1. Develop a Structure 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),

Third field is 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.

```
#include <stdio.h>
#include <stdlib.h>                          //stdlib.h for malloc() and free()
struct day                                   // Structure to initialise variables- day name, date and activity
{
        char* dayname;
        int date;
        char* activity;
};

struct day* createday()                      // Function to allocates a memory dynamically using malloc
{
        struct day* newday = (struct day*)malloc(sizeof(struct day));
        newday->dayname = (char*)malloc(sizeof(char));
        newday->activity = (char*)malloc(sizeof(char));
        printf("Enter day name: ");
        scanf("%s", newday->dayname);
        printf("Enter date: ");
        scanf("%d", &(newday->date));
        printf("Enter activity description: ");
        scanf(" %[^\n]", newday->activity);
        return newday;
}

void read(struct day* calendar[], int size)  // Function to read calendar data from keyboard
{
        for (int i = 0; i < size; i++)
        {
                printf("Enter details for day %d:\n", i + 1);
                calendar[i] = createday();
        }
}

void display(struct day* calendar[], int size)  // Function to display calendar information-day, date, activity
{
        printf("\nWeek's Activity Details:\n");
        for (int i = 0; i < size; i++)
        {
                printf("Day %d:\n", i + 1);
                printf("Day Name: %s\n", calendar[i]->dayname);
                printf("Date: %d\n", calendar[i]->date);
                printf("Activity Description: %s\n", calendar[i]->activity);
                printf("\n");
        }
}
```

```
void freememory(struct day* calendar[], int size)          // Function to delete the memory allocated for calendar
{
        for (int i = 0; i < size; i++)
        {
                free(calendar[i]->dayname);                //free keyword used to delete dynamic allocated memory.
                free(calendar[i]->activity);
                free(calendar[i]);
        }
}

int main()
{
        struct day* week[7];                               //Since seven days
        read(week, 7);
        display(week, 7);
        freememory(week, 7);                               // Delete allocated memory
        return 0;
}
```

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.

c. Report suitable messages in case PAT does not exist in STR. Don't use Built-in functions.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main()
{
	int i, j, k, textlength, patternlength, replacelength;
	char *text=(char *)malloc(sizeof(char));
	char *pattern=(char *)malloc(sizeof(char));
	char *replace=(char *)malloc(sizeof(char));

	printf("Enter the text:");
	scanf(" %[^\n]",text);
	printf("Enter the pattern:");
	scanf(" %[^\n]",pattern);
	printf("Enter the replace string:");
	scanf(" %[^\n]",replace);

	textlength = strlen(text);
	patternlength = strlen(pattern);
	replacelength=strlen(replace);

	for (i = 0; i <= textlength - patternlength; i++)
	{
		for (j = 0; j < patternlength; j++)
		{
			if (text[i + j] != pattern[j])
			{
				break;
			}
		}
		if (j == patternlength)
		{
			printf("Position fount at %d\n", i);
			break;
		}
	}
	if (patternlength==replacelength)
	{
		for(k=0; k<replacelength; k++)
		{
			text[i]=replace[k];
			i++;
		}
		printf("\nUpdated Text is: %s", text);
	}
	else
	{
		printf("Not possible to replace the string");
	}
}
```

3. Develop menu driven Program in C for 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>

int size, choice, f, top=-1, num, k, i, stack[6], rev[6];

void push ();
void pop ();
void display ();
int pali ();

void main ()
{
        printf ("Enter the size for stack\n");
        scanf ("%d", &size);
        printf ("\n-----MENU----\n 1.Push    2.Pop     3.Display      4.Check for Palindrome      5.Exit\n");
        while (1)
        {
                printf ("Enter the choice\n");
                scanf ("%d", &choice);
                switch (choice)
                {
                        case 1: push ();
                                break;
                        case 2: pop ();
                                break;
                        case 3: display ();
                                break;
                        case 4: f = pali ();
                        if (f == 1)
                                printf ("It's Palindrome\n");
                        else
                                printf ("It's not a Palindrome\n");
                        break;
                        case 5: exit (0);
                        default: printf ("Wrong choice...\n");
                }
         }
}

void push ()
{
        if (top == (size - 1))
        {
                printf ("Stack Overflow\n");
        }
        else
```

```
        {
                printf ("Enter the number to be pushed\n");
                scanf ("%d", &num);
                top++;
                stack[top] = num;
         }
}

void pop ()
{
        if (top == -1)
        {
                printf ("Stack Underflow\n");
        }
        else
        {
                num = stack[top];
                printf ("Popped element is %d\n", num);
                top--;
        }
}

void display ()
{
        if (top == -1)
        {
                printf ("Stack is Empty\n");
        }
        else
        {
                printf ("Stack Contents....\n");
                for (i = top; i >= 0; i--)
                {
                        printf ("%d\n", stack[i]);
                }
         }
}

int pali()
{
        int flag=1;
        for (i = top; i >= 0; i--)
        {
                rev[k++] = stack[i];
        }
        for (i = top; i >= 0; i--)
        {
                if (stack[i] != rev[--k])
                {
                        flag = 0;
                }
        }
        return flag;
}
```

