EXPERIMENT – 08

Design, Develop and Implement Program in C to Reverse a Singly Linked List (SLL) of a given integers.

Program Code:

```
#include<stdio.h>
#include<malloc.h>
struct node
{
        int value;
        struct node *next;
};
int main()
{
        int item,n;
        struct node * head;
        struct  node * tail;
        struct  node * temp;
        struct  node * prev;
        struct  node * current;
        struct  node * next;
        temp=(struct node*)malloc(sizeof(struct node));
        head=temp;
        printf("enter the size of list\n");
        scanf("%d",&n);
        printf("enter the list to be reversed\n");
        temp = (struct node *)malloc(sizeof(struct node));
         scanf("%d",&item);
        temp->value = item;
        head = temp;
        n--;
        while(n!=0)
        {
                temp->next = (struct node *)malloc(sizeof(struct node));
                 temp = temp->next;
                 scanf("%d",&item);
                temp->value = item;
                 --n;
        }
        temp->next = NULL;
        tail = temp;
        temp = head;
```

```
    while(temp)
    {
        printf(" %d\n", temp->value);
        temp = temp->next;
    }
    printf("Reversing the linked list\n");
    prev = NULL;
    current = next = head;
    while(current)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    temp = tail;
    while(temp)
    {
        printf(" %d\n", temp->value);
        temp = temp->next;
    }
    return 0;
}
```

OUTPUT:

```
Enter the size of list 4
enter the list to be reversed  1 1 1 2
Reversing the linked list      2 1 1 1
```

EXPERIMENT – 09

Design, Develop and Implement a menu driven Program in C for the following operations on Priority queue

a. Create a Priority queue by using Insert function
b. Insertion data and priority values as input
c. Perform Deletion operation
d. Display the elements of Priority queue

Program Code:

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

typedef struct node
        {
        int priority;
        int info;
        struct node *link;
}NODE;
NODE *front = NULL;

void insert(int item,int priority)
{
        NODE *tmp,*q;
        tmp = (NODE *)malloc(sizeof(NODE));
        tmp->info = item;
        tmp->priority = priority;
        if( front == NULL || priority < front->priority )
        {
                tmp->link = front;
                front = tmp;
        }
        else
        {
                q = front;
                while( q->link != NULL && q->link->priority <= priority )
                q=q->link;
                tmp->link = q->link;
                q->link = tmp;
        }
}
```

```
void del()
{
	NODE *tmp;
	if(front == NULL)
		printf("Queue Underflow\n");
	else
	{
		tmp = front;
		printf("Deleted item is %d\n",tmp->info);
		front = front->link;
		free(tmp);
	}
}

void display()
{
	NODE *ptr;
	ptr = front;
	if(front == NULL)
		printf("Queue is empty\n");
	else
	{
		printf("Queue is :\n");
		printf("Priority	Item\n");
		while(ptr != NULL)
		{
			printf("%5d	%5d\n",ptr->priority,ptr->info);
			ptr = ptr->link;
		}
	}
}

int main()
{
	int choice,item,priority;
	do
	{
		printf("1.Insert\n");
		printf("2.Delete\n");
		printf("3.Display\n");
		printf("4.Quit\n");
		printf("Enter your choice : ");
		scanf("%d", &choice);
		switch(choice)
```

```
        {
            case 1:
                printf("Input the item value to be added in the queue : ");
                scanf("%d",&item);
                printf("Enter its priority : ");
                scanf("%d",&priority);
                insert(item,priority);
                break;
            case 2:
                del();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
                break;
                default :
                printf("Wrong choice\n");
        }
    }while(choice!=4);
    return 0;
}
```

OUTPUT:

```
1.insert
2.delete
3.display
4.exit
enter your choice: 1
input the item values to be added in the queue: 1 2 3
enter its priority: 1

