SRINIVAS UNIVERSITY
INSTITUTE OF ENGINEERING AND TECHNOLOGY
MUKKA, MANGALURU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL FOR DATA STRUCTURE-I- LABORATORY 2024-25

SUBJECT CODE: 23SCS032L

PREPARED BY:

Dr. J. Venkata Krishna
Associate Professor

Name:___________________________________

SECTION/ Branch:_______________________

Semester:________________________________

USN:______________________________________

SRINIVAS UNIVERSITY
INSTITUTE OF ENGINEERING AND TECHNOLOGY
MUKKA, MANGALURU
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

III Semester B. Tech

DATA STRUCTURE-I-LABORATORY

Sub Code			**23SCS032L**	
TEACHING HOURS/ WEEK			**IA Marks**	**25**
		Practical	**Exam Marks**	**25**
		08	**Exam Hours**	**03**
Credits : 2				

List of Experiments:

Sr.No	**Experiment Details**
1	Design, Develop and Implement a menu-driven Program in C for the following Array operations: a. Creating an Array of N Integer Elements b. Display of Array Elements with Suitable Headings c. Inserting an Element (ELEM) at a given valid Position (POS) d. Deleting an Element at a given valid Position (POS) e. Exit. Support the program with functions for each of the above operations.
2	Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX) a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate Overflow and Underflow situations on Stack d. Display the status of Stack e. Exit Support the program with appropriate functions for each of the above Operations.
3	Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. The program should support both parenthesized and free parenthesized expressions with the operators: +, -, *,/, %(Remainder), ^(Power), and alphanumeric operands.

4	Design, Develop and Implement a Program in C for the following Stack Applications, Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^
5	Design, Develop and Implement a Program in C for the following Stack Applications, Solving Tower of Hanoi problem with n disks.
6	Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX) a. Insert an Element on to Circular QUEUE b. Delete an Element from Circular QUEUE c. Demonstrate Overflow and Underflow situations on Circular QUEUE d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above Operations.
7	Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo a. Create a SLL of N Students Data by using front insertion. b. Display the status of SLL and count the number of nodes in it c. Perform Insertion and Deletion at End of SLL d. Perform Insertion and Deletion at Front of SLL e. Exit.
8	Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo a. Create a DLL of N Employees Data by using end insertion. b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue f. Exit
9	Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers

	a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2 b. Traverse the BST in Inorder, Preorder and Post Order c. Search the BST for a given element (KEY) and report the appropriate message d. Exit

COURSE OBJECTIVES:

This course will enable students to

- Understand the working of Pointers and their implementation in an Arrays, Structures and Unions.
- Understand the working of various linear data structures such as stacks, queues, lists.
- Analyze nonlinear data structures: trees

COURSE OUTCOMES:

Course Outcomes: After studying this course, students will be able to:

CO-1: Understand the types of data structures, operations, and algorithms and apply searching and sorting operations on files.

CO-2: Analyze the computational complexity of different algorithms using asymptotic analysis.

CO-3: Apply the greedy paradigm and dynamic programming and explain when an algorithmic design situation calls for it.

Scheme of Examination:

1. All laboratory experiments are to be included for practical examination.
2. Students are allowed to pick one experiment from the lot.
3. Strictly follow the instructions as printed on the cover page of answer script
4. Marks distribution: Procedure + Conduction + Viva:12 + 28 +10 = 50 and Scale to 25
5. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero

LABORATORYGUIDELINES

1. Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.

2. Students are instructed to come to lab in formal dresses only.

3. Students are supposed to occupy the systems allotted to them and are not supposed to talk or make noise in the lab.

4. Students are required to carry their observation book and lab records with completed exercises while entering the lab.