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<string.h>

int pre(char symbol)                                    //Operator precedence
{
        switch (symbol)
        {
                case '+':
                case '-': return 2;
                case '*':
                case '/':
                case '%': return 4;
                case '^':
                case '$': return 5;
                case '(': return 0;
                case '#': return -1;
                default: return 8;
        }
}

int inpre(char symbol)                                  //incoming operator precedence
{
        switch (symbol)
        {
                case '+':
                case '-': return 1;
                case '*':
                case '/':
                case '%': return 3;
                case '^':
                case '$': return 6;
                case '(': return 3;
                case ')': return 0;
                default: return 7;
        }
}

void infixpostfix(char infix[], char postfix[])
{
        int top=-1, j=0, i;
        char s[30], symbol;
        s[++top] = '#';
        for(i=0; i < strlen(infix); i++)
        {
                symbol = infix[i];
                while (pre(s[top]) > inpre(symbol))
                {
                        postfix[j] = s[top--];
                        j++;
                }
                if(pre(s[top])!= inpre(symbol))
                s[++top] = symbol;
                else
```

```
                top--;
        }
        while(s[top]!= '#')
        postfix[j++] = s[top--];
        postfix[j] = '\0';
}

void main()
{
        char infix[20], postfix[20];
        printf("Enter a valid infix expression\n") ;
        scanf ("%s", infix);
        infixpostfix (infix, postfix);
        printf("\nThe postfix expression is:\n");
        printf ("%s", postfix);
}
```

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

5a. Evaluation of suffix expression.

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

int i, top=-1, op1, op2, res, s[20];          //operand 1 and 2, result, stack of size 20
char postfix[90], symb;                       //postfix expression, symbols or operators

void push(int item)
{
        s[++top] = item;                      //push operation
}

int pop()
{
        int item;
        item = s[top--];                      //pop operation
        return item;
}

void main()
{
        printf("Enter a valid postfix expression:");
        scanf("%s", postfix);
        for(i=0; postfix[i]!='\0'; i++)
        {
                symb = postfix[i];
                if(isdigit(symb))             //header file ctype.h required
                {
                        push(symb - '0');     //if we received digit, then push into stack
                }
                else
                {
                        op2 = pop();
                        op1 = pop();          //else pop top most two operands & perform arithmetic operation
                        switch(symb)
                        {
                                case '+': push(op1+op2);
                                          break;
                                case '-':  push(op1-op2);
                                          break;
                                case '*': push(op1*op2);
                                          break;
                                case '/':  push(op1/op2);
                                          break;
                                case '%': push(op1%op2);
                                          break;
                                case '^':  push(pow(op1, op2));       //header file math.h required
                                          break;
                                default:  push(0);
                        }
                }
```

```
        }
        res = pop();
        printf("\n Result = %d", res);
}
```

