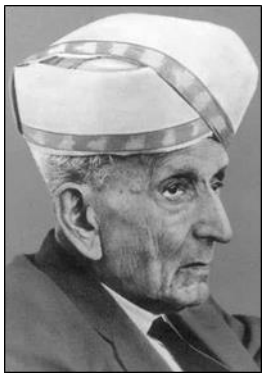
DATA STRUCTURES LABORATORY MANUAL

CHOICE BASED CREDIT SYSTEM

(BCSL305- III Semester B.E)
(Only for Computer Science Stream)
(2022 Scheme)

-: OUR MISSION :-

Disciplined and Integrated Development of Personality
Through Academic Excellence, Sports and Cultural Activities



DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING
SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY
(Affiliated to VTU, Recognized by AICTE and
Accredited by NBA, NAAC)
Bengaluru - 562157

Sir M. VISVESVARAYA INSTITUTE OF TECHNOLOGY
BENGALURU - 562 157

Institute Vision and Mission

VISION

- To be a center of excellence in technical and management education concurrently focusing on disciplined and integrated development of personality through quality education, sports, cultural and co-curricular activities.
- To promote transformation of students into better human beings, responsible citizens and competent professionals to serve as a valuable resource for industry, work environment and society.

MISSION

1. To impart quality technical education, provide state-of-art facilities, achieve high quality in teaching-learning & research and encourage extra & co-curricular activities.

2. To stimulate in students a spirit of inquiry and desire to gain knowledge and skills to meet the changing needs that can enrich their lives.

3. To provide opportunity and resources for developing skills for employability and entrepreneurship, nurturing leadership qualities, imbibing professional ethics and societal commitment.

4. To create an ambiance and nurture conducive environment for dedicated and quality staff to upgrade their knowledge & skills and disseminate the same to students on a sustainable long term basis.

5. To facilitate effective interaction with the industries, alumni and research institutions.

DEPT. OF INFORMATION SCIENCE AND ENGINEERING	
Vision	**Mission**
• To empower students with knowledge and skills to develop the competency in the emerging areas of Information Technology.	• To train the students to have Professional career in IT industry and Higher studies through Quality Education. • To provide outstanding Teaching and Research environment by implementing innovative Teaching and Research Methodologies for Quality Education and Research.

Data Structures and Applications

Course Code	**BCSL305**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2	SEE Marks	50
Total Hours of Pedagogy	--	Total Marks	100
Credits	01	Exam Hours	03

Course Objectives:

This laboratory course enables students to get practical experience in design, develop, implement, analyze and evaluation/testing of

- Dynamic memory management
- Linear data structures and their applications such as stacks, queues and lists
- Non-Linear data structures and their applications such as trees and graphs

Sl. No.	**Practice Programs**
	• Implement all the programs in "C " Programming Language and Linux OS
	PART A – List of problems for which student should develop program and execute in the Laboratory
1	Develop a Program in C for the following: a) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String). b) Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen..
2	Develop a Program in C for the following operations on Strings. a) Read a main String (STR), a Pattern String (PAT) and a Replace String (REP) b) Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR Support the program with functions for each of the above operations. Don't use Built-in functions..
3	Develop 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 how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations
4	Develop 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.
5	Develop a Program in C for the following Stack Applications a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^ a. Solving Tower of Hanoi problem with n disks.
6	Develop 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	Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Programme, 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 / Deletion at End of SLL d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack) e. Exit
8	Develop 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
9	Develop a Program in C for the following operationson Singly Circular Linked List (SCLL) with header nodes a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3 b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z) Support the program with appropriate functions for each of the above operations
10	Develop 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 a. d. Exit
11	Develop a Program in C for the following operations on Graph(G) of Cities a. Create a Graph of N cities using Adjacency Matrix. b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method
12	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. 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.

Course Outcome (Course Skill Set)

At the end of the course the student will be able to:

- Analyze various linear and non-linear data structures
- Demonstrate the working nature of different types of data structures and their applications
- Use appropriate searching and sorting algorithms for the give scenario.
- Apply the appropriate data structure for solving real world problems

Continuous Internal Evaluation(CIE): The CIE marks for the theory component of the IC shall be 30 marks and for the laboratory component 20 Marks.

CIE for the theory component of the IC

Experiment distribution

- For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
- For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution (Need to change in accordance with university regulations)
- c) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 =
- 100 Marks
- d) For laboratories having PART A and PART B
- i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
- ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

Course Outcomes - CO

CO1:Analyze various linear and non-linear data structures

CO2:Demonstrate the working nature of different types of data structures and their applications

CO3:Use appropriate searching and sorting algorithms for the give scenario.

CO4:Apply the appropriate data structure for solving real world problems

CO-PO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1												
CO2												
CO3												
CO4												

PROGRAM OUTCOMES

PO's	PO Description
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complexengineering problems.
PO2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. • That cannot be solved by straightforward application of knowledge, theories and techniques applicable to the engineering discipline as against problems given at the end of chapters in a typical text book that can be solved using simple engineering theories and techniques • That may not have a unique solution. For example, a design problem can be solved in many ways and lead to multiple possible solutions; • That require consideration of appropriate constraints / requirements not explicitly given in the problem statement such as cost, power requirement, durability, product life, etc.; which need to be defined (modelled) within appropriate mathematical framework; and • That often require use of modern computational concepts and tools
PO5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a memberor leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding ofthe engineering and management principles and apply these to one's own work, as a member and leader in a team to manage projects and in multidisciplinaryenvironments.
PO12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM -1

Develop a Program in C for the following:

a) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).

Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen..

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct fields
{
        char *Day_Name;
        int Day_Date;
        char *Discription;
};
typedef struct fields Fields;
Fields **Activity;
Fields *Create(char *Name,int Day,char *Disc)
{
        Fields *data;
        int nlen,nDisc;
        nlen=strlen(Name);
        nDisc=strlen(Disc);
        data = (Fields *)malloc(sizeof(struct fields));
        data->Day_Name=(char *)calloc(nlen,sizeof(char));
        data->Discription =(char *)calloc(nDisc,sizeof(char));
        data->Day_Date=Day;
        strcpy(data->Day_Name,Name);
        strcpy(data->Discription,Disc);
        return data;
}
```

```
void readData()
{
        char Name[10],Disc[25];
        int Day,i;
        Activity = (Fields **)calloc(7,sizeof(Fields *));
        printf("\n Enter the calendar details \n");
        for(i=0;i<7;i++)  {
                printf("Enter the Week Name\t");
                gets(Name);
                printf("Enter the Week Discription\t");
                gets(Disc);
                printf("Enter the Week Day\t");
                scanf("%d",&Day);
                Activity[i]=Create(Name,Day,Disc);
                fflush(stdin);
        }
}
void display() {
        int i;
        printf("\n Week Name\t\t Day  \t Discription\n");
        for(i=0;i<7;i++)
        {
                printf("%10s\t%d\t\t%s\n",
                        Activity[i]->Day_Name,
                        Activity[i]->Day_Date,
                        Activity[i]->Discription);
        }
}
void main() {
        readData();
        display();
}
```

PROGRAM - 2

Develop a Program in C for the following operations on Strings.

a) Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)
b) Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR

Support the program with functions for each of the above operations. Don't use Built-in functions..

```
#include<stdio.h>
#include<string.h>
int nfind(char *str,char *pat)  {
        int i,j,start=0;
        int lasts=strlen(str)-1;
        int lastp=strlen(pat)-1;
        int endmatch=lastp;
        for(i=0;endmatch<=lasts;endmatch++,start++)    {
                if(str[endmatch]==pat[lastp])    {
                        j=0;
                        for(i=start;j<lastp && str[i]==pat[j];i++)
                                j++;
                        if(j==lastp)
                                return start;
                }
        }
        return -1;
}
void StrReplace(char *string,char *pat,char *Rep) {
        int i,j;
        char Res[50];
        int pos=0;
        while(pos!=-1)  {
                pos=nfind(string,pat);
                Res[0]='\0';
                if(pos==-1)
                        printf("\n %s is not found in %s",pat,string);
                else
```

```
            {
                printf("\n %s is found at pos %d in  %s",
                        pat, pos, string);
                for(i=0;i<pos;i++)
                    Res[i]=string[i];
                for(j=0;j<strlen(Rep);j++)
                    Res[i++]=Rep[j];
                for(j=pos+strlen(pat);string[j]!='\0';j++)
                    Res[i++]=string[j];
                Res[i]='\0';
                printf("\nAfter Replace %s\n",Res);
                for(i=0;i<strlen(Res);i++)
                    string[i]=Res[i];
                string[i]='\0';
            }
        }
}
void main()
{
        char string[]={"AADAACAADAABAADAA"};
        char pat[]={"ADAA"};
        char Rep[]={"XXX"};
        StrReplace(string,pat,Rep);
}
```

PROGRAM - 3

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

1. Push an Element on to Stack
2. Pop an Element from Stack
3. Demonstrate how Stack can be used to check Palindrome
4. Demonstrate Overflow and Underflow situations on Stack
5. Display the status of Stack
6. Exit

Support the program with appropriate functions for each of the above operations **charges.**

```
#include<stdio.h>
#include<stdlib.h>
#define STACK_SIZE 10
typedef struct
{
        int key;
}element;
element Stack[STACK_SIZE];
int top = -1;
int IsEmpty() {
        return ((top == -1)? 1 : 0);
}
int IsFull() {
        return ((top >= STACK_SIZE - 1)? 1 : 0);
}
void push(int ele) {
        if(IsFull())
            printf("\nStack Full");
        else
            Stack[++top].key=ele;
}
int pop() {
        if(IsEmpty()) {
            printf("\nStack Empty");
            return -1;
```

```
        }
        else
            return(Stack[top--].key);
}
void display()
{
        int i;
        if(IsEmpty())
            printf("\n Stack is Empty\n");
        else
        {
            printf("\n Elements of Stack\t");
            for (i=top;i>=0;i--)
               printf("\t%d",Stack[i].key);
        }
}
void main()
{
        int n=10,i,oldTop,num,OrgNum,ele;
        int ch,flag=1;
        int flag1;
        while(flag)
        {
            printf("\n Menu Driven Program\n");
            printf("\n1 Push\n2 Pop\n3 Palindrom\n4 Display\n5
Exit\n");
            printf("\nEnter the Choice\t");
            scanf("%d",&ch);
            switch(ch)
            {
               case 1:
                           printf("\nEnter the Element to push\t");
                           scanf("%d",&ele);
```

