Program

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

struct node
{
    int coeff,exp;
    struct node *link;
}*A=NULL,*B=NULL,*S=NULL,*P=NULL;


struct node * create()
{
    struct node *TP=NULL,*temp=NULL,*LTP=NULL;
    int n,coeff,exp,i;
    printf("\nEnter no of Terms of Polynomial :: ");
    scanf("%d",&n);
    printf("Enter Coefficint and Exponent of Poly in order(k1x^n+k2x^n-1..+knx^0) \n");
    for(i=0;i<n;i++)
    {
        scanf("%d%d",&coeff,&exp);
        temp=(struct node*)malloc(sizeof(struct node));
        temp->coeff=coeff;
        temp->exp=exp;
        temp->link=NULL;
        if(TP==NULL)
        {
            TP=temp;
            LTP=temp;
        }
        else
        {
            LTP->link=temp;
            LTP=temp;
        }
    }
    return TP;
}


void display(struct node*temp)
{
    while(temp!=NULL)
        {
            printf("%dx^%d ---> ",temp->coeff,temp->exp);
            temp=temp->link;
        }
    printf("NULL\n");
}


struct node * polyadd(struct node*P1, struct node *P2)
{
    struct node *P3=NULL,*lastP3=NULL,*temp=NULL;
    if(P1==NULL)
        P3=P2;
    else if(P2==NULL)
        P3=P1;
    else
    {
    while(P1!=NULL && P2!= NULL)
```

Experiment No: 17
Date:

POLYNOMIAL MULTIPLICATION USING LINKED LIST

Aim:

To read two polynomials and display their product using a linked list.

Algorithm:

1. Start
2. Define Struct Node and Initialise variables

```
int coeff
int exp
node *link
```

3. Initialize pointers A, B, S, P as NULL
4. Create a function to create two polynomials

```
Function create
    Initialize TP, temp, LTP as NULL
    Input n  // Number of terms in the polynomial
    For i = 0 to n-1
        Input coeff, exp
        Allocate memory for temp node
        temp.coeff = coeff
        temp.exp = exp
        temp.link = NULL

        If TP == NULL then
            TP = temp
            LTP = temp
        Else
            LTP.link = temp
            LTP = temp
        End if
    End for loop
    Return TP  // Return the head of the polynomial
End function
```

```
        {
            if(P2->exp > P1->exp)
            {
                temp=(struct node*)malloc(sizeof(struct node));
                temp->coeff=P2->coeff;
                temp->exp=P2->exp;
                temp->link=NULL;

                if(P3==NULL)
                {
                    P3=temp;
                    lastP3=temp;
                }
                else
                {
                    lastP3->link=temp;
                    lastP3=temp;
                }
                P2=P2->link;
            }
            else if(P2->exp < P1->exp)
            {
                temp=(struct node*)malloc(sizeof(struct node));
                temp->coeff=P1->coeff;
                temp->exp=P1->exp;
                temp->link=NULL;
                if(P3==NULL)
                {
                    P3=temp;
                    lastP3=temp;
                }
                else
                {
                    lastP3->link=temp;
                    lastP3=temp;
                }
                P1=P1->link;
            }
            else
            {
                temp=(struct node*)malloc(sizeof(struct node));
                temp->coeff=P1->coeff+P2->coeff;
                temp->exp=P1->exp;
                temp->link=NULL;
                if(P3==NULL)
                {
                    P3=temp;
                    lastP3=temp;
                }
                else
                {
                    lastP3->link=temp;
                    lastP3=temp;
                }
                P1=P1->link;
                P2=P2->link;
            }
        }
    while(P1!=NULL)
        {
            lastP3->link=P1;
            lastP3=P1;
            P1=P1->link;
```

5. Create a function to display polynomials

```
Function display(temp)
    While temp != NULL
        Print temp.coeff, "x^", temp.exp, "---> "
        temp = temp.link
    End While
    Print "NULL"
End Function
```

6. Create a function to add polynomials

```
Function polyadd(P1, P2)
    Initialize P3, lastP3, temp as NULL

    If P1 == NULL then
        Return P2
    Else If P2 == NULL then
        Return P1
    End If

    While P1 != NULL AND P2 != NULL
        If P2.exp > P1.exp then
            Allocate memory for temp
            temp.coeff = P2.coeff
            temp.exp = P2.exp
            temp.link = NULL

            If P3 == NULL then
                P3 = temp
                lastP3 = temp
            Else
                lastP3.link = temp
                lastP3 = temp
            End If
            P2 = P2.link
        Else If P2.exp < P1.exp then
            Allocate memory for temp
            temp.coeff = P1.coeff
            temp.exp = P1.exp
            temp.link = NULL