5b. Towers of Hanoi

```
#include <stdio.h>
void towers(int, char, char, char);
int main()
{
        int n;
        printf("Enter the number of disks : ");
        scanf("%d", &n);
        printf("The sequence of moves involved in the Tower of Hanoi are :\n");
        towers(n, 'A', 'B', 'C');
        return 0;
}

void towers(int n, char A, char B, char C)
{
        if (n == 1)
        {
                printf("\nMove disk 1 from peg %c to peg %c", A, C);
                return;
        }
        towers(n-1, A, C, B);
        printf("\nMove disk %d from peg %c to peg %c", n, A, C);
        towers(n-1, B, A, C);
}
```

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

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

#define size 5
char cq[size],ele;
int front=-1, rear=-1, ch;
void enque();
void deque();
void display();

void main()
{
        printf("1.Insert   2.Delete  3.Display   4.Exit");
        while(1)
        {
                printf("\nEnter your choice:");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1: enque();
                                break;
                        case 2:deque();
                                break;
                        case 3:display();
                                break;
                        case 4: exit(0);
                }
        }
}

void enque()
{
        if(front==(rear+1)%size)
        {
                printf("Circular Queue Overflow");
                return;
        }
        if(front==-1)
        front++;
        printf("Enter the character to the circular queue");
        scanf("\n%c", &ele);
        rear = (rear+1)%size;
        cq[rear] =ele;
}

void deque()
{
```

```
        char item;
        if(front == -1)
        {
                printf("Circular Queue Underflow");
                return;
        }
        else if(front == rear)
        {
                item=cq[front];
                printf("Deleted element is: %c", item);
                front=-1;
                rear=-1;
        }
        else
        {
                item =cq[front];
                printf("Deleted element is: %c", item);
                front = (front+1)%size;
        }
}

void display()
{
        int i;
        if(front==-1)
        printf("Circular Queue is Empty");
        else
        {
                printf("Elements of the circular queue are:");
                for(i=front; i!=rear; i=(i+1)%size)
                {
                        printf("%c\t", cq[i]);
                }
                printf("%c", cq[rear]);
        }
}
```

7. Develop a menu driven Program in C for 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 and Deletion at End of SLL

d. Perform Insertion and Deletion at Front of SLL(Demonstration of stack)

e. Exit

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

struct node
{
        char usn[25], name[25],programme[25];
        int sem;
        long int phone;
        struct node *next;
};
typedef struct node *NODE;
NODE first = NULL;
int count=0;

NODE create()
{
        NODE snode;
        snode = (NODE)malloc(sizeof(struct node));
        printf("\n Enter the USN, Name, Programme, sem, PhoneNo of the student:");
        scanf("%s %s %s %d %ld",snode->usn, snode->name, snode->programme, &snode->sem, &snode->phone);
        snode->next=NULL;
        count++;
        return snode;
}

NODE insertfirst()
{
        NODE temp;
        temp = create();
        if(first == NULL)
        {
                return temp;
        }
        temp->next = first;
        return temp;
}

NODE deletefirst()
{
        NODE temp;
        if(first == NULL)
        {
                printf("Linked list is empty");
                return NULL;
        }
        if(first->next == NULL)
        {
                printf("\nStudent node with usn:%s is deleted ",first->usn);
```

```
                count--;
                free(first);
                return NULL;
        }
        temp = first;
        first = first->next;
        printf("\nStudent node with usn:%s is deleted",temp->usn);
        count--;
        free(temp);
        return first;
}

NODE insertend()
{
        NODE cur,temp;
        temp = create();
        if(first == NULL)
        {
                return temp;
        }
        cur = first;
        while(cur->next !=NULL)
        {
                cur = cur->next;
        }
        cur->next = temp;
        return first;
}

NODE deleteend()
{
        NODE cur, prev;
        if(first == NULL)
        {
                printf("\n Linked List is empty");
                return NULL;
        }
        if(first->next == NULL)
        {
                printf("\n Student node with the usn:%s is deleted", first->usn);
                free(first);
                count--;
                return NULL;
        }
        prev = NULL;
        cur = first;
        while(cur->next !=NULL)
        {
                prev = cur;
                cur = cur->next;
        }
        printf("\nStudent node with the usn:%s is deleted", cur->usn);
        free(cur);
        prev->next = NULL;
        count--;
        return first;
}
```

```
void display()
{
        NODE cur;
        int num=1;
        if(first == NULL)
        {
                printf("\nLinked list is empty\n");
                return;
        }
        printf("\nThe contents of SLL: \n");
        cur = first;
        while(cur!=NULL)
        {
                printf("\n %d USN:%s Name:%s Programme:%s Sem:%d Ph:%ld", num, cur->usn, cur->name,
                cur->programme, cur->sem, cur->phone);
                cur = cur->next;
                num++;
        }
        printf("\n Number of students or nodes is %d \n", count);
}

void stackdemo()
{
        int ch;
        printf("\n~~~Stack Demo using SLL~~~\n1:Push operation\n2: Pop operation\n3:Display \n4:Exit \n");
        while(1)
        {
                printf("\n Enter your choice for stack demo : ");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1: first = insertfirst();
                                break;
                        case 2: first = deletefirst();
                                break;
                        case 3: display();
                                break;
                        default: return;
                }
        }
        return;
}

int main()
{
        int ch, i, n;
        printf("\nMenu\n 1:Create SLL of Student Node\n 2:Display Status\n 3:Insert At End\n 4:Delete At End");
        printf("\n 5:Stack Demo using SLL(Insertion and Deletion at First)\n 6:Exit \n");
        while(1)
        {
                printf("\n Enter your choice:");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1 :printf("\nEnter the number of students (or nodes):");
                                scanf("%d", &n);
                                for(i=1;i<=n; i++)
                                first = insertfirst();
```

```
                        break;
                case 2: display();
                        break;
                case 3: first = insertend();
                        break;
                case 4: first = deleteend();
                        break;
                case 5: stackdemo();
                        break;
                case 6: exit(0);
                default: printf("Enter valid choice ");
            }
        }
}
```