```
                push(ele);
                break;
        case 2:
                ele=pop();
                if(ele!=-1)
                        printf("\n The Element Poped is %d",ele);
                break;
        case 3:
                oldTop=top;
                top=-1;
                printf("\n Enter a number\t");
                scanf("%5d",&num);
                OrgNum=num;
                while(num!=0)
                {
                        push(num%10);
                        num=num/10;
                }
                flag1=1;
                num=OrgNum;
                while(!IsEmpty())
                {
                        if(num%10 != pop())
                                flag1=0;
                        num=num/10;
                }
                if(flag1==0)
                        printf("\n %d is not palindrom",OrgNum);
                else
                        printf("\n %d is palindrom",OrgNum);
                top=-1;
                break;
        case 4:
```

```
                    display();
                    break;
            case 5:
            default : flag=0;
          }
       }
}
```

PROGRAM - 4

Develop 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>
#include<string.h>
#define MAX 100
typedef enum
{lparn,rparn,plus,minus,times,divide,mod,eos,operand} predence;
char stack[MAX];
char exprn[MAX],postfix[MAX];
static int isp [ ] = {0, 19, 12, 12, 13, 13, 13, 0};
static int icp [ ] = {20, 19, 12, 12, 13, 13, 13, 0};
predence getToken(char *symbol,int *n)
{
        *symbol = exprn[(*n)++];
        switch(*symbol)
        {
           case '(': return lparn;
           case ')': return rparn;
           case '+': return plus;
           case '-': return minus;
           case '*': return times;
           case '/': return divide;
           case '%': return mod;
           case '#': return eos;
           default: return operand;
        }
}
```

```
predence getStackToken(char symbol)
{
        switch(symbol)
        {
           case '(': return lparn;
           case ')': return rparn;
           case '+': return plus;
           case '-': return minus;
           case '*': return times;
           case '/': return divide;
           case '%': return mod;
           case '#': return eos;
           default: return operand;
        }
}
void push(char ele,int *top)
{
        stack[++(*top)]  = ele;
}

char pop(int *top)
{
        char ele;
        ele=stack[(*top)--];
        return(ele);
}
void convert(char *exprn,char *postfix)
{
        int n=0,i=0;
        predence token;
        char symbol;
        int opr1,opr2;
        int top = -1;
```

```
        token=getToken(&symbol,&n);
        while(token!=eos)
        {
           switch(token)                {
              case operand:
                              postfix[i++]=symbol;
                              break;
              case lparn:
                              push(symbol,&top);
                              break;
              case rparn:
                              while(stack[top]!='(')
                                    postfix[i++]=pop(&top);
                              pop(&top);
                              break;
              case plus:
              case minus:
              case times:
              case divide:
              case mod:
                             if(top==-1)
                                    push(symbol,&top);
                             else
                             {
                              while(isp[getStackToken(stack[top])]  >=
icp[token] && top!=-1)
                                          postfix[i++]=pop(&top);
                                    push(symbol,&top);
                             }
                             break;
           }
           token=getToken(&symbol,&n);
        }
```

```
        while(top!=-1)
           postfix[i++]=pop(&top);
        postfix[i]='\0';
}
void main()
{
        printf("\n Enter the postfix expn\t");
        scanf("%s",exprn);
        strcat(exprn,"#");
        convert(exprn,postfix);
        printf("\n After Evaluation %s",postfix);
}
```

PROGRAM - 5

Develop a Program in C for the following Stack Applications

a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^
b. Solving Tower of Hanoi problem with n disks.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 100
typedef enum
{lparn=1,rparn,plus,minus,times,divide,mod,eos,operand} predence;
int stack[MAX];
char exprn[MAX],postfix[MAX];
static int isp [ ] = {0, 19, 12, 12, 13, 13, 13, 0};
static int icp [ ] = {20, 19, 12, 12, 13, 13, 13, 0};
predence getToken(char *symbol,int *n)
{
        *symbol = exprn[(*n)++];
        switch(*symbol)
        {
                case '(': return lparn;
                case ')': return rparn;
                case '+': return plus;
                case '-': return minus;
                case '*': return times;
                case '/': return divide;
                case '%': return mod;
                case '#': return eos;
                default: return operand;
        }
}
predence getStackToken(char symbol)
{
        switch(symbol)
```

```
        {
                case '(': return lparn;
                case ')': return rparn;
                case '+': return plus;
                case '-': return minus;
                case '*': return times;
                case '/': return divide;
                case '%': return mod;
                case '#': return eos;
                default: return operand;
        }
}
void push(int ele,int *top)
{
        stack[++(*top)]     = ele;
}
int pop(int *top)
{
        int ele;
        ele=stack[(*top)--];
        return(ele);
}
void convert(char *exprn,char *postfix)
{
        int n=0,i=0;
        predence token;
        char symbol;
        int opr1,opr2;
        int top = -1;
        token=getToken(&symbol,&n);
        while(token!=eos)
        {
                switch(token)
```

