DATA STRUCTURES LABORATORY [As per Choice Based Credit System (CBCS) scheme] (Effective from the academic year 2015 -2016) SEMESTER - III			
Laboratory Code	15CSL38	IA Marks	20
Number of Lecture Hours/Week	01I + 02P	Exam Marks	80
Total Number of Lecture Hours	40	Exam Hours	03
CREDITS - 02			

LAB – MANUAL

Department of Computer Science and Engineering
BMSIT&M
Bengaluru-98

```
/* 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 50
void create(int a[], int n);
void display(int a[], int n);
void insert(int ele, int pos, int a[], int *n);
void delete(int pos, int a[], int *n);
void main()
{
      int a[SIZE], i, choice, n, ele, pos;
      printf("Enter number of elements in the array:\n");
      scanf("%d", &n);
      create(a, n);
      for(;;)
      {
            printf("Enter\n1. Display\n2. Insert\n3. Delete\n4. Exit\n");
            scanf("%d", &choice);
            switch(choice)
            {
                  case 1: display(a, n);
                        break;
                  case 2: printf("Enter the element to be inserted:\n");
                        scanf("%d", &ele);
                        printf("Enter position for insertion:\n");
                        scanf("%d", &pos);
                        insert(ele, pos, a, &n);
                        break;
                  case 3: printf("Enter position for deletion:\n");
                        scanf("%d", &pos);
                        delete(pos, a, &n);
                        break;
                  case 4: exit(0);
            }
      }
}
void create(int a[], int n)
{
      int i;
      printf("Enter array elements:\n");
      for(i=0;i<n;i++)
            scanf("%d", &a[i]);
}
void display(int a[], int n)
```

```
{
	int i;
	printf("Array elements:\n");
	for(i=0;i<n;i++)
		printf("%d\t", a[i]);
	printf("\n");
}
void insert(int ele, int pos, int a[], int *n)
{
	int i;
	if(pos>*n+1)
	{
		printf("Invalid position.\n");
		return;
	}
	for(i= *n-1; i>=pos-1; i--)
		a[i+1]= a[i];
	a[pos-1]= ele;
	*n+=1;
	printf("Array after insertion:\n");
	for(i=0;i<*n;i++)
		printf("%d\t", a[i]);
	printf("\n");
}
void delete(int pos, int a[], int *n)
{
	int i;
	if(pos>*n)
	{
		printf("Invalid position.\n");
		return;
	}
	for(i=pos-1; i< *n-1; i++)
		a[i]= a[i+1];
	*n-=1;
	printf("Array after deletion:\n");
	for(i=0;i<*n;i++)
		printf("%d\t", a[i]);
	printf("\n");
}
```