5. Lab records need to be submitted every week.

6. Students are not supposed to use pen drives in the lab.

Do's

1. Come with completed observation and record
2. Wear ID card before entering into the lab.
3. Write in time, out time and system details in the login register.
4. Remove your shoes or wear foot socks before you enter the lab.
5. Do not touch any part of the computer with wet hands.
6. Shut down the computer properly.
7. Read and understand how to carry out an activity thoroughly before coming to the laboratory.
8. Report any broken plugs or exposed electrical wires to your lecturer/laboratory technician immediately.
9. Be on time

Don'ts

1. Do not eat or drink in the laboratory.
2. Do not operate mobile phones in the lab.
3. Do not change system settings.
4. Do not disturb your neighboring students. They may be busy in completing tasks.
5. Do not remove anything from the computer laboratory without permission.
6. Do not use pen drives.
7. Do not misbehave.
8. Do not switch off the computer directly.

	d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above Operations.	
7	Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo a. Create a SLL of N Students Data by using front insertion. b. Display the status of SLL and count the number of nodes in it c. Perform Insertion and Deletion at End of SLL d. Perform Insertion and Deletion at Front of SLL e. Exit.	
8	Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo a. Create a DLL of N Employees Data by using end insertion. b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue f. Exit	
9	Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2 b. Traverse the BST in Inorder, Preorder and Post Order c. Search the BST for a given element (KEY) and report the appropriate message d. Exit	

1. Design, Develop and Implement a menu driven Program in C for the following Array Operations.

a. Creating an Array of N Integer Elements
b. Display of Array Elements with Suitable Headings
c. Inserting an Element (ELEM) at a given valid Position (POS)
d. Deleting an Element at a given valid Position(POS)
e. Exit.
Support the program with functions for each of the above operations.

```
#include<stdio.h>
#include<stdlib.h>
#define size 20

void insert(int arr[], int *n);
void delete(int arr[], int *n);
void display(int arr[], int n);

void main()
{
	int arr[size], i, n, ch;
	printf("Enter Initial size of Array\n");
	scanf("%d", &n);
	printf("Enter Array Elements\n");
	for(i=0; i<n; i++)
		scanf("%d", &arr[i]);
	for(;;)
	{
		printf("\nMenu\n");
		printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
		scanf("%d", &ch);
		switch(ch)
		{
			case 1:
				insert(arr, &n);
				break;
			case 2:
				delete(arr, &n);
				break;
			case 3:
				display(arr, n);
				break;
			case 4:
				exit(0);
```

```
                    }
            }
    }

    void insert(int arr[], int *n)
    {
            int i, pos, ele;
            printf("Enter Element\n");
            scanf("%d", &ele);
            printf("Enter Position\n");
            scanf("%d", &pos);
            for(i=(*n)-1; i>=pos-1; i--)
            {
                    arr[i+1] = arr[i];
            }
            arr[pos-1] = ele;
            *n = *n + 1;
    }

    void delete(int arr[], int *n)
    {
            int i, pos;
            printf("Enter the Position to delete the element\n");
            scanf("%d", &pos);
            for(i=pos-1; i<(*n)-1; i++)
            {
                    arr[i] = arr[i+1];
            }
            *n = *n - 1;
    }

    void display(int arr[], int n)
    {
            int i;
            for(i=0; i<n; i++)
                    printf("%d ", arr[i]);
            printf("\n");
    }
```

2. Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

a. Push an Element on to Stack
b. Pop an Element from Stack
c. Demonstrate Overflow and Underflow situations on Stack
d. Display the status of Stack
e. Exit
Support the program with appropriate functions for each of the above Operations.

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 20

int stack_full(int top);
int stack_empty(int top);
void push(int stack[], int *top, int ele);
int pop(int stack[], int *top);
void display(int stack[], int top);

void main()
{
        int stack[MAX], top = -1, ele, ch;
        for(;;)
        {
                printf("\nMenu\n");
                printf("1. Push\n2. Pop \n 3. Stack Status\n 4. Display\n 5. Exit\n");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1:
                                if(stack_full(top))
                                        printf("Stack Full\n");
                                else
                                {
                                        printf("Enter an Element\n");
                                        scanf("%d", &ele);
                                        push(stack, &top, ele);
                                }
                                break;
                        case 2:
                                if(stack_empty(top))
                                        printf("Stack Empty\n");
                else
                {
                    ele = pop(stack, &top);
                    printf("Deleted Element is %d", ele);
                }
                break;
            case 3:
                if(stack_empty(top))
                    printf("Stack Empty\n");
                else if(stack_full(top))
                    printf("Stack Full\n");
                else
                    printf("Stack contains %d elements\n", top+1);
                break;
            case 4:
                if(stack_empty(top))
                    printf("Stack Empty\n");
                else
                    display(stack, top);
                break;
            case 5:
                exit(0);
        }
    }
}