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

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

struct node
{
        char ssn[15], name[20], dept[5], des[10];
        int salary;
        long int ph;
        struct node *prev, *next;
}*first=NULL;
struct node *last, *temp1, *temp2;

void create(char ssn[15],char name[20],char dept[5],char des[10],int salary, long int ph)
{
        temp1=(struct node *)malloc(1*sizeof(struct node));
        strcpy(temp1->ssn, ssn);
        strcpy(temp1->name, name);
        strcpy(temp1->dept, dept);
        strcpy(temp1->des, des);
        temp1->salary=salary;
        temp1->ph=ph;
        temp1->prev=NULL;
        temp1->next=NULL;

        if(first==NULL)
        {
                first=last=temp1;
        }
        else
        {
                last->next=temp1;
                temp1->next=NULL;
                temp1->prev=last;
                last=temp1;
        }
}

void display()
{
        int count=0;
        temp1=first;
        if(first==NULL)
        {
                printf("empty list\n");
                return;
        }
        printf("Employee Details .....\n");
```

```
        while(temp1!=NULL)
        {
                printf("SSN:%s\nNAME:%s\nDEPT:%s\nDESIGNATION:%s\nSALARY:%d\nPH:%ld\n",
                temp1->ssn,temp1->name,temp1->dept,temp1->des, temp1->salary,temp1->ph);
                temp1=temp1->next;
                count++;
        }
        printf("Number of nodes=%d\n", count);
}

void inbeg(char ssn[15],char name[20],char dept[5],char des[10],int salary, long int ph)
{
        temp1=(struct node *)malloc(1*sizeof(struct node));
        strcpy(temp1->ssn, ssn);
        strcpy(temp1->name, name);
        strcpy(temp1->dept, dept);
        strcpy(temp1->des, des);
        temp1->salary=salary;
        temp1->ph=ph;
        temp1->next=first;
        first->prev=temp1;
        first=temp1;
        temp1->prev=NULL;
}