/*2. Design, Develop and Implement 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. */

```
//THIS PROGRAM IS CASE SENSITIVE
//This program doesn't include spaces since scanf() is used for reading a string instead of gets()
#include<stdio.h>
#include<stdlib.h>
void string_replace(char str[], char pat[], char rep_pat[], char new_str[], int *mflag, int *n);
void display_result(char new_str[], int mflag, int n);
void main()
{
      char str[100],pat[20],rep_pat[20], new_str[100];
      int mflag=0, n=0;
      printf("Enter the string\n");
      scanf("%s",str);
      printf("Enter the pattern to be replaced\n");
      scanf("%s",pat);
      printf("Enter the replacing string\n");
      scanf("%s", rep_pat);
      string_replace(str, pat, rep_pat, new_str, &mflag, &n);
      display_result(new_str, mflag, n);
}
void string_replace(char str[], char pat[], char rep_pat[], char new_str[], int *mflag, int *n)
{
      int i=0, j=0, k, rep_ind, flag=0;
      while(str[i]!='\0')//1st while satrts
      {
            j=0,k=i,rep_ind=0;
            while((str[k]==pat[j])&&(pat[j]!='\0'))
            {
                  k++;
                  j++;
            }
            if(pat[j]=='\0')
            {
                  flag=1;
                  *mflag=1;
                  while(rep_pat[rep_ind]!='\0')
                  {
                        new_str[*n]=rep_pat[rep_ind];
```

```
                        rep_ind++;
                        (*n)++;
                  }
            }
            else
            {
                  flag=0;
            }
            if(flag==1)
            {
                  i=k;
            }
            else
            {
                  new_str[*n]=str[i];
                  i++;
                  (*n)++;
            }
      }//1st while ends
}
void display_result(char new_str[], int mflag, int n)
{

      if(mflag!=1)
      {
            printf("Pattern not found!!!\n");
            exit(0);
      }
      new_str[n]='\0';
      printf("The new string is:\n");
      printf("%s\n",new_str);
}
```

```
/* 3. 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 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 */


#include<stdio.h>
#include<stdlib.h>
#define SIZE 20
void push(int ele, int *top, int stack[]);
void pop(int *top, int stack[]);
void display(int top, int stack[]);
int palindrome(int *top, int stack[]);
void main()
{
      int choice, top=-1, ele, flag;
      int stack[SIZE];
      for(;;)
      {
            printf("Enter\n1. Push\n2. Pop\n3. Display\n4. Palindrome\n5.
Exit\n");
            scanf("%d", &choice);
            switch(choice)
            {
                  case 1: if(top==(SIZE-1))
                              printf("Stack overflow!!!\n");
                        else
                        {
                              printf("Enter element to be pushed:\n");
                              scanf("%d", &ele);
                              push(ele, &top, stack);
                        }
                        break;
                  case 2: if(top==-1)
                              printf("Stack underflow!!!\n");
                        else
                              pop(&top, stack);
                        break;
                  case 3:  if(top==-1)
                              printf("Stack underflow!!!\n");
                        else
                              display(top, stack);
                        break;
                  case 4: flag= palindrome(&top, stack);
                        if(flag==-1)
                              printf("Not a palindrome.\n");
                        else
```

```
                              printf("Palindrome.\n");
                        break;
                  case 5: exit(0);
            }
      }
}
void push(int ele, int *top, int stack[])
{
      *top+=1;
      stack[*top]= ele;
}
void pop(int *top, int stack[])
{
      printf("Element to be deleted:\n%d\n", stack[*top]);
      *top-=1;
}
void display(int top, int stack[])
{
      int i;
      printf("Elements are:\n");
      for(i=top; i>=0; i--)
            printf("%d\t", stack[i]);
      printf("\n");
}
int palindrome(int *top, int stack[])
{
      int temp[SIZE], i, j, count=0;
      for(j=0; *top>=0; j++)
      {
            temp[j]= stack[(*top)--];
            count+=1;
      }
      for(i=0; i<=count/2; i++)
      {
            if( temp[i]!=temp[count-i-1])
                  return -1;
      }
      return 1;
}
```

```
/* 4. 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, power, eos,
opnd} precedence;
void postfix( char infix[]);
precedence get_token(char * symbol, int *n, char infix[]);
void print_token(precedence item);
void push(int value, int stack[], int *top);
int pop(int stack[], int *top);
void main()
{
      char infix[20];
      printf("Enter infix expression:\n");
      scanf("%s", infix);
      printf("The postfix expression is:\n");
      postfix(infix);
      printf("\n");
}
void postfix( char infix[])
{
      int stack[20], top=0, n=0;
      int isp[]= {0, 4, 1, 1, 2, 2, 2, 3, 0};
      int icp[]= {5, 4, 1, 1, 2, 2, 2, 3, 0};
      char symbol;
      precedence token;
      stack[top]= eos;
      for(token=get_token(&symbol, &n, infix); token!=eos;
token=get_token(&symbol, &n, infix))
      {
            if(token==opnd)
                  printf("%c", symbol);
            else if(token==rparen)
            {
                  while(stack[top]!=lparen)
                        print_token(pop(stack, &top));
                  pop(stack, &top);
            }
            else
            {
                  while(isp[stack[top]]>=icp[token])
                        print_token(pop(stack, &top));
                  push(token, stack, &top);
            }
      }
      while((token=pop(stack, &top))!=eos)
            print_token(token);
}
```

```
precedence get_token(char * symbol, int *n, char infix[])
{
      *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 power;
            case '\0': return eos;
            default: return opnd;
      }
}
void print_token(precedence item)
{
      switch(item)
      {
            case plus: printf("+"); break;
            case minus: printf("-"); break;
            case mul: printf("*"); break;
            case divi: printf("/"); break;
            case mod: printf("%%"); break;  //It is necessary to put %%
instead of % for compilation in gcc
            case power: printf("^"); break;
      }
      return;
}
void push(int value, int stack[], int *top)
{
      stack[++(*top)]= value;
}
int pop(int stack[], int *top)
{
      return stack[(*top)--];
}
```

```
/* 5. Design, Develop and Implement a Program in C for the following Stack
Application
a. Evaluation of Suffix expression with single digit operands and
operators: +, -, *, /, %, ^ */


#include<stdio.h>  //Incase you use stdlib.h you'll have to change div to
divi since div is a built-in function
#include<math.h>   //Use [cc suffix_evaluation.c -o a.out -lm] for
compiling programs containing math.h  {O/P command: ./a.out}
typedef enum{ plus, minus, multi, div, mod, power, eos, opnd}precedence;
int evaluate( char postfix[]);
precedence get_token( char *s, int *n, char post[]);
void push(int n, int *top, int stack[]);
int pop(int *top, int stack[]);
int operation(int o1, int o2, precedence token);
void main()
{
      char post[20];
      printf("Enter the postfix expression:\n");
      scanf("%s", post);
      printf("Result:\n%d\n",evaluate(post));
}
int evaluate( char postfix[])
{
      int n=0, op1, op2;
      int stack[10], top=-1;
      char symbol;
      precedence token= get_token( &symbol, &n, postfix);
      while(token!=eos)
      {
            if(token==opnd)
                  push( symbol-'0', &top, stack);
            else
            {
                  op2= pop(&top, stack);
                  op1= pop(&top, stack);
                  push(operation(op1, op2, token), &top, stack);
            }
            token= get_token( &symbol, &n, postfix);
      }
      return stack[0];
}
precedence get_token( char *s, int *n, char post[])
{
      *s= post[(*n)++];
      switch(*s)
      {
            case '+': return plus;
            case '-': return minus;
            case '*': return multi;
            case '/': return div;
            case '%': return mod;
            case '^': return power;
```

```
            case '\0': return eos;
            default: return opnd;
      }
}
void push(int n, int *top, int stack[])
{
      stack[++(*top)]=n;
}
int pop(int *top, int stack[])
{
      return stack[(*top)--];
}
int operation(int o1, int o2, precedence token)
{
      switch(token)
      {
            case plus: return o1+o2;
            case minus: return o1-o2;
            case multi: return o1*o2;
            case div: return o1/o2;
            case mod: return o1%o2;
            case power: return pow(o1,o2);
      }
      return 0;
}
```