int stack_full(int top)
{
    if(top == MAX-1)
        return 1;
    return 0;
}

int stack_empty(int top)
{
    if(top == -1)
        return 1;
    return 0;
}

void push(int stack[], int *top, int ele)
{
    stack[++(*top)] = ele;
}
```

```
int pop(int stack[], int *top)
{
        return stack[(*top)--];
}

void display(int stack[], int top)
{
        int i;
        for(i=0; i<= top; i++)
        {
                printf("%d ", stack[i]);
        }
        printf("\n");
}
```

3. Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *,/, %(Remainder), ^(Power) and alphanumeric operands.

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

typedef enum{lparen, rparen, plus, minus, mul, divi, mod, pwr, eos, operand} precedence;

precedence getToken(char infix[], char *symbol, int *n);
void convert(char infix[]);
void push(int stack[], int *top, precedence token);
precedence pop(int stack[], int *top);
void printToken(precedence token);

void main()
{
        char infix[30];
        printf("Enter Infix Expression\n");
        gets(infix);
        convert(infix);
}

precedence getToken(char infix[], char *symbol, int *n)
{
        *symbol = infix[(*n)++];
        switch(*symbol)
        {
                case '(':return lparen;
                case ')':return rparen;
                case '+':return plus;
                case '-':return minus;
                case '*':return mul;
                case '/': return divi;
                case '%':return mod;
                case '^':return pwr;
                case '\0':return eos;
                default:return operand;
        }
}
```

```
void convert(char infix[])
{
        int stack[20], top = 0, n=0;
        int icp[] = {5, 4, 1, 1, 2, 2, 2, 3, 0};
        int isp[] = {0, 4, 1, 1, 2, 2, 2, 3, 0};
        precedence token;
        char symbol;
        stack[0] = eos;
        for(token = getToken(infix, &symbol, &n); token != eos; token = getToken(infix,
&symbol, &n))
        {
                if(token == operand)
                        printf("%c", symbol);
                else if(token == rparen)
                {
                        while(stack[top] != lparen)
                        {
                                printToken(pop(stack, &top));
                        }
                        pop(stack, &top);
                }
                else
                {
                        while(isp[stack[top]] >= icp[token])
                        {
                                printToken(pop(stack, &top));
                        }
                        push(stack, &top, token);
                }
        }
        while(stack[top] != eos)
        {
                printToken(pop(stack, &top));
        }
}

void push(int stack[], int *top, precedence token)
{
        stack[++(*top)] = token;
}

precedence pop(int stack[], int *top)
{
        return stack[(*top)--];
}
void printToken(precedence token)
{
        switch(token)
        {
                case plus:
                        printf("+");
                        break;
                case minus:
                        printf("-");
                        break;
                case mul:
                        printf("*");
                        break;
                case divi:
                        printf("/");
                        break;
                case mod:
                        printf("%c", '%');
                        break;
                case pwr:
                        printf("^");
                        break;
        }
}
```

}

4. Design, Develop and Implement a Program in C for the following Stack Applications
Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^

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

typedef enum {plus, minus, mul, divi, mod, pwr, eos, operand} precedence;

precedence getToken(char postfix[], int *n, char *symbol);
int eval(char postfix[]);
void push(int stack[], int *top, int ele);
int pop(int stack[], int *top);
int operation(int opr1, int opr2, precedence token);

void main()
{
        char postfix[30];
        printf("Enter Postfix Expression\n");
        gets(postfix);
        printf("The Result after Evaluation is %d", eval(postfix));
}

precedence getToken(char postfix[], int *n, char *symbol)
{
        *symbol = postfix[(*n)++];
        switch(*symbol)
        {
                case '+':return plus;
                case '-':return minus;
                case '*':return mul;
                case '/':return divi;
                case '%':return mod;
                case '^':return pwr;
                case '\0':return eos;
                default:return operand;
        }
}
```

