2

TITLE : C Program to Merge two Linked List Data Structures

PROGRAM :

```
#include <stdio.h>
#include <stdlib.h>
struct lis
{
        int data;
        struct lis *next;
};
struct lis *h1,*h2;
int main()
{
        int a,b,c,i,j,k=0;
        while(k==0)
        {
                printf("1.INSERT1\n2.INSERT2\n");
                printf("Enter Choice:");
                scanf("%d",&b);
                switch(b)
                {
                        case 1:
                                printf("Enter -1 to stop Storing\n");
                                printf("Enter The Data:");
                                scanf("%d",&c);
                                if(c!=-1)
                                {
                                        struct lis *nn;
                                        nn=(struct lis*)malloc(sizeof(struct lis));
                                        nn->data=c;
                                        nn->next=NULL;
                                        h1=nn;
                                        while(c!=-1)
                                        {
                                                struct lis *h;
```

```
h=h1;
                    }
                    break;
            case 2:
                    printf("Enter -1 to stop Storing\n");
                    printf("Enter The Data:");
                    scanf("%d",&c);
                    if(c!=-1)
                    {
                            struct lis *nn;
                            nn=(struct lis*)malloc(sizeof(struct lis));
                            nn->data=c;
                            nn->next=NULL;
                            h2=nn;
                            while(c!=-1)
                            {
                                    struct lis *h;
                                    struct lis*n;
                                    n=(struct lis*)malloc(sizeof(struct lis));
                                    h=h2;
                                    printf("Enter The Data:");
                                    scanf("%d",&c);
                                    n->data=c;
                                    n->next=NULL;
                                    while(h->next!=NULL)
                                    {
                                            h=h->next;
                                    }
                                    h->next=n;
                            }
                            struct lis *h;
                            h=h2;
                            k=1;
                    }
                    break;
```

```
            }
        }
        struct lis *h,*l,*t;
        h=h1;
        while(h->next!=NULL)
        {
            t=h;
            h=h->next;
        }
        t->next=h2;
        h=h1;
        while(h->next!=NULL)
        {
            printf("[%d|.]->",h->data);
            h=h->next;
        }
        printf("Merged LinkedList");
}
```

INPUT AND OUTPUT :

```
1.INSERT1
2.INSERT2
Enter Choice:1
Enter -1 to stop Storing
Enter The Data:5
Enter The Data:9
Enter The Data:4
Enter The Data:3
Enter The Data:
8
Enter The Data:-1
1.INSERT1
2.INSERT2
Enter Choice:2
Enter -1 to stop Storing
Enter The Data:3
Enter The Data:4
Enter The Data:7
Enter The Data:-1
[5|.]->[9|.]->[4|.]->[3|.]->[8|.]->[3|.]->[4|.]->[7|.]->Merged LinkedList
```

RESULT :

The C Program for Merging Linked List Data Structures is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-4 (Data-Structures) Lab Session

EXP - 3

TITLE : C Program for Sorting elements using Bubble Sort.

PROGRAM :

```
#include <stdio.h>
int main()
{
        int a[10],size;
        printf("Enter the Size of the array:");
        scanf("%d",&size);
        printf("Enter the array Elements:");
        for(int i=0;i<size;i++)
        {
                scanf("%d",&a[i]);
        }
        for(int i=0;i<size-1;i++)
        {
                printf("PASS-%d:\n",i+1);
                for(int j=0;j<size-i-1;j++)
                {
                        if(a[j]>a[j+1])
                        {
                                int temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;
                        }
                }
                for(int j=0;j<size;j++)
                {
                        printf("%d ",a[j]);
                }
                printf("\n");
        }
}
```

INPUT AND OUTPUT :

```
Enter the Size of the array:5
Enter the array Elements:5 4 3 2 1
PASS-1:
4 3 2 1 5
PASS-2:
3 2 1 4 5
PASS-3:
2 1 3 4 5
PASS-4:
1 2 3 4 5
```

RESULT :

The C Program for Sorting Elements using Bubble Sort is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-4 (Data-Structures) Lab Session

EXP - 4

TITLE : C Program for Implementing Insertion Sort.