]

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

#include<stdio.h>
#include<stdlib.h>
void TOH( int n, char source, char destination, char spare);
void main()
{
      int n;
      printf("Enter number of rings:\n");
      scanf("%d", &n);
      TOH(n, 'A', 'C', 'B');
}
void TOH( int n, char source, char destination, char spare)
{
      if(n==1)
            printf("Move from %c to %c\n",source, destination);
      else
      {
            TOH(n-1, source, spare, destination);
            TOH(1, source, destination, spare);
            TOH(n-1, spare, destination, source);
      }
}
```

```
/* 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 50
void insert(char ele, int *r, char q[], int *count);
void delete(int *f, char q[], int *count);
void display(int f, char q[], int count);
void main()
{
      int choice, front=0, rear=-1, count=0;
      char q[SIZE], ele;
      for(;;)
      {
            printf("Enter\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");
            scanf("%d", &choice);
            switch(choice)
            {
                  case 1: if(count==(SIZE-1))
                              printf("Queue is full!!!\n");
                        else
                        {
                              printf("Enter character to be inserted:\n");
                              scanf("%s", &ele); // It is necessary to put
%s instead of %c 'cause of the compiler
                              insert(ele, &rear, q, &count);
                        }
                        break;
                  case 2: if(count==0)
                              printf("Queue is empty!!!\n");
                        else
                              delete(&front, q, &count);
                        break;
                  case 3: if(count==0)
                              printf("Queue is empty!!!\n");
                        else
                              display(front, q, count);
                        break;
                  case 4: exit(0);
            }
      }
}
void insert(char ele, int *r, char q[], int *count)
{
```