1.insert
2.delete
3.display
4.exit
enter your choice: 2
Deleted item is 1 2 3

1.insert
2.delete
3.display
4.exit
enter your choice: 3
queue is empty

1.insert
2.delete
3.display
4.exit
enter your choice: 4
```

EXPERIMENT – 10

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,
c. Traverse the BST in Preorder
d. Traverse the BST in Post Order

Program Code:

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

struct node
{
        int value;
        struct  node *left;
        struct node *right;
};


struct node *root;
struct node* insert(struct node* r, int data);
void inOrder(struct node* r);
void preOrder(struct node* r);
void postOrder(struct node* r);

int main()
{
        root = NULL;
        int n, v;
        printf("How many data's do you want to insert ?\n");
        scanf("%d", &n);
        for(int i=0; i<n; i++)
        {
                printf("Data %d: ", i+1);
                scanf("%d", &v);
                root = insert(root, v);
        }
        printf("Inorder Traversal: ");
        inOrder(root);
        printf("\n");
        printf("Preorder Traversal: ");
         preOrder(root);
```

```
	printf("\n");
	 printf("Postorder Traversal: ");
	postOrder(root);
	printf("\n");
	return 0;
}

struct node* insert(struct node* r, int data)
{
	if(r==NULL)
	{
		r = (struct node*) malloc(sizeof(struct node));
		r->value = data;
		r->left = NULL;
		r->right = NULL;
	}
	else if(data < r->value)
	{
		r->left = insert(r->left, data);
	}
	else
	{
		r->right = insert(r->right, data);
	}
	return r;

}

void inOrder(struct node* r)
{
	if(r!=NULL)
	{
		inOrder(r->left);
		printf("%d ", r->value);
		inOrder(r->right);
	}
}
void preOrder(struct node* r)
{
	if(r!=NULL)
	{
		printf("%d ", r->value);
		preOrder(r->left);
		preOrder(r->right);
	}
```

```
}

void postOrder(struct node* r)
{
        if(r!=NULL)
        {
                postOrder(r->left);
                postOrder(r->right);
                printf("%d ", r->value);
        }
}
```

OUTPUT:

```
How many data's do you want to insert?   5
data 1: 3
data 2: 7
data 3: 2
data 4: 8
data 5: 1
inorder traversal: 1 2 3 7 8
preorder traversal: 3 2 1 7 8
postorder traversal: 1 2 8 7 3
```

EXPERIMENT – 11

Given a File of N employee records with a set K of Keys(4- digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table(HT) of m memory locations with L as the set of memory addresses (2- digit) of locations in HT. Let the keys in K and Addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K ®L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing

Program Code:

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

struct employee
{
        int id;
        char name[15];
};
typedef struct employee EMP;
EMP emp[MAX];
int a[MAX];
int create(int num)
{
        int key;
        key = num % 100;
        return key;
}
int getemp(EMP emp[],int key)
{
        printf("\nEnter emp id: ");
        scanf("%d",&emp[key].id);
        printf("\nEnter emp name: ");
        flushall();
        gets(emp[key].name);
        return key;
}
void display()
{
        int i, ch;
        printf("\n1.Display ALL\n2.Filtered Display");
```

```
        printf("\nEnter the choice: ");
        scanf("%d",&ch);
        if(ch == 1)
        {
                printf("\nThe hash table is:\n");
                printf("\nHTKey\tEmpID\tEmpName");
                for(i=0; i<MAX; i++)
                printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
        }
   else
        {
                printf("\nThe hash table is:\n");
                printf("\nHTKey\tEmpID\tEmpName");
                for(i=0; i<MAX; i++)
                if(a[i] != -1)
                {
                        printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
                        continue;
                }
        }
}

void linear_prob(int key, int num)
{
        int flag, i, count = 0; flag = 0;
        if(a[key] == -1)
        {
                a[key]=getemp(emp, key);
        }
        else
        {
                printf("\nCollision Detected...!!!\n");
                i = 0;
                while(i < MAX)
                {
                        if (a[i] != -1)
                        count++;
                        else
                        i++;
                }
                printf("\nCollision avoided successfully using LINEAR PROBING\n");
                if(count == MAX)
                {
                        printf("\n Hash table is full");
                        display(emp);
```

```
                exit(1);
        }
        for(i=key; i<MAX; i++)
        if(a[i] == -1)
        {
                a[i] = num;
                flag = 1;
                break;
        }
        i = 0;
        while((i < key) && (flag == 0))
        {
                if(a[i] == -1)
                {
                        a[i] = num;
                        flag=1; break;
                }
                i++;
        } // end while
    } // end else
} // end linear_prob()

void main()
{
    int num, key, i;
    int ans = 1;
    clrscr();
    printf("\nCollision handling by linear probing: ");
    for (i=0; i < MAX; i++)
    {
        a[i] = -1;
    }
    do
    {
        printf("\nEnter the data: ");
        scanf("%d", &num);
        key=create(num);
        linear_prob(key,num);
        printf("\nDo you wish to continue? (1/0): ");
        scanf("%d",&ans);
    }while(ans);
    display(emp);
}
```