```
        {
                case operand:
                                postfix[i++]=symbol;
                                break;
                case lparn:
                                push(symbol,&top);
                                break;
                case rparn:
                                while(stack[top]!='(')
                                        postfix[i++]=pop(&top);
                                pop(&top);
                                break;
                case plus:
                case minus:
                case times:
                case divide:
                case mod:
                                if(top==-1)
                                        push(symbol,&top);
                                else
                                {
                                while(
                                isp[getStackToken(stack[top])]
                                        >=
                                 icp[token] && top!=-1)
                                                postfix[i++]=pop(&top);
                                        push(symbol,&top);
                                }
                                break;
        }

        token=getToken(&symbol,&n);
    }
```

```
        while(top!=-1)
                postfix[i++]=pop(&top);
        postfix[i]='\0';
}
int eval()
{
        int n=0,i,j;
        predence token;
        char symbol;
        int opr1,opr2;
        int top = -1;
        token=getToken(&symbol,&n);
        while(token!=eos)
        {
                if(token==operand)
                        push(symbol-'0',&top);
                else
                {
                        opr2=pop(&top);
                        opr1=pop(&top);
                        switch(token)
                        {
                                case plus       : push(opr1+opr2,&top); break;
                                case minus      : push(opr1-opr2,&top); break;
                                case times      : push(opr1*opr2,&top); break;
                                case mod        : push(opr1%opr2,&top); break;
                                case divide     : if(opr2!=0)
                                                        push(opr1/opr2,&top);
                                                  else
                                                  {
                                                  printf("\nDivide by Zero Error");
                                                        exit(0);
                                                  }
```

```
                                        break;
                }
            }
            for(j=top;j>=0;j--)
                    printf("\nStack[%d] = %d",j,stack[j]);
            token=getToken(&symbol,&n);
        }
        return pop(&top);
}
void main()
{
      char Texprn[MAX];
      printf("\n Enter the Sufix expn\t");
      scanf("%s",exprn);
      strcat(exprn,"#");
      strcpy(Texprn,exprn);
      convert(exprn,postfix);
      strcpy(exprn,postfix);
      strcat(exprn,"#");
      printf("\n Prefix exprn %s",exprn);
      printf("\n After Evaluation of Expression %s is %d",Texprn,eval());
}
```

c. Tower of Haonoi

```
#include<stdio.h>
void towerOfHanoi(int n, char from_rod,char to_rod, char aux_rod)
{
   if (n == 1)
   {
      printf("Move disk %d from rod %c to rod %c\n" ,
            n,
            from_rod ,
            to_rod);
      return;
   }
   // Push all values of n, from_rod, aux_rod, to_rod stack
   towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
   // Pop all values of n, from_rod, aux_rod, to_rod stack
   printf("Move disk %d from rod %c to rod %c\n" , n,from_rod
,to_rod);
    // Push all values of n, from_rod, aux_rod, to_rod stack
   towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
    // Pop all values of n, from_rod, aux_rod, to_rod stack
}

// Driver code
int main()
{
   int n = 3;
   towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
   return 0;
}
```

PROGRAM - 6

Develop 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.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_SIZE 5
typedef struct
{
        int key;
}element;
element Queue[MAX_SIZE];
int front = -1;
int rear = -1;
int count = 0;
int Qfull()
{
        return count ==  MAX_SIZE;
}
int Qempty()
{
        return count ==  0;
}
void insert(int ele) {
        rear=(rear+1) % MAX_SIZE;
        if(Qfull()) {
                printf("\nQueue Full");
        }
        if(front==-1)
                front=0;
```

```
        count++;
        Queue[rear].key=ele;
}
int deleteq() {
        int ret;
        if(Qempty()) {
                printf("\nQueue Empty");
        }
        ret = Queue[front].key;
        front=(front+1) % MAX_SIZE;
        count--;
        return(ret);
}
void display()
{
        int i,j;
        if(Qempty())
                printf("\n Queue is empty\n");
        else
        {
                j=front;
                printf("\n Elements in the Queue\n");
                for(i=0;i<count;i++)
                {
                        printf(" %d(%d)",Queue[j].key,j);
                        j=(j+1) % MAX_SIZE;
                }
        }
}
void main()
{
        int n=4,i,ch,ele,flag=1;
        while(flag)
```

```
    {
        printf("\n Menu driven Program in C for the Circular
QUEUE operations \n");
        printf("\n1 Insert\n2 Delete\n3 Display\n4 Exit\n Enter the
Choice");
        scanf("%d",&ch);
        switch(ch) {
            case 1:
                if(Qfull()) {
                    printf("\nQueue Full");
                }
                else
                {
                    printf("\n Enter the value for
insert\t");
                    scanf("%d",&ele);
                    insert(ele);
                }
                break;
            case 2: if (!Qempty()) {
                    ele = deleteq();
                    printf("\n%d",ele);
                }
                else
                        printf("\nQueue Empty");
                break;
            case 3: display();
                break;
            default: flag = 0;
        }
    }
}
```

PROGRAM - 7

Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Programme, 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 / Deletion at End of SLL
d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
e. Exit