```
	*r=(*r+1)%SIZE;
	q[*r]=ele;
	(*count)++;
}
void delete(int *f, char q[], int *count)
{
	printf("Deleted character:\n%c\n", q[*f]);
	(*count)--;
	*f=(*f+1)%SIZE;
}
void display(int f, char q[], int count)
{
	int i;
	printf("Elements in the queue:\n");
	for(i=1; i<=count; i++)
	{
		printf("%c\t", q[f]);
		f=(f+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 / Deletion at End of SLL
d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
e. Exit */



#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
      char USN[10], name[20], branch[10];
      int sem;
      long int ph;
      struct node *link;
}NODE;
typedef struct head_node
{
      int count;
      NODE *link;
}HEAD;
void ins_front(HEAD *head);
void ins_rear(HEAD *head);
void del_front(HEAD *head);
void del_rear(HEAD *head);
void display(HEAD *head);
void main()
{
      HEAD *head=(HEAD*)malloc(sizeof(HEAD));
      int choice;
      head->count=0;
      head->link=NULL;
      for(;;)
      {
            printf("Enter\n1. Insert at front\n2. Insert at rear\n3.
Delete from front\n4. Delete from rear\n5. Display\n6. Exit\n");
            scanf("%d", &choice);
            switch(choice)
            {
                  case 1: ins_front(head); break;
                  case 2: ins_rear(head); break;
                  case 3: del_front(head); break;
                  case 4: del_rear(head); break;
                  case 5: display(head); break;
                  case 6: exit(0);
            }
      }
}
void ins_front(HEAD *head)
{
```

```
	NODE *newN=(NODE*)malloc(sizeof(NODE));
	printf("Enter USN, NAME, BRANCH, SEM, PHONE of the student:\n");
	scanf("%s%s%s%d%ld", (newN->USN), (newN->name), (newN->branch),
&(newN->sem), &(newN->ph));
	newN->link= head->link;
	(head->count)++;
	head->link= newN;
}
void ins_rear(HEAD *head)
{
	NODE *newN=(NODE*)malloc(sizeof(NODE));
	NODE *temp;
	printf("Enter USN, NAME, BRANCH, SEM, PHONe of the student:\n");
	scanf("%s%s%s%d%ld", (newN->USN), (newN->name), (newN->branch),
&(newN->sem), &(newN->ph));
	newN->link= NULL;
	if(head->link==NULL)
	{
		head->link=newN;
		(head->count)++;
		return;
	}
	temp= head->link;
	while(temp->link!=NULL)
		temp= temp->link;
	temp->link= newN;
	(head->count)++;
}
void del_front(HEAD *head)
{
	NODE *temp;
	if(head->link==NULL)
	{
		printf("List Empty!!!\n");
		return;
	}
	temp= head->link;
	printf("Deleted record:\n");
	printf("%s\t%s\t%s\t%d\t%ld\n", (temp->USN), (temp->name), (temp-
>branch), (temp->sem), (temp->ph));
	head->link= temp->link;
	(head->count)--;
	free(temp);
}
void del_rear(HEAD *head)
{
	NODE *present, *previous;
	if(head->link==NULL)
	{
		printf("List Empty!!!\n");
		return;
	}
	present= head->link;
	if(present->link==NULL)
```