```
int eval(char postfix[])
{
	int stack[20], top=-1, n=0, opr1, opr2;
	precedence token;
	char symbol;
	for(token = getToken(postfix, &n, &symbol); token != eos; token = 
getToken(postfix, &n, &symbol))
	{
		if(token == operand)
		{
			push(stack, &top, symbol - '0');
		}
		else
		{
			opr2 = pop(stack, &top);
			opr1 = pop(stack, &top);
			push(stack, &top, operation(opr1, opr2, token));
		}
	}
	return stack[0];
}

void push(int stack[], int *top, int ele)
{
	stack[++(*top)] = ele;
}

int pop(int stack[], int *top)
{
	return stack[(*top)--];
}

int operation(int opr1, int opr2, precedence token)
{
	switch(token)
	{
		case plus:return opr1+opr2;
		case minus:return opr1-opr2;
		case mul:return opr1*opr2;
		case divi:return opr1/opr2;
		case mod:return opr1%opr2;
		case pwr:return pow(opr1, opr2);
	}
}
```

5. Design, Develop and Implement a Program in C for the following Stack Applications, Solving Tower of Hanoi problem with n disks.

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

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod);

void main()
{
        int n;
        printf("Enter the number of disks\n");
        scanf("%d", &n);
        towerOfHanoi(n, 'A', 'C', 'B');
        return;
}

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
        if(n == 1)
        {
                printf("Move disk 1 from rod %c to rod %c\n", from_rod, to_rod);
                return;
        }

        towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
        printf("Move disk %d from rod %c to rod %c\n", n, from_rod, to_rod);
        towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
        return;
}
```

6. Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit
Support the program with appropriate functions for each of the above operations

```
#include<stdio.h>
#include<stdlib.h>
#define size 10

void cqinsert(char queue[], int *rear, int *count);
void cqdelete(char queue[], int *front, int *count);
void display(char queue[], int front, int count);

void main()
{
	char queue[size];
	int front = 0, rear = -1, ch, count = 0;
	for(;;)
	{
		printf("\nMenu\n");
		printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
		scanf("%d", &ch);
		switch(ch)
		{
			case 1:
				if(count == size)
					printf("Queue is Full\n");
				else
					cqinsert(queue, &rear, &count);
				break;
			case 2:
				if(count == 0)
					printf("Queue is Empty\n");
				else
					cqdelete(queue, &front, &count);
				break;
			case 3:
				if(count == 0)
```

```
                                        printf("Queue is Empty\n");
                                else
                                        display(queue, front, count);
                                break;
                        case 4:
                                exit(0);
                }
        }
}

void cqinsert(char queue[], int *rear, int *count)
{
        char ele;
        printf("Enter a Character\n");
        scanf(" %c", &ele);
        *rear = (*rear + 1) % size;
        queue[*rear] = ele;
        (*count)++;
}

void cqdelete(char queue[], int *front, int *count)
{
        printf("Deleted Character is %c\n", queue[*front]);
        *front = (*front + 1) % size;
        (*count)--;
}

void display(char queue[], int front, int count)
{
        int i;
        printf("The Characters are\n");
        for(i=0; i<count; i++)
        {
                printf("%c\t", queue[front]);
                front = (front + 1) % size;
        }
        printf("\n");
}
```

7. Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo

a. Create a SLL of N Students Data by using front insertion.

b. Display the status of SLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of SLL
d. Perform Insertion and Deletion at Front of SLL
e. Demonstrate how this SLL can be used as STACK
f. Exit

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

typedef struct node
{
	char usn[10];
	char name[15];
	char branch[2];
	int sem;
	long int phno;
	struct node *link;
} NODE;

typedef struct head_node
{
	int count;
	struct node *link;
} HEAD;