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct student {
        char USN[10];
        char Name[20];
        char Branch[15];
        char Sem[3];
        char PhNo[10];
};
typedef struct student STU;
struct node {
        STU info;
        struct node *link;
};
typedef struct node* Node;
STU GetData() {
         STU temp;
         printf("Enter the usn: ");
         fflush(stdin);
         gets(temp.USN);
         printf("\nEnter the Name:");
         fflush(stdin);
         gets(temp.Name);
         printf("\nEnter branch:");
         fflush(stdin);
         gets(temp.Branch);
```

```
        printf("\nEnter sem:");
        fflush(stdin);
        gets(temp.Sem);
        printf("\nEnter the phone number:");
        fflush(stdin);
        gets(temp.PhNo);
        return temp;
}
void displayRec(STU ele)  {
        printf("\n%s\t\t",ele.USN);
        printf("%s\t\t",ele.Name);
        printf("%s\t",ele.Branch);
        printf("%s\t",ele.Sem);
        printf("%s\n",ele.PhNo);
}
Node Create(STU ele)  {
        Node temp;
        temp=(Node)malloc(sizeof(struct node));
        temp->info=ele;
        temp->link=NULL;
        return temp;
}
Node Finsert(Node first,STU ele)  {
        Node temp=Create(ele);
        if(first== NULL)
                first=temp;
        else
        {
                temp->link=first;
                first = temp;
        }
        return first;
}
```

```
Node Einsert(Node first,STU ele)
{
	Node temp=Create(ele),cur;
	if(first== NULL)
		first=temp;
	else
	{
		//for(cur=first;cur->link!=NULL;cur=cur->link);
		cur=first;
		while(cur->link!=NULL)
			cur=cur->link;
		cur->link=temp;
	}
	return first;
}
void display(Node first)
{
	Node temp=first;
	if(temp == NULL)
		printf("\nList is empty\n");
	else
	{
		printf("\nStudent Details in the list\n");
		printf("USN\t\tName\t\tBranch\tSem\tPhno\n");
		while(temp!=NULL)
		{
			displayRec(temp->info);
			temp=temp->link;
		}
	}
}
```

```
int Count(Node first)
{
	Node temp=first;
	int cnt=0;
	if(temp == NULL)
		printf("\nList is empty\n");
	else
	{
		while(temp!=NULL)
		{
			cnt++;
			temp=temp->link;
		}
	}
	return cnt;
}
Node Fdelete(Node first)
{
	Node temp=first;
	if(first== NULL)
		printf("\nList is empty");
	else
	{
		temp=first;
		printf("\nElement Deleted is\n");
		printf("USN\t\tName\t\tBranch\tSem\tPhno\n");
		displayRec(temp->info);
		first = first->link;
		free(temp);
	}
	return first;
}
```

```
Node Edelete(Node first)
{
	Node temp=first,t;
	if(first== NULL)
		printf("\nList is empty");
	else if(first->link==NULL)
	{
		printf("\nElement Deleted is\n");
		printf("USN\t\tName\t\tBranch\tSem\tPhno\n");
		displayRec(first->info);
		first=NULL;
	}
	else
	{
		//for(temp=first;temp->link->link!=NULL;temp=temp-
>link);
		while(temp->link->link!=NULL)
			temp=temp->link;
		printf("\nElement Deleted is\n");
		printf("USN\t\tName\t\tBranch\tSem\tPhno\n");
		displayRec(temp->link->info);
		t = temp->link;
		temp->link=NULL;
		free(t);
	}
	return first;
}
```