```
	{
		printf("List contains one record.\n");
		head->link=NULL;
	}
	else
	{
		while(present->link!=NULL)
		{
			previous= present;
			present= present->link;
		}
		previous->link=NULL;
	}
	printf("Deleted record:\n");
	printf("%s\t%s\t%s\t%d\t%ld\n", (present->USN), (present->name),
(present->branch), (present->sem), (present->ph));
	(head->count)--;
	free(present);
}
void display(HEAD *head)
{
	NODE *temp;
	if(head->link==NULL)
	{
		printf("List Empty!!!\n");
		return;
	}
	else
	{
		printf("Number of nodes: %d\n", head->count);
		printf("Contents of the list\n");
		temp= head->link;
		while(temp!=NULL)
		{
			printf("%s\t%s\t%s\t%d\t%ld\n", (temp->USN), (temp-
>name), (temp->branch), (temp->sem),  (temp->ph));
			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 emp
{
      char SSN[10], name[20], department[10], designation[20];
      float salary;
      long int ph;
      struct emp *llink, *rlink;
}NODE;
typedef struct head_node
{
      int count;
      NODE *link;
}HEAD;
void ins_front(HEAD * head);
void ins_rear(HEAD * head);
void del_front(HEAD * head);
void del_rear(HEAD * head);
void display(HEAD * head);
void main()
{
      HEAD * head=(HEAD*)malloc(sizeof(HEAD));
      int choice;
      head->count= 0;
      head->link= NULL;
      for(;;)
      {
            printf("Enter\n1. Insert at front\n2. Insert at rear\n3.
Delete from front\n4. Delete from rear\n5. Display the list\n6. Choose
options 1 and 3 or 4  OR 2 and 3 or 4 for demonstration of Deque\n7.
exit\n");
            scanf("%d", &choice);
            switch(choice)
            {
                  case 1: ins_front(head);
                        break;
                  case 2: ins_rear(head);
                        break;
                  case 3: del_front(head);
                        break;
                  case 4: del_rear(head);
                        break;
                  case 5: display(head);
                        break;
```

```
                case 7: exit(0);
            }
      }
}
void ins_front(HEAD * head)
{
      NODE *newN=(NODE*)malloc(sizeof(NODE));
      printf("Enter SSN, Employee name, Department, Designation, Salary,
Phone:\n");
      scanf("%s%s%s%s%f%ld", (newN->SSN), (newN->name), (newN-
>department), (newN->designation), &(newN->salary), &(newN->ph));
      newN->rlink=head->link;
      //newN->llink=head;
      head->link= newN;
      (head->count)++;
      return;
}
void ins_rear(HEAD * head)
{
      NODE *newN=(NODE*)malloc(sizeof(NODE));
      NODE *temp;
      printf("Enter SSN, Employee name, Department, Designation, Salary,
Phone:\n");
      scanf("%s%s%s%s%f%ld", (newN->SSN), (newN->name), (newN-
>department), (newN->designation), &(newN->salary), &(newN->ph));
      newN->rlink=NULL;
      if(head->link==NULL)
      {
            head->link=newN;
            //newN->llink=head;
            return;
      }
      temp=head->link;
      while(temp->rlink!=NULL)
      {
            temp=temp->rlink;
      }
      temp->rlink=newN;
      //newN->llink=temp;
      (head->count)++;
}
void del_front(HEAD * head)
{
      NODE *temp;
      if(head->link==NULL)
      {
            printf("List Empty!!!\n");
            return;
      }
      temp=head->link;
      printf("Deleted record:\n%s\t%s\t%s\t%s\t%f\t%ld\n",(temp->SSN),
(temp->name), (temp->department), (temp->designation),(temp->salary),
(temp->ph));
      head->link=temp->rlink;
```

```
      free(temp);
      (head->count)--;
}
void del_rear(HEAD * head)
{
      NODE *present, *previous;
      if(head->link==NULL)
      {
            printf("List Empty!!!\n");
            return;
      }
      present=head->link;
      while(present->rlink!=NULL)
      {
            previous=present;
            present=present->rlink;
      }
      printf("Deleted record:\n%s\t%s\t%s\t%s\t%f\t%ld\n",(present->SSN),
(present->name), (present->department),       (present->designation),
(present->salary), (present->ph));
      previous->rlink=NULL;
      free(present);
      (head->count)--;
}
void display( HEAD * head)
{
      NODE *temp;
      if(head->link==NULL)
      {
            printf("List Empty!!!\n");
            return;
      }
      temp=head->link;
      printf("Number of nodes: %d\n", (head->count));
      printf("Records:\nSSN\tNAME\tDEPARTMENT\tDESIGNATION\tSALARY\tPHONE\
n");
      while(temp!=NULL)
      {
            printf("%s\t%s\t%s\t%s\t%f\t%ld\n",(temp->SSN), (temp->name),
(temp->department), (temp->designation),(temp->salary), (temp->ph));
            temp=temp->rlink;
      }
}
```