void inend(char ssn[15],char name[20],char dept[5],char des[10],int salary, long int ph)
{
        temp1=(struct node *)malloc(1*sizeof(struct node));
        strcpy(temp1->ssn, ssn);
        strcpy(temp1->name, name);
        strcpy(temp1->dept, dept);
        strcpy(temp1->des,des);
        temp1->salary=salary;
        temp1->ph=ph;
        last->next=temp1;
        temp1->prev=last;
        temp1->next=NULL;
        last=temp1;
}

void delbeg()
{
        if(first==NULL)
        {
                printf("List is empty\n");
        }
        else
        {
                temp1=first->next;
                if(temp1!=NULL) //DLL has many nodes
                {
                        temp1->prev=NULL;
                        printf("deleted node is:\n");
                        printf("SSN:%s\nNAME:%s\nDEPT:%s\nDESIGNATION:%s\nSALARY:%d\nPH:%ld\n",
                        first->ssn, first->name, first->dept, first->des, first->salary, first->ph);
                        free(first);
                        first=temp1;
                }
```

```
                else
                {
                        printf("deleted node is:\n");
                        printf("SSN:%s\nNAME:%s\nDEPT:%s\nDESIGNATION:%s\nSALARY:%d\nPH:%ld\n",
                        first->ssn, first->name, first->dept, first->des, first->salary, first->ph);
                        free(first);
                        first=NULL;
                }
        }
}

void delend()
{
        if(first==NULL)
        {
                printf("empty list\n");
        }
        else
        {
                temp1=last;
                if(first==last) //only one node
                {
                        printf("deleted node is:\n");
                        printf("SSN:%s\nNAME:%s\nDEPT:%s\nDESIGNATION:%s\nSALARY:%d\nPH:%ld\n",
                        last->ssn, last->name, last->dept, last->des, last->salary, last->ph);
                        first=last=NULL;
                        free(temp1);
                }
                else
                {
                        printf("deleted node is:\n");
                        printf("SSN:%s\nNAME:%s\nDEPT:%s\nDESIGNATION:%s\nSALARY:%d\nPH:%ld\n",
                        last->ssn, last->name, last->dept, last->des, last->salary, last->ph);
                        last=temp1->prev;
                        last->next=NULL;
                        free(temp1);
                }
        }
}

void main()
{
        int choice;
        char ssn[15],name[20],dept[5],des[10];
        int salary;
        long int ph;
        printf("1.Create\n2.Display\n3.Insert at beginning\n4.Insert at End\n5.Delete at beginning\n 6.Delete at
        end\n7.Exit\n");
        while(1)
        {
                printf("\nEnter your choice\n");
                scanf("%d", &choice);
                switch(choice)
                {
                        case 1: printf("Enter Emp SSN, Name, Department, Designation, salary, phone\n");
                                scanf("%s%s%s%s%d%ld",ssn,name,dept,des,&salary,&ph);
                                create(ssn, name, dept, des, salary, ph);
                                break;
```