```
void main() {
        Node first=NULL;
        int flag=1,ch;
        STU ele;
        while(flag)        {
                printf("\n Menu \n");
                printf("\n1 Front Insert\n2 End Insert\n3 Front Delete\n4
End Delete\n5 Display\n6 No. of Nodes in the list\n");
                printf("7 Exit\n Enter the Choice");
                scanf("%d",&ch);
                switch(ch)                {
                        case 1: printf("\n Enter the Student Details \t");
                                        ele=GetData();
                                        first=Finsert(first,ele);
                                        break;
                        case 2: printf("\n Enter the Student Details \t");
                                        ele=GetData();
                                        first=Einsert(first,ele);
                                        break;
                        case 3: first=Fdelete(first);
                                        break;
                        case 4: first=Edelete(first);
                                        break;
                        case 5: display(first);
                                        break;
                        case 6: printf("\n the Number of Node in list
%d\t",Count(first));
                                        break;
                        default: flag = 0;
                }
        }
}
```

PROGRAM 8

Develop 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

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct Employee
{
        char SSN[10];
        char Name[10];
        char Branch[10];
        char Des[10];
        char sal[10];
        char phone[10];
};
typedef struct Employee EMP;
struct node
{
        EMP info;
        struct node *lptr,*rptr;
};
typedef struct node* Node;
Node front=NULL,rear=NULL;
EMP GetRec()
{
        EMP temp;
        printf("Enter the SSN: ");
        fflush(stdin);
        gets(temp.SSN);
```

```
        printf("Enter the Name:");
        fflush(stdin);
        gets(temp.Name);
        printf("Enter branch:");
        fflush(stdin);
        gets(temp.Branch);
        printf("Enter the designation: ");
        fflush(stdin);
        gets(temp.Des);
        printf("Enter sal:");
        fflush(stdin);
        gets(temp.sal);
        printf("Enter the phone number:");
        fflush(stdin);
        gets(temp.phone);
        return temp;
}
void DispRec(EMP temp)
{
        printf("%s\t",temp.SSN);
        printf("%s\t",temp.Name);
        printf("%s\t",temp.Branch);
        printf("%s\t",temp.Des);
        printf("%s\t",temp.sal);
        printf("%s\n",temp.phone);
}
Node Create(EMP ele)
{
        Node temp;
        temp=(Node)malloc(sizeof(struct node));
        temp->info=ele;
        temp->lptr=NULL;
        temp->rptr=NULL;
```

```
        return temp;
}
void Finsert(EMP ele)
{
        Node temp=Create(ele);
        if(front== NULL)
        {
                front=temp;
                rear=temp;
        }
        else
        {
                temp->rptr=front;
                front->lptr=temp;
                front = temp;
        }
}
void Einsert(EMP ele)
{
        Node temp=Create(ele);
        if(rear == NULL)
        {
                front=temp;
                rear=temp;
        }
        else
        {
                rear->rptr=temp;
                temp->lptr=rear;
                rear=temp;
        }
}
void Fdelete()
```

```
{
        Node temp=front;
        if(front == NULL)
                printf("\nList is empty");
        else
        {
                temp=front;
                printf("\nDeleted Employee Records \n");
                DispRec(temp->info);
                front = front->rptr;
                front->lptr= NULL;
                if(front == NULL)
                        rear = NULL;
                free(temp);
        }
}
void Edelete()
{
        Node temp=rear,t;
        if(rear== NULL)
                printf("\nList is empty");
        else if(rear->lptr == NULL)
        {
                printf("\nDeleted Employee Records \n");
                DispRec(rear->info);
                front=rear=NULL;
                free(temp);
        }
        else
        {
                rear = rear->lptr;
                rear->rptr=NULL;
                printf("\nDeleted Employee Records \n");
```

```
            DispRec(temp->info);
            free(temp);
        }
}
void display(Node front)
{
        Node temp=front;
        if(temp == NULL)
            printf("\nList is empty\n");
        else
        {
            printf("\nEmployee Records list\n");
            while(temp!=NULL)
            {
                DispRec(temp->info);
                temp=temp->rptr;
            }
        }
}
void main()
{
        EMP ele;
        int flag=1,ch;
        while(flag)
        {
            printf("\n Menu Implementation of Double Ended Queue
using DLL\n");
            printf("\n1 Insert Front \n2 Insert rear \n3 Delete Front \n4
Delete Rear \n5 Display\n");
            printf("6 Exit\n Enter the Choice");
            scanf("%d",&ch);
            switch(ch)
            {
```

```
                case 1: printf("\n Enter the Employee Record to Insert
Front of Queue\n");
                        ele = GetRec();
                        Finsert(ele);
                        break;
                case 2: printf("\n Enter the Employee Record to Insert
Front of Queue\n");
                        ele=GetRec();
                        Einsert(ele);
                        break;
                case 3: Fdelete();
                        break;
                case 4: Edelete();
                        break;
                case 5: display(front);
                        break;
                default: flag = 0;
            }
        }
}
```

PROGRAM - 9

Develop a Program in C for the following operationson Singly Circular Linked List (SCLL) with header nodes

a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3

b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)

Support the program with appropriate functions for each of the above operations

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct term
{
    int coeff;
    int pow_x;
    int pow_y;
    int pow_z;
};
typedef struct term TERM;
struct node
{
    TERM info;
    struct node* next;
};
typedef struct node* Node;
Node Create(TERM t)
{
    Node temp = (Node)malloc(sizeof(struct node));
    temp->info=t;
    temp->next = temp;
    return temp;
}
Node MKHeader()
{
```

```
        TERM x;
        Node temp;
        x.coeff=-1;
        x.pow_x=-1;
        x.pow_y=-1;
        x.pow_z=-1;
        temp=Create(x);
        return temp;
}
Node Insert(Node p,TERM t)
{
        Node temp = Create(t),cur=p;
        while(cur->next!=p)
                cur=cur->next;
        cur->next=temp;
        temp->next=p;
        return p;
}
double Compute(Node temp,int x,int y,int z)
{
        double ret;
        TERM t=temp->info;
        ret=t.coeff * pow(x, t.pow_x) * pow(y, t.pow_y) *
pow(z,t.pow_z);
        return ret;
}
double Evaluate(Node p, int x, int y, int z)
{
        Node po = p->next;
        double sum = 0;
        while (po!=p)
        {
                sum += Compute(po,x,y,z);
                po = po->next;
        }
```

```
	return sum;
}
void DispTerm(TERM t)
{
	if(t.coeff!=0)
	{
		if(t.coeff>0)
			printf("+");
		printf("%dx^%dy^%dz^%d", t.coeff, t.pow_x, t.pow_y,
t.pow_z);
	}
}
void PrintPoly(Node p)
{
	Node po = p->next;
	while (po!=p)
	{
		DispTerm(po->info);
		po = po->next;
	}
	printf("\n");
}
void ReadPoly(Node t1,int n)
{
	TERM t;
	int i;
	for(i=1;i<=n;i++)
	{
		printf("Enter the value of coefficent and powers of x,y
and z");

	scanf("%d%d%d%d",&t.coeff,&t.pow_x,&t.pow_y,&t.pow_z
);
		t1 = Insert(t1, t);
	}
```

