}

3.9 Polynomial arithmetic with linked list

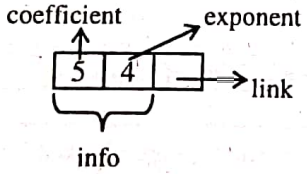
A useful application of linked list is the representation of polynomial expressions. Let us take a polynomial expression with single variable-

$5x^4 + x^3 - 6x + 2$

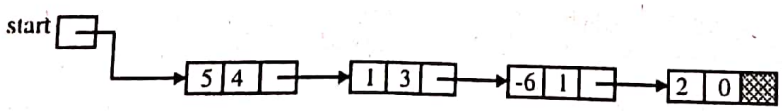
In each term we have a coefficient and an exponent. For example in the term $5x^4$, coefficient is 5 and exponent is 4. The whole polynomial can be represented through linked list where each node will represent a term of the expression. The structure for each node will be-

```
struct node{
        float coefficient;
        int exponent;
        struct node *link;
    }
```

Here the info part of the node contains coefficient and exponent and the link part is same as before and will be used to point to the next node of the list. The node representing the term $5x^4$ can be represented as-



The polynomial $(5x^4 + x^3 - 6x + 2)$ can be represented through linked list as-



Here 2 is considered as $2x^0$ because $x^0 = 1$.

The arithmetic operations are easier if the terms are arranged in descending order of their exponents. For example it would be better if the polynomial expression $(5x + 6x^3 + x^2 - 9 + 2x^6)$ is stored as $(2x^6 + 6x^3 + x^2 + 5x - 9)$. So for representing the polynomial expression, we will use sorted linked list which would be in

descending order based on the exponent. An empty list will represent zero polynomial. The following program shows creation of polynomial linked lists and their addition and multiplication.

```
/*P3.10 Program of polynomial addition and multiplication using linked list*/
#include<stdio.h>
#include<stdlib.h>
struct node
{       float coef;
        int expo;
        struct node *link;
};
struct node *create(struct node *);
struct node *insert_s(struct node *,float,int);
struct node *insert(struct node *,float,int);
void display(struct node *ptr);
void poly_add(struct node *,struct node *);
void poly_mult(struct node *,struct node *);

main()
{       struct node *start1 = NULL,*start2 = NULL;
        printf("Enter polynomial 1 :\n");   start1 = create(start1);
        printf("Enter polynomial 2 :\n");   start2 = create(start2);
        printf("Polynomial 1 is :  "); display(start1);
        printf("Polynomial 2 is :  "); display(start2);
        poly_add(start1, start2);
        poly_mult(start1, start2);
}/*End of main()*/

struct node *create(struct node *start)
{       int i,n,ex;
        float co;
        printf("Enter the number of terms : ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("Enter coeficient for term %d : ",i);
                scanf("%f",&co);
                printf("Enter exponent for term %d : ",i);
                scanf("%d",&ex);
                start = insert_s(start,co,ex);
        }
        return start;
}/*End of create()*/

struct node *insert_s(struct node *start,float co,int ex)
{
        struct node *ptr,*tmp;
        tmp = (struct node *)malloc(sizeof(struct node));
        tmp->coef = co;
        tmp->expo = ex;
        /*list empty or exp greater than first one*/
        if(start == NULL || ex > start->expo)
        {
                tmp->link = start;