            If P3 == NULL then
                P3 = temp
                lastP3 = temp
            Else
                lastP3.link = temp
                lastP3 = temp
            End If
            P1 = P1.link
        Else  // Exponents are equal
            Allocate memory for temp
            temp.coeff = P1.coeff + P2.coeff
            temp.exp = P1.exp
            temp.link = NULL

            If P3 == NULL then
                P3 = temp
                lastP3 = temp
            Else
                lastP3.link = temp
                lastP3 = temp
            End If
        }
    while(P2!=NULL)
        {
            lastP3->link=P2;
            lastP3=P2;
            P2=P2->link;
        }
    }
    return P3;
}

struct node *polymult(struct node *P1,struct node *P2)
{
    struct node *CTEMP,*LCTEMP,*P=NULL,*temp;
    while(P1!=NULL)
    {
        P2=B;
        CTEMP=NULL;
        while(P2!=NULL)
        {
            temp=(struct node*)malloc(sizeof(struct node));
            temp->link=NULL;
            temp->coeff=P1->coeff*P2->coeff;
            temp->exp=P1->exp+P2->exp;
            if(CTEMP==NULL)
            {
                CTEMP=temp;
                LCTEMP=temp;
            }
            else
            {
                LCTEMP->link=temp;
                LCTEMP=temp;
            }
            P2=P2->link;
        }
        P=polyadd(P,CTEMP);
        P1=P1->link;
    }
    return P;
}
void main()
{
    int ch;
    do
    {
        printf(" 1. Create Poly \n 2. Multiply Poly \n 3. Display Poly \n 4. Exit  ");
        printf("\nEnter Option :: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                    A=create();
                    B=create();
                    printf("\n");
                    break;
            case 2:
                P=polymult(A,B);
                printf("\n    Polinomial Product : ");
                display(P);
                printf("\n");
                break;
            case 3:
                P1 = P1.link
                P2 = P2.link
            End If
        End While

        While P1 != NULL
            lastP3.link = P1
            lastP3 = P1
            P1 = P1.link
        End While

        WHILE P2 != NULL
            lastP3.link = P2
            lastP3 = P2
            P2 = P2.link
        End While

        Return P3
    End Function
```

7. Create a function to multiply two polynomials

```
    Function polymult(P1, P2)
        Initialize CTEMP, LCTEMP, P as NULL
        While P1 != NULL
            Set P2 = B  // Reset P2 to start of polynomial B
            Set CTEMP = NULL

            While P2 != NULL
                Allocate memory for temp
                temp.coeff = P1.coeff * P2.coeff
                temp.exp = P1.exp + P2.exp
                temp.link = NULL

                If CTEMP == NULL then
                    CTEMP = temp
                    LCTEMP = temp
                Else
                    LCTEMP.link = temp
                    LCTEMP = temp
                End If
                P2 = P2.link
            End While

            P = polyadd(P, CTEMP)
            P1 = P1.link
        End While
        Return P
    End Function
```

8. In Main Function

```
    Do
        Display options and accept input from user
        Input choice
        Switch choice
            Case 1:
                call Create() function
            Case 2:
                Call polymult() function
                printf("\n    Polynomial A : ");
                display(A);
                printf("\n    Polynomial B : ");
                display(B);
                printf("\n");
                break;
            case 4:
                exit(0);
            default :
                printf("\n    Invalid Option !!!");
        }
    }
    while(1);
}
```

Output

```
 1. Create Poly
 2. Multiply Poly
 3. Display Poly
 4. Exit
Enter Option :: 1

Enter no of Terms of Polynomial :: 3
Enter Coefficint and Exponent of Poly in order(k1x^n+k2x^n-1..+knx^0)
2 7
3 4
1 1

Enter no of Terms of Polynomial :: 2
Enter Coefficint and Exponent of Poly in order(k1x^n+k2x^n-1..+knx^0)
5 5
4 4

 1. Create Poly
 2. Multiply Poly
 3. Display Poly
 4. Exit
Enter Option :: 3

    Polynomial A : 2x^7 ---> 3x^4 ---> 1x^1 ---> NULL

    Polynomial B : 5x^5 ---> 4x^4 ---> NULL

 1. Create Poly
 2. Multiply Poly
 3. Display Poly
 4. Exit
Enter Option :: 2

    Polinomial Product : 10x^12 ---> 8x^11 ---> 15x^9 ---> 12x^8 ---> 5x^6 ---> 4x^5 ---> NULL

 1. Create Poly
 2. Multiply Poly
 3. Display Poly
 4. Exit
Enter Option :: 4
```

```
            Print "Polynomial Product:"
            display(P)
        Case 3:
            Print "Polynomial A:"
            display(A)
            Print "Polynomial B:"
            display(B)
        Case 4:
            Exit Loop
        Default:
            Print "Invalid Option"
    End Switch
While True
```

9. Stop

Result:

Program has been executed successfully and obtained the output.