NODE *getNode();
void insfront(HEAD *head);
void insrear(HEAD *head);
void delfront(HEAD *head);
void delrear(HEAD *head);
void display(HEAD *head);

void main()
{
	HEAD *head = (HEAD *) malloc(sizeof(HEAD));
	int ch;
	head->count = 0;
	head->link = NULL;
	for(;;)
	{
	printf("\nMenu\n");
	printf("1. Insert Front\n2. Insert Rear\n3. Delete Front\n4. Delete Rear\n5. Display\n6.
Exit\n");
	printf("For Stack Choose 2,4 and 5 options\n");
```

```
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1:
                                insfront(head);
                                break;
                        case 2:
                                insrear(head);
                                break;
                        case 3:
                                if(head->link == NULL)
                                printf("List Empty\n");
                                else
                                        delfront(head);
                                break;
                        case 4:
                                if(head->link == NULL)
                                printf("List Empty\n");
                                else
                                        delrear(head);
                                break;
                        case 5:
                                if(head->link == NULL)
                                printf("List Empty\n");
                                else
                                        display(head);
                                break;
                        case 6:
                                exit(0);
                }
        }
}

NODE *getNode()
{
        NODE *temp = (NODE *) malloc(sizeof(NODE));
        if(temp == NULL)
        {
                printf("No Memory\n");
                exit(0);
        }
        return temp;
}
```

```
void insfront(HEAD *head)
{
        NODE *new = getNode();
        (head->count)++;
        printf("Enter Details such as USN Name Branch Semester PhNo\n");
scanf("%s%s%s%d%ld", (new->usn), (new->name), (new->branch), &(new->sem), &(new-
>phno));
        new->link = head->link;
        head->link = new;
        return;
}

void insrear(HEAD *head)
{
        NODE *new = getNode();
        NODE *temp = head->link;
        (head->count)++;
printf("Enter Details such as USN Name Branch Semester PhNo\n");
scanf("%s%s%s%d%ld", (new->usn), (new->name), (new->branch), &(new->sem), &(new-
>phno));
        new->link = NULL;
        if(temp == NULL)
        {
                head->link = new;
                return;
        }
        while(temp->link != NULL)
        {
                temp = temp->link;
        }
        temp->link = new;
        return;
}

void delfront(HEAD *head)
{
        NODE *temp = head->link;
        (head->count)--;
printf("Deleted Record is \n");
printf("%s\t%s\t%s\t%d\t%ld\n", (temp->usn), (temp->name), (temp->branch), (temp->sem),
(temp->phno));
        head->link = temp->link;
        return;
}
```

```
void delrear(HEAD *head)
{
        NODE *previous = NULL, *present = head->link;
        (head->count)--;
        if(present->link == NULL)
        {
                head->link = NULL;
        }
        else
        {
                while(present->link != NULL)
                {
                        previous = present;
                        present = present->link;
                }
                previous->link = NULL;
        }
printf("Deleted Record is \n");
printf("%s\t%s\t%s\t%d\t%ld\n",    (present->usn),    (present->name),    (present->branch),
(present->sem), (present->phno));
        free(present);
        return;
}