PROGRAM :

```
#include <stdio.h>
int main()
{
        int a[10],size;
        printf("Enter the size of the array:");
        scanf("%d",&size);
        printf("Enter the Elements:");
        for(int i=0;i<size;i++)
        {
                scanf("%d",&a[i]);
        }
        for(int i=0;i<size-1;i++)
        {
                printf("INSERT-%d\n",a[i+1]);
                for(int j=i+1;j>0;j--)
                {
                        if(a[j]<a[j-1])
                        {
                                int temp=a[j];
                                a[j]=a[j-1];
                                a[j-1]=temp;
                        }
                }
                for(int j=0;j<size;j++)
                {
                        printf("%d ",a[j]);
                }
                printf("\n");
        }
}
```

INPUT AND OUTPUT :

```
Enter the size of the array:5
Enter the Elements:5 4 3 2 1
INSERT-4
4 5 3 2 1
INSERT-3
3 4 5 2 1
INSERT-2
2 3 4 5 1
INSERT-1
1 2 3 4 5
```

RESULT :

The C Program for Implementing Insertion Sort is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-4 (Data-Structures) Lab Session

EXP - 5

TITLE : C Program for Implementing Selection Sort.

PROGRAM :

```
#include <stdio.h>
int main()
{
        int a[10],size;
        printf("Enter the Size of array:");
        scanf("%d",&size);
        printf("Enter the Elements:");
        for(int i=0;i<size;i++)
        {
                scanf("%d",&a[i]);
        }
        for(int i=0;i<size;i++)
        {
                int min=a[i];
                int f=-1;
                for(int j=i+1;j<size;j++)
                {
                        if(min>a[j])
                        {
                                min=a[j];
                                f=j;
                        }
                }
                if(f!=-1)
                {
                        int temp=a[i];
                        a[i]=min;
                        a[f]=temp;
                }
                for(int j=0;j<size;j++)
                {
                        printf("%d ",a[j]);}}}
```

INPUT AND OUTPUT :

```
Enter the Size of array:5
Enter the Elements:8 4 1 9 7
1 4 8 9 7
1 4 8 9 7
1 4 7 9 8
1 4 7 8 9
1 4 7 8 9
```

RESULT :

The C Program for Implementing Selection Sort is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-4 (Data-Structures) Lab Session

EXP - 6

TITLE : C Program for Implementing Merge Sort.

PROGRAM :

```
#include <stdio.h>
int msort(int a[],int l,int h)
{
	if(l!=h)
	{
		int mid=(l+h)/2;
		msort(a,l,mid);
		msort(a,mid+1,h);
	}
	else
	{
		int temp;
		for(int i=0;i<h;i++)
		{
			for(int j=0;j<h;j++)
			{
				if(a[j]>a[j+1])
				{
					temp=a[j];
					a[j]=a[j+1];
					a[j+1]=temp;
				}
			}
		}
	}
}
int main()
{
	int a[10],size;
	printf("Enter the Size of the array:");
	scanf("%d",&size);
	printf("Enter The Elements:");
```

```
        for(int i=0;i<size;i++)
        {
                scanf("%d",&a[i]);
        }
        msort(a,0,size-1);
        for(int i=0;i<size;i++)
        {
                printf("%d ",a[i]);
        }
}
```

INPUT AND OUTPUT :

```
Enter the Size of the array:5
Enter The Elements:14 2 9 8 4
2 4 8 9 14
```

RESULT :

The C Program for Implementing Merge Sort is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-4 (Data-Structures) Lab Session

EXP - 7

TITLE : C Program for Performing Hashing using Linear Probing.