```
/*Design, develop and implement a C program for the following operations
on a singly linked list with header nodes
a. Represent and evaluate a polynomial P(x, y, z)
b. Find the sum of 2 polynomials POLY1(x, y, z) and POLY2(x, y, z) and
store the result in POLYSUM(x, y, z)


P.S: To compile code containing math.h--- "  cc polynomial.c -o a.out -lm
"
*/


#include<stdio.h>
#include<stdlib.h>
#include<math.h>
typedef struct poly_node
{
      float coeff;
      int expx, expy, expz;
      struct poly_node * link;
}POLY;
POLY * getnode();
void read_poly( int n, POLY * head);
void print_poly(POLY *head);
POLY * add_poly( POLY * h1, POLY * h2);
int COMP(POLY *t1, POLY *t2);
void attach(float cf, POLY *temp, POLY **tres);
POLY * delete(POLY * head, POLY * temp);
void evaluate( POLY *head);
void main()
{
      int n1, n2;
      POLY *head1=getnode();
      POLY *head2=getnode();
      POLY *head3=getnode();
      head1->link= head1;
      head2->link= head2;
      head3->link= head3;
      printf("Enter number of terms in first and second polynomial:\n");
      scanf("%d%d", &n1, &n2);
      printf("Enter first polynomial:\n");
      read_poly(n1, head1);
      printf("Enter second polynomial:\n");
      read_poly(n2, head2);
      printf("The first polynomial:\n");
      print_poly(head1);
      printf("The second polynomial:\n");
      print_poly(head2);
      //Adding two polynomials
      head3= add_poly( head1, head2);
      printf("Resultant polynomial:\n");
      print_poly(head3);
      //Evaluation
      printf("Evaluation of first polynomial:\n");
```

```
	evaluate(head1);
	printf("\n");
}
POLY * getnode()
{
	POLY * temp= (POLY*)malloc(sizeof(POLY));
	return temp;
}
void read_poly( int n, POLY * head)
{
	POLY *newN, *temp;
	int i;
	temp= head;
	for(i=0;i<n;i++)
	{
		newN= getnode();
		printf("Enter coefficient and exponent:\n");
		scanf("%f%d%d%d", &(newN->coeff), &(newN->expx), &(newN-
>expy), &(newN->expz));
		temp->link= newN;
		temp=temp->link;
	}
	temp->link= head;
}
void print_poly(POLY *head)
{
	POLY * temp= head->link;
	while(temp!=head)
	{
		printf("(%2.2fx^%dy^%dz^%d) + ", temp->coeff, temp->expx,
temp->expy, temp->expz);
		temp= temp->link;
	}
	printf("\n");
}
POLY * add_poly( POLY * h1, POLY * h2)
{
	POLY *temp1, *temp2;
	POLY *result=getnode();
	POLY *tempres= result;
	double sum;
	temp1= h1->link;
	while(temp1!=h1)
	{
		temp2= h2->link;
		while(temp2!=h2)
		{
			switch(COMP(temp1, temp2))
			{
				case 1: //Exponents are equal.
					sum= temp1->coeff+temp2->coeff;
					if(sum)
						attach(sum, temp1, &tempres);
					h2=delete(h2, temp2);
```

```
                              temp2=h2->link;
                              temp1=temp1->link;
                              break;
                        case 2: //Exponents are unequal
                              temp2=temp2->link;
                              break;
                  }
            }
            if(temp1!=h1)
            {
                  attach(temp1->coeff, temp1, &tempres);
                  temp1=temp1->link;
            }
      }
      temp2= h2->link;
      while(temp2!=h2)
      {
            attach(temp2->coeff, temp2, &tempres);
            temp2= temp2->link;
      }
      tempres->link= result;
      return result;
}
int COMP(POLY *t1, POLY *t2)
{
      if((t1->expx==t2->expx)&&(t1->expy==t2->expy)&&(t1->expz==t2->expz))
            return 1;
      else
            return -1;
}
void attach(float cf, POLY *temp, POLY **tres)
{
      POLY *newN= getnode();
      newN->coeff= cf;
      newN->expx= temp->expx;
      newN->expy= temp->expy;
      newN->expz= temp->expz;
      (*tres)->link= newN;
      *tres= newN;
}
POLY * delete(POLY * head, POLY * temp)
{
      POLY *prev, *pres;
      prev=head;
      pres=head->link;
      while(pres!=temp)
      {
            prev=pres;
            pres= pres->link;
      }
      prev->link=pres->link;
      free(temp);
      return head;
}
```