void display(HEAD *head)
{
        NODE *temp = head->link;
        printf("Total Records are %d\n", head->count);
        printf("USN\tName\t\tBranch\tSem\tPhNo\n");
        while(temp != NULL)
        {
printf("%s\t%s\t\t%s\t%d\t%ld\n",   (temp->usn),   (temp->name),   (temp->branch),   (temp-
>sem), (temp->phno));
                temp = temp->link;
        }
}
```

8. Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo

a. Create a DLL of N Employees Data by using end insertion.
b. Display the status of DLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of DLL
d. Perform Insertion and Deletion at Front of DLL
e. Demonstrate how this DLL can be used as Double Ended Queue
f. Exit

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

typedef struct node
{
        char ssn[20], name[20], department[20], designation[20];
        float sal;
        long int phno;
        struct node *llink, *rlink;
} NODE;

typedef struct headnode
{
        int count;
        struct node *llink, *rlink;
} HEAD;

void insfront(HEAD *head);
void insrear(HEAD *head);
void delfront(HEAD *head);
void delrear(HEAD *head);
void display(HEAD *head);
NODE *getNode();

void main()
{
        int ch;
        HEAD *head = (HEAD *) malloc(sizeof(HEAD));
        head->count = 0;
        head->llink = NULL;
        head->rlink = NULL;
        for(;;)
        {
                printf("\n\nMenu\n");
                printf("\n1. Insert Front\n2. Insert Rear\n3. Delete Front\n4. Delete Rear\n5.
Display\n6. Exit\n");
                scanf("%d", &ch);
```

```
                    switch(ch)
                    {
                              case 1:
                                        insfront(head);
                                        break;
                              case 2:
                                        insrear(head);
                                        break;
                              case 3:
                                        if(head->rlink == NULL)
                                                  printf("List Empty");
                                        else
                                                  delfront(head);
                                        break;
                              case 4:
                                        if(head->rlink == NULL)
                                                  printf("List Empty");
                                        else
                                                  delrear(head);
                                        break;
                              case 5:
                                        if(head->rlink == NULL)
                                                  printf("List Empty");
                                        else
                                                  display(head);
                                        break;
                              case 6:
                                        exit(0);
                    }
          }
}

NODE *getNode()
{
          NODE *temp = (NODE *) malloc(sizeof(NODE));
          if(temp == NULL)
          {
                    printf("No Memory\n");
                    exit(0);
          }
          return temp;
}

void insfront(HEAD *head)
```

```
{
        NODE *new = getNode();
        NODE *next = head->rlink;
        printf("Enter Details such as SSN Name Department Designation Salary PhNo\n");
        scanf("%s%s%s%s%f%ld", (new->ssn), (new->name), (new->department), (new-
>designation), &(new->sal), &(new->phno));
        if(next != NULL)
                next->llink = new;
        new->rlink = next;
        head->rlink = new;
        (head->count)++;
}

void insrear(HEAD *head)
{
        NODE *new = getNode();
        NODE *temp = NULL;
        printf("Enter Details such as SSN Name Department Designation Salary PhNo\n");
        scanf("%s%s%s%s%f%ld", (new->ssn), (new->name), (new->department), (new-
>designation), &(new->sal), &(new->phno));
        (head->count)++;
        new->rlink = NULL;
        if(head->rlink == NULL)
        {
                head->rlink = new;
                return;
        }
        temp = head->rlink;
        while(temp->rlink != NULL)
                temp = temp->rlink;
        temp->rlink = new;
        new->llink = temp;
}

void delfront(HEAD *head)
{
        NODE *temp = head->rlink;
        printf("Deleted Record is\n");
        printf("%s\t%s\t%s\t%s\t%f\t%ld\n", (temp->ssn), (temp->name), (temp-
>department), (temp->designation), (temp->sal), (temp->phno));
        head->rlink = temp->rlink;
        free(temp);
        (head->count)--;
}
```

```
void delrear(HEAD *head)
{
        NODE *previous = NULL, *present = head->rlink;
        if(present->rlink == NULL)
        {
                head->rlink = NULL;
        }
        else
        {
                while(present->rlink != NULL)
                {
                        previous = present;
                        present =  present->rlink;
                }
                previous->rlink = NULL;
        }
        printf("Deleted Record is\n");
        printf("%s\t%s\t%s\t%s\t%f\t%ld\n",    (present->ssn),    (present->name),    (present-
>department), (present->designation), (present->sal), (present->phno));
        (head->count)--;
        free(present);
}

void display(HEAD *head)
{
        NODE *temp = head->rlink;
        printf("Total Number of records are %d\n", head->count);
        printf("SSN\tName\tDepartment\tDesignation\tSalary\t\tPhNo\n");
        while(temp != NULL)
        {
                printf("%s\t%s\t%s\t\t%s\t\t%f\t%ld\n",  (temp->ssn),  (temp->name),  (temp-
>department), (temp->designation), (temp->sal), (temp->phno));
                temp = temp->rlink;

        }
}
```

9. Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers

a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate message
d. Delete an element(ELEM) from BST
e. Exit