```
                case 2: display();
                        break;
                case 3: printf("Enter Emp SSN, Name, Department, Designation, salary, phone\n");
                        scanf("%s%s%s%s%d%ld", ssn, name, dept, des,&salary,&ph);
                        inbeg(ssn, name, dept, des, salary, ph);
                        break;
                case 4: printf("Enter Emp SSN, Name, Department, Designation, salary, phone\n");
                        scanf("%s%s%s%s%d%ld", ssn,name,dept,des,&salary,&ph);
                        inend(ssn, name, dept, des, salary, ph);
                        break;
                case 5: delbeg();
                        break;
                case 6: delend();
                        break;
                case 7: exit(0);
        }
    }
}
```

9. Develop Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes
a. Represent and Evaluate a Polynomial P(x,y,z) = $6x^2y^2z-4yz^5 +3x^3yz+2xy^5z-2xyz^3$
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

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
struct node
{
        int coef;
        int pow_x;
        int pow_y;
        int pow_z;
 struct node* next;
};
typedef struct node* POLYPTR;

POLYPTR InsertTerm(POLYPTR poly, int coef, int pow_x, int pow_y, int pow_z)
{
        POLYPTR cur;
        POLYPTR newNode = (POLYPTR)malloc(sizeof(struct node));
        newNode->coef = coef;
        newNode->pow_x = pow_x;
        newNode->pow_y = pow_y;
        newNode->pow_z = pow_z;
        newNode->next = NULL;
        cur = poly;
        while(cur->next != poly)
        {
                cur = cur->next;
        }
        cur->next = newNode;
        newNode->next = poly;
        return poly;
}

void Display(POLYPTR poly)
{
        if (poly->next == poly)
        {
                printf("Polynomial is empty.n");
                return;
        }
        POLYPTR cur = poly->next;
        do
        {
                printf("%dx^%dy^%dz^%d ", cur->coef, cur->pow_x, cur->pow_y, cur->pow_z);
                cur = cur->next;
                if (cur != poly)
                {
                        printf("+ ");
                }
        } while (cur != poly);
 }
```

```
int Evaluate(POLYPTR poly, int x, int y, int z)
{
	int result = 0;
	 if (poly->next == poly)
	{
		return result;
	}
	POLYPTR cur = poly->next;
	do
	{
		int Val = cur->coef*pow(x, cur->pow_x)* pow(y, cur->pow_y)* pow(z, cur->pow_z);
		result =result+ Val;
		cur = cur->next;
	} while (cur != poly);
	return result;
}

bool MatchTerm(POLYPTR p1, POLYPTR p2)
{
	bool bMatches = true;
	if(p1->pow_x != p2->pow_x)
	bMatches = false;
	if(p1->pow_y != p2->pow_y)
	bMatches = false;
	if(p1->pow_z != p2->pow_z)
	bMatches = false;
	return bMatches;
}

POLYPTR AddPoly(POLYPTR poly1, POLYPTR poly2, POLYPTR polySum)
{
	 POLYPTR cur1 = poly1->next;
	POLYPTR cur2 = poly2->next;
	do
	{
		polySum = InsertTerm(polySum, cur1->coef, cur1->pow_x, cur1->pow_y, cur1->pow_z);
		cur1 = cur1->next;
	}while(cur1 != poly1);
	do
	{
		cur1 = polySum->next;
		bool MatchFound = false;
		do
		{
			if(MatchTerm(cur1, cur2))
			{
				cur1->coef += cur2->coef;
				MatchFound = true;
				break;
			}
			cur1 = cur1->next;
		}while(cur1 != polySum);
		if(!MatchFound)
		{
			polySum = InsertTerm(polySum, cur2->coef, cur2->pow_x, cur2->pow_y, cur2->pow_z);
		}
		cur2 = cur2->next;
	}while(cur2 != poly2);
```

```
        return polySum;
}

int main()
{
        POLYPTR poly1 = (POLYPTR)malloc(sizeof(struct node));
        poly1->next = poly1;

        POLYPTR poly2 = (POLYPTR)malloc(sizeof(struct node));
        poly2->next = poly2;

        POLYPTR polySum = (POLYPTR)malloc(sizeof(struct node));

        polySum->next = polySum;

        poly1 = InsertTerm(poly1, 6, 2, 2, 1);
        poly1 = InsertTerm(poly1, -4, 0, 1, 5);
        poly1 = InsertTerm(poly1, 3, 3, 1, 1);
        poly1 = InsertTerm(poly1, 2, 1, 5, 1);
        poly1 = InsertTerm(poly1, -2, 1, 1, 3);
        printf("POLY1(x, y, z) = ");
        Display(poly1);

        poly2 = InsertTerm(poly2, 1, 1, 1, 1);
        poly2 = InsertTerm(poly2, 4, 3, 1, 1);
        printf("\nPOLY2(x, y, z) = ");
        Display(poly2);

        polySum = AddPoly(poly1, poly2, polySum);
        printf("\nPOLYSUM(x, y, z) = ");
        Display(polySum);

        int x = 1, y = 1, z = 1;
        int iRes = Evaluate(polySum, x, y, z);
        printf("\nResult of POLYSUM= %d\n", iRes);
        return 0;
}
```