```
        PrintPoly(t1);
}
int ComparePower(TERM m,TERM n)
{
        if(m.pow_x==n.pow_x && m.pow_y==n.pow_y &&
m.pow_z==n.pow_z)
                return 1;
        else
                return 0;
}
TERM AddTerms(TERM m,TERM n)
{
        TERM temp;
        temp.coeff=m.coeff+n.coeff;
        temp.pow_x = m.pow_x;
        temp.pow_y = m.pow_y;
        temp.pow_z = m.pow_z;
        return temp;
}
Node AddPoly(Node p1,Node p2)
{
        Node Newlist=MKHeader();
        Node t1=p1->next,t3;
        Node t2=p2->next,t4;
        TERM res;
        int i,flag;
        t3=t1;
        t4=t2;
        while(t1!=p1)
        {
                t2=p2->next;
                flag=1;
                while(t2!=p2 && flag)
                {
                        if(ComparePower(t1->info,t2->info))
```

```
                {
                        res=AddTerms(t1->info,t2->info);
                        Newlist=Insert(Newlist,res);
                        flag=0;
                }
                t2=t2->next;
        }
        if (flag==1)
                Newlist=Insert(Newlist,t1->info);
        t1=t1->next;
    }
    while(t4!=p2)
    {
        t3=Newlist->next;
        flag=1;
        while(t3!=Newlist && flag)
        {
                if(ComparePower(t3->info,t4->info))
                        flag=0;
                t3=t3->next;
        }
        if (flag==1)
                Newlist=Insert(Newlist,t4->info);
        t4=t4->next;
    }
    return Newlist;
}
int main()
{
    int n,x,y,z,ch,i,coeff;
    Node polysum;
    Node poly1 = MKHeader();
    Node poly2 = MKHeader();
    while(1)
    {
```

```
            printf("\nMenu\n 1:Evaluate Polynomial \n 2:Add\n
3:Exit\n Enter your choice\n");
            scanf("%d",&ch);
            switch(ch)
            {
                  case 1: printf("Enter the terms in the polynomial");
                              scanf("%d",&n);
                              ReadPoly(poly1,n);
                              printf("Enter the values of x,y and z");
                              scanf("%d%d%d",&x,&y,&z);
                              printf("%.2f\n", Evaluate(poly1, x, y, z));
                              break;
                  case 2:     printf("Enter the terms in the polynomial
2");
                              scanf("%d",&n);
                              ReadPoly(poly2,n);
                              polysum = AddPoly(poly1, poly2);
                              PrintPoly(polysum);
                              break;
                  case 3: exit(0);
            }
      }
      return 0;
}
```

PROGRAM - 10

Develop 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

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        struct node *left;
        int data;
        struct node *right;
};
typedef struct node* Node;
Node newNode(int item)
{
        Node temp = (Node)malloc(sizeof(struct node));
        temp->data = item;
        temp->left = temp->right = NULL;
        return temp;
}
Node insert(Node root,int info)
{
        Node temp=newNode(info),T1,curr;
        if(root==NULL)
                root=temp;
        else
        {
                T1=root;
                while(T1!=NULL)
                {
                        curr=T1;
```

```
                if(info == T1->data)
                {
                        printf("\nDuplicate Number\n");
                        return root;
                }
                //T1= (info<T1->data) ? T1->left:T1->right;
                if(info<T1->data)
                        T1=T1->left;
                else
                        T1=T1->right;
        }
        if(info<curr->data)
                curr->left=temp;
        else
                curr->right=temp;
    }
    return root;
}
int search(Node root, int key)
{
    if (root == NULL)
        return -1;
    if(root->data == key)
        return 1;
    if (root->data < key)
        return search(root->right, key);
    return search(root->left, key);
}
 void inorder(Node root)
{
    if (root != NULL)
    {
        inorder(root->left);
```

```
                printf("%d \t", root->data);
                inorder(root->right);
        }
}
void preorder(Node root)
{
        if (root != NULL)
        {
                printf("%d \t", root->data);
                preorder(root->left);
                preorder(root->right);
        }
}
void postorder(Node root)
{
        if (root != NULL)
        {
                postorder(root->left);
                postorder(root->right);
                printf("%d \t", root->data);
        }
}
int main()
{
        int n,i,ch,ch1,key,pos;
        Node root=NULL;
        printf("Enterthe no of nodes in the BST\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("Enter the element to be inserted\n");
                scanf("%d",&key);
                root=insert(root,key);
```

```
        }
        while(1)
        {
                printf("\nEnter the choice\n1: Insert Node\n2:
Traversal\n3: Search for key\n4: Exit\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                printf("Enter the element to be
inserted\n");
                                scanf("%d",&key);
                                root=insert(root,key);
                                break;
                        case 2:
                                printf("Enter your choice\n1: Preorder\n2:
Inorder\n3: Postorder\n");
                                scanf("%d",&ch1);
                                switch(ch1)
                                {
                                case 1:
                                        preorder(root);
                                        break;
                                case 2:
                                        inorder(root);
                                        break;
                                case 3:
                                        postorder(root);
                                        break;
                                default:
                                        printf("\n Make Correct Choice");
                                }
                                break;
```

```
                case 3:
                        printf("Enter the key to be searched\n");
                        scanf("%d",&key);
                        pos=search(root,key);
                        if (pos==-1)
                                printf("\n Key is not found\n");
                        else
                                printf("\n Key is found\n");
                        break;
                case 4: exit(0);
            }
        }
        return 0;
}
```