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

typedef struct t_node
{
        char data;
        struct t_node *llink;
        struct t_node *rlink;
} TNODE;

void inorder(TNODE *root);
void preorder(TNODE *root);
void postorder(TNODE *root);
int search(TNODE *root, int key);
TNODE *getnode();
TNODE *insert(TNODE *root, int ele);
TNODE *delete(TNODE *root, int ele);
TNODE *findMin(TNODE *root);

void main()
{
        TNODE *root=NULL;
        int ch, flag;
        int ele, key;
        for(;;)
        {

        printf("\n1.Insert\n2.Inorder\n3.Preorder\n4.Postorder\n5.Search\n6.Delete\n7.Exit\n"
);
                printf("\nEnter your choice\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                printf("Enter the element to be inserted\n");
                                scanf("%d",&ele);
                                root=insert(root,ele);
                                break;

                        case 2:
                                if(root==NULL)
                                        printf("Empty tree\n");
                                else
                                {
                                        printf("The contents are\n");
                                        inorder(root);
                                }
                                break;

                        case 3:
                                if(root==NULL)
                                        printf("Empty tree\n");
                                else
                                {
                                        printf("The contents are\n");
                                        preorder(root);
                                }
                                break;

                        case 4:
                                if(root==NULL)
                                        printf("Empty tree\n");
                                else
                                {
                                        printf("The contents are\n");
                                        postorder(root);
                                }
                                break;

                        case 5:
                                printf("Enter the element to be searched\n");
                                scanf("%d",&key);
                                flag=search(root, key);
                                if(flag)
                                        printf("Found\n");
                                else
                                        printf("Not Found\n");
                                break;
```

```
                    case 6:
                        printf("Enter the element to be deleted\n");
                        scanf("%d", &ele);
                        root = delete(root, ele);
                        break;

                    case 7: exit(0);
                }
        }
}

TNODE *getnode()
{
        TNODE *temp=(TNODE *)malloc(sizeof(TNODE));
        if(temp==NULL)
        {
                printf("No Memory\n");
                exit(0);
        }
        return temp;
}

TNODE *insert(TNODE *root,int ele)
{
        TNODE *new=NULL;
        if(root==NULL)
        {
                new = getnode();
                new->data=ele;
                new->rlink=new->llink=NULL;
                return new;
        }
        if(ele>(root->data))
                root->rlink=insert(root->rlink,ele);
        if(ele<(root->data))
                root->llink=insert(root->llink,ele);
        return root;
}

void inorder(TNODE *root)
{
        if(root!=NULL)
        {
                inorder(root->llink);
                printf("%d ",root->data);
                inorder(root->rlink);
        }
}

void preorder(TNODE *root)
{
        if(root!=NULL)
        {
                printf("%d ",root->data);
                preorder(root->llink);
                preorder(root->rlink);
        }
}

void postorder(TNODE *root)
{
        if(root!=NULL)
        {
                postorder(root->llink);
                postorder(root->rlink);
                printf("%d ",root->data);
        }
}

int search(TNODE *root,int key)
{
        if(root!=NULL)
        {
                if(key==root->data)
                        return 1;
                if(key>root->data)
                        return(search(root->rlink,key));
                if(key<root->data)
                        return(search(root->llink,key));
        }
        return 0;
}

TNODE *delete(TNODE *root, int ele)
{
        TNODE *temp = NULL;
        if(root == NULL)
```

```
                printf("Element not found\n");
        else if(ele< root->data)
                root->llink = delete(root->llink, ele);
        else if(ele> root->data)
                root->rlink = delete(root->rlink, ele);
        else
        {
                if(root->llink != NULL && root->rlink != NULL)
                {
                        temp = findMin(root->rlink);
                        root->data = temp->data;
                        root->rlink = delete(root->rlink, temp->data);
                }
                else
                {
                        temp = root;
                        if(root->llink == NULL)
                                root = root->rlink;
                        else if(root->rlink == NULL)
                                root = root->llink;
                        free(temp);
                }
        }
        return root;
}

TNODE *findMin(TNODE *root)
{
        if(root == NULL)
                return NULL;

        if(root->llink != NULL)
                return findMin(root->llink);
        else
                return root;
}
```