PROGRAM :

```
#include <stdio.h>
int main()
{
        int a[10],size=-1,i,f=0,c,e,h;
        for(i=0;i<10;i++)
        {
                a[i]=-1;
        }
        while(true)
        {
                printf("\n1.INSERT\n2.DELETE\n3.DISPLAY\n");
                printf("Enter the Choice:");
                scanf("%d",&c);
                switch(c)
                {
                        case 1:
                                if(size<9)
                                {
                                        printf("Enter the Element to Insert:");
                                        scanf("%d",&e);
                                        i=0,f=0;
                                        while(f==0)
                                        {
                                                h=(e+i)%10;
                                                if(a[h]==-1)
                                                {
                                                        f=1;
                                                        a[h]=e;
                                                        size++;
                                                        printf("\nElement Settled at %d Position\n",h);
                                                }
                                                else
```

```
{
                                        printf("Collision ocuured at %d Position\n",h);
                                        i++;
                                }
                        }
                }
                else
                {
                        printf("Hashing Table Full");
                }
                break;
        case 2:
                if(size>-1)
                {
                        int k=0;
                        printf("Enter The Element to Delete:");
                        scanf("%d",&e);
                        for(int j=0;j<10;j++)
                        {
                                if(a[j]==e)
                                {
                                        printf("\nElement Found at %d Position and
Deleted\n",j);
                                        a[j]=-1;
                                        size--;
                                        k=1;
                                        break;
                                }
                        }
                        if(k==0)
                        {
                                printf("\nElement Not Found\n");
                        }
                }
                else
                {
```

```
                        printf("Hashing Table Empty");
                }
                break;
        case 3:
                for(int j=9;j>=0;j--)
                {
                        if(a[j]==-1)
                        {
                                printf("%d  NULL\n",j);
                        }
                        else
                        {
                                printf("%d  %d\n",j,a[j]);}}
                break;}}}
```

INPUT AND OUTPUT :

```
1.INSERT
2.DELETE
3.DISPLAY
Enter the Choice:1
Enter the Element to Insert:5

Element Settled at 5 Position

1.INSERT
2.DELETE
3.DISPLAY
Enter the Choice:3
9   NULL
8   NULL
7   NULL
6   NULL
5   5
4   NULL
3   NULL
2   NULL
1   NULL
0   NULL
```

RESULT :

The C Program for Performing Hashing using Linear Probing is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-4 (Data-Structures) Lab Session

EXP - 8

TITLE : C Program for Sorting array Using Quick Sort.

PROGRAM :

```
#include <stdio.h>
int swap(int *a,int *b)
{
	int t=*a;
	*a=*b;
	*b=t;
}
int partition(int a[],int low,int high)
{
	int p=a[high];
	int i=(low-1);
	for(int j=low;j<high;j++)
	{
		if(a[j]<=p)
		{
			i++;
			swap(&a[i],&a[j]);
		}
	}
	swap(&a[i+1],&a[high]);
	return i+1;
}
int quick(int a[],int low,int high)
{
	if(low<high)
	{
		int pi=partition(a,low,high);
		quick(a,low,pi-1);
		quick(a,pi+1,high);
	}
}
int main()
```

```
{
        int a[10];
        printf("Enter Elements:");
        for(int i=0;i<5;i++)
        {
                scanf("%d",&a[i]);
        }
        quick(a,0,4);
        for(int i=0;i<5;i++)
        {
                printf("%d ",a[i]);
        }
}
```

INPUT AND OUTPUT :

```
Enter Elements:4 14 7 2 1
1 2 4 7 14
```

RESULT :

The C Program for Implementing Quick Sort is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-4 (Data-Structures) Lab Session

EXP - 9

TITLE : C Program to Implement AVL Tree

PROGRAM :

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

struct Node {
  int key;
  struct Node *left;
  struct Node *right;
  int height;
};

int max(int a, int b);

int height(struct Node *N) {
  if (N == NULL)
    return 0;
  return N->height;
}

int max(int a, int b) {
  return (a > b) ? a : b;
}

struct Node *newNode(int key) {
  struct Node *node = (struct Node *)
    malloc(sizeof(struct Node));
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}
```