```
void evaluate( POLY *head)
{
      POLY *temp= head->link;
      double x, y, z, sum=0;
      double tx, ty, tz;
      printf("Enter values of x, y and z:\n");
      scanf("%lf%lf%lf", &x, &y, &z);
      while(temp!=head)
      {
            tx=(double)temp->expx;
            ty=(double)temp->expy;
            tz=(double)temp->expz;
            sum+=((temp->coeff)*pow(x,tx)*pow(y,ty)*pow(z,tz));
            temp=temp->link;
      }
      printf("Result:\n%lf\n", sum);
}
```

```
/* 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, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate
message
e. Exit  */

#include<stdio.h>
#include<stdlib.h>
typedef struct tree
{
      int data;
      struct tree *rlink, *llink;
}TNODE;
TNODE * getnode();
TNODE * insert(int ele, TNODE * root);
void inorder(TNODE * root);
void preorder(TNODE * root);
void postorder(TNODE * root);
int search(TNODE * root, int key);
void main()
{
      TNODE *root= NULL;
      int choice, ele, key, flag;
      for(;;)
      {
            printf("Enter\n1. Insert\n2. Inorder\n3. Preorder\n4.
Postorder\n5. Search\n6. Exit\n");
            scanf("%d", &choice);
            switch(choice)
            {
                  case 1: printf("Enter element to be inserted:\n");
                        scanf("%d", &ele);
                        root= insert(ele, root);
                        break;
                  case 2: if(root==NULL)
                              printf("Tree is empty\n");
                        else
                        {
                              printf("The contents are:\n");
                              inorder(root);
                        }
                        break;
                  case 3: if(root==NULL)
                              printf("Tree is empty\n");
                        else
                        {
                              printf("The contents are:\n");
                              preorder(root);
                        }
                        break;
                  case 4: if(root==NULL)
                              printf("Tree is empty\n");
```

```
                        else
                        {
                              printf("The contents are:\n");
                              postorder(root);
                        }
                        break;
                  case 5: printf("Enter the node to be searched:\n");
                        scanf("%d", &key);
                        flag= search(root, key);
                        if(flag==-1)
                              printf("Unsuccessful search!!!\n");
                        else
                              printf("Successful search!!!\n");
                        break;
                  case 6: exit(0);
            }
      }
}
TNODE * getnode()
{
      TNODE *temp= (TNODE*)malloc(sizeof(TNODE));
      if(temp==NULL)
      {
            printf("Out of memory!!!\n");
            return NULL;
      }
      return temp;
}
TNODE * insert( int ele, TNODE * root)
{
      TNODE *newN= getnode();
      //TNODE *previous, *present;
      newN->data= ele;
      newN->rlink= newN->llink= NULL;
      if(root==NULL)
            return newN;
      if(ele<root->data)
            root->llink= insert(ele, root->llink);
      if(ele>root->data)
            root->rlink= insert(ele, root->rlink);
      return root;
}
void inorder(TNODE * root)
{
      if(root!=NULL)
      {
            inorder(root->llink);
            printf("%d\n", root->data);
            inorder(root->rlink);
      }
}
void preorder(TNODE * root)
{
      if(root!=NULL)
```

```
	{
		printf("%d\n", root->data);
		preorder(root->llink);
		preorder(root->rlink);
	}
}
void postorder(TNODE * root)
{
	if(root!=NULL)
	{
		postorder(root->llink);
		postorder(root->rlink);
		printf("%d\n", root->data);
	}
}
int search( TNODE * root, int key)
{
	if(root!=NULL)
	{
		if(root->data==key)
			return key;
		if(key < root->data)
			return search(root->llink, key);
		return search(root->rlink, key);
	}
	return -1;
}
```