PROGRAM - 11

Develop a Program in C for the following operations on Graph(G) of Cities a. Create a Graph of N cities using Adjacency Matrix. b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct Term
{
        int vertex,cost;
};
struct node
{
   struct node *next;
   int vertex;
};
typedef struct node* Node;
Node front,rear,Bfront,Brear;
Node G[20];
int visited[20];
Node CreateNode(int info)
{
        Node temp =(Node)malloc(sizeof(struct node));
        temp->vertex=info;
        temp->next=NULL;
        return temp;
}
void DFS(int i)
{
   Node p;
   printf("%5d",i);
   p=G[i];
```

```
   visited[i]=1;
   while(p!=NULL)
   {
     i=p->vertex;
     if(!visited[i])
         DFS(i);
      p=p->next;
   }
}
void insert(int vi,int vj)
{
   Node p,q;
   q=CreateNode(vj);
   if(G[vi]==NULL)
      G[vi]=q;
   else
   {
      p=G[vi];
      while(p->next!=NULL)
         p=p->next;
      p->next=q;
   }
}
void read_graph(int n)
{
   int i,st,ed,no_of_edges,j;
   Node first;
   for(i=0;i<n;i++)
   {
       first = NULL;
       G[i]=first;
      printf("Enter number of edges from node %d:",i);
       scanf("%d",&no_of_edges);
```

```
        for(j=0;j<no_of_edges;j++)
      {
         printf("Enter an edge(%d,v):",i);
         scanf("%d",&ed);
         insert(i,ed);
      }
   }
}
void addQ(int i)
{
        Node temp=CreateNode(i);
        if(front==NULL && rear==NULL)
                front=rear=temp;
        else
        {
                rear->next=temp;
                rear=temp;
        }
}
int delQ()
{
        int i;
        i=front->vertex;
        front=front->next;
        if(front==NULL)
                rear=NULL;
        return i;
}
void BFS(int v)
{
        Node w;
        printf("%5d",v);
        visited[v]=1;
```

```
        addQ(v);
        while(front)
        {
                v=delQ();
                for(w=G[v];w;w=w->next)
                {
                        if(!visited[w->vertex])
                        {
                                printf("%5d",w->vertex);
                                addQ(w->vertex);
                                visited[w->vertex]=1;
                        }
                }
        }

}


void main()
{
        int i,flag=1,ch,n;
        front=NULL;
        rear=NULL;
        Bfront=NULL;
        Brear=NULL;
        printf("Enter number of vertices:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
                visited[i]=0;
         read_graph(n);
        Bfront=front;
        Brear=rear;
        while(flag)
```

```
        {
                printf("\nEnter the choice\n1: DFS\n2: BFS\n3: Exit\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                        DFS(0);
                                        break;
                        case 2:
                                        BFS(0);
                                        break;
                        case 3:
                        default: flag=0;
                }
        }
}
```

PROGRAM - 12

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. 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

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

int key[20],n,m;
int *ht,indX;
int count = 0;

void insert(int key)
{
   indX = key % m;
   while(ht[indX] != -1)
   {
      indX = (indX+1)%m;
   }
   ht[indX] = key;
   count++;
 }

void display()
{
        int i;
        if(count == 0)
        {
                printf("\nHash Table is empty");
                return;
```

```
        }
        printf("\nHash Table contents are:\n ");
        for(i=0; i<m; i++)
       printf("\n T[%d] --> %d ", i, ht[i]);
}
void main()
{
        int i;
        printf("\nEnter the number of employee  records (N) :   ");
        scanf("%d", &n);
        printf("\nEnter the two digit memory locations (m) for hash
table:   ");
        scanf("%d", &m);
        ht = (int *)malloc(m*sizeof(int));
        for(i=0; i<m; i++)
                ht[i] = -1;
        printf("\nEnter the four digit key values (K) for N Employee
Records:\n  ");
        for(i=0; i<n; i++)
                scanf("%d", &key[i]);
        for(i=0;i<n;i++)  {
                if(count == m)          {
                        printf("\n~~~Hash table is full. Cannot insert the
record %d key~~~",i+1);
                        break;
                }
                insert(key[i]);
        }
        //Displaying Keys inserted into hash table
        display();
}
```

OPEN-ENDED EXPERIMENT / PROJECT

Report Format for Open-Ended Experiment based Program / Project

Title page	Key information and one illustration
Executive summary	One page summary of the project
Table of contents	
Problem definition • Problem scope • Technical Review • Design requirements	Introduces and defines the problem
Design description • Overview • Detailed description • Use	Describes the design
Evaluation • Overview • Prototype • Testing and results • Assessment • Next Steps	Evaluates the design
References	List of references used