```
struct Node *rightRotate(struct Node *y) {
  struct Node *x = y->left;
  struct Node *T2 = x->right;

  x->right = y;
  y->left = T2;

  y->height = max(height(y->left), height(y->right)) + 1;
  x->height = max(height(x->left), height(x->right)) + 1;

  return x;
}

struct Node *leftRotate(struct Node *x) {
  struct Node *y = x->right;
  struct Node *T2 = y->left;

  y->left = x;
  x->right = T2;

  x->height = max(height(x->left), height(x->right)) + 1;
  y->height = max(height(y->left), height(y->right)) + 1;

  return y;
}

int getBalance(struct Node *N) {
  if (N == NULL)
    return 0;
  return height(N->left) - height(N->right);
}
struct Node *insertNode(struct Node *node, int key) {
  if (node == NULL)
    return (newNode(key));

  if (key < node->key)
```

```
node->left = insertNode(node->left, key);
  else if (key > node->key)
    node->right = insertNode(node->right, key);
  else
    return node;

  node->height = 1 + max(height(node->left),
              height(node->right));

  int balance = getBalance(node);
  if (balance > 1 && key < node->left->key)
    return rightRotate(node);

  if (balance < -1 && key > node->right->key)
    return leftRotate(node);

  if (balance > 1 && key > node->left->key) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
  }

  if (balance < -1 && key < node->right->key) {
    node->right = rightRotate(node->right);
    return leftRotate(node);
  }

  return node;
}
struct Node *minValueNode(struct Node *node) {
  struct Node *current = node;

  while (current->left != NULL)
    current = current->left;

  return current;
`
```

```
struct Node *deleteNode(struct Node *root, int key) {
  if (root == NULL)
    return root;

  if (key < root->key)
    root->left = deleteNode(root->left, key);

  else if (key > root->key)
    root->right = deleteNode(root->right, key);

  else {
    if ((root->left == NULL) || (root->right == NULL)) {
      struct Node *temp = root->left ? root->left : root->right;

      if (temp == NULL) {
        temp = root;
        root = NULL;
      } else
        *root = *temp;
      free(temp);
    } else {
      struct Node *temp = minValueNode(root->right);

      root->key = temp->key;

      root->right = deleteNode(root->right, temp->key);
    }
  }

  if (root == NULL)
    return root;
```

```
  return root;

  root->height = 1 + max(height(root->left),
              height(root->right));

  int balance = getBalance(root);
  if (balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(root);

  if (balance > 1 && getBalance(root->left) < 0) {
    root->left = leftRotate(root->left);
    return rightRotate(root);
  }

  if (balance < -1 && getBalance(root->right) <= 0)
    return leftRotate(root);

  if (balance < -1 && getBalance(root->right) > 0) {
    root->right = rightRotate(root->right);
    return leftRotate(root);
  }

  return root;
}

void printPreOrder(struct Node *root) {
  if (root != NULL) {
    printf("%d ", root->key);
    printPreOrder(root->left);
    printPreOrder(root->right);
  }
}
int main() {
  struct Node *root = NULL;
  root = insertNode(root, 2);
```

```
    root = insertNode(root, 7);
    root = insertNode(root, 4);
    root = insertNode(root, 5);
    root = insertNode(root, 3);
    root = insertNode(root, 8);

    printPreOrder(root);

    root = deleteNode(root, 3);

    printf("\nAfter deletion: ");
    printPreOrder(root);

    return 0;
}
```

INPUT AND OUTPUT :

```
4 2 1 3 7 5 8
After deletion: 4 2 1 7 5 8
```

RESULT :

The C Program for Implementing AVLTree is Compiled and Executed Using Dev-C++ and the Output is Verified.