```
/* 11. Design, Develop and Implement 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 */


#include<stdio.h>
#include<stdlib.h>
#define SIZE 20
void bfs(int n, int source, int amat[][SIZE], int visited[]);
void main()
{
      int n, amat[SIZE][SIZE], source, visited[SIZE], i, j;
      printf("Enter number of vertices:\n");
      scanf("%d", &n);
      printf("Enter the adjacency matrix:\n");
      for(i=0; i<n; i++)
      {
            for(j=0; j<n; j++)
            {
                  scanf("%d", &amat[i][j]);
            }
      }
      printf("The adjacency matrix is:\n");
      for(i=0; i<n; i++)
      {
            for(j=0; j<n; j++)
            {
                  printf("%d\t", amat[i][j]);
            }
            printf("\n");
      }
      printf("Give the source:\n");
      scanf("%d", &source);
      for(i=0; i<n; i++)
            visited[i]=0;
      bfs(n, source, amat, visited);
      for(i=0;i<n;i++)
      {
                  if(visited[i]==0)
                        printf("%d is not reachable\n", i);
                  else
                        printf("%d is reachable\n",i);
      }
}
void bfs(int n, int source, int amat[][SIZE], int visited[])
{
      int q[SIZE], r=0, f=0, u, v;
      visited[source]=1;
      q[r]=source;
      while(f<=r)
```

```
        {
                u= q[f++];
                for(v=0;v<n;v++)
                {
                        if((amat[u][v]==1)&(visited[v]==0))
                        {
                                q[++r]=v;
                                visited[v]=1;
                        }
                }
        }
}
```

/*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. Design and develop a Program in C that uses Hash function H: K â†’ 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.*/

```
#include<stdio.h>
#include<stdlib.h>
typedef struct emp
{
      int empno;
      char name[20];
      int sal;
}EMPLOYEE;
void main()
{
      EMPLOYEE E;
      FILE *fp;
      int n, i, s=(2*sizeof(int)+20), minusone=-1, choice, flag, index,
indexcopy, id;
      printf("Enter number of records:\n");
      scanf("%d", &n);
      fp= fopen("emp.txt", "w+");
      for(i=0; i<n; i++)
      {
            fwrite(&minusone, sizeof(int), 1, fp);
            fseek(fp, s-sizeof(int), SEEK_CUR);
      }
      while(1)
      {
            printf("Enter\n1. Add Record\n2. Display Records\n3.
Exit\n");
            scanf("%d", &choice);
            flag=0;
            switch(choice)
            {
                  case 1: printf("Enter Employee number, Employee name
and Salary:\n");
                        scanf("%d%s%d", &E.empno, E.name, &E.sal);
                        //Hash function
                        index= indexcopy= (E.empno % n);
                        fseek(fp, s*index, SEEK_SET);
                        fread(&id, sizeof(int), 1, fp);
                        //Linear Probing
                        while(id!=-1)
                        {
                              index++;
```

```
                        fseek(fp, s*index, SEEK_SET);
                        flag=1;
                        if(index==n)
                              index=0;
                        if(index==indexcopy)
                        {
                              printf("FILE FULL!!\n");
                              break;
                        }
                        fread(&id, sizeof(int), 1, fp);
                  }
                  if(!((index==indexcopy)&&flag))
                  {
                        fseek(fp, s*index, SEEK_SET);
                        fwrite(&E, sizeof(EMPLOYEE), 1, fp);
                  }
                  break;
            case 2: printf("Records are:\n");
                  for(index=0; index<n; index++)
                  {
                        fseek(fp, s*index, SEEK_SET);
                        fread(&E.empno, sizeof(int), 1, fp);
                        printf("%d\t",E.empno);
                        if(E.empno!=-1)
                        {
                              fread(E.name, 20, 1, fp);
                              fread(&E.sal, sizeof(int), 1, fp);
                              printf("%s\t%d\n", E.name, E.sal);
                        }
                  }
                  break;
            case 3: fclose(fp);
                  exit(0);
      }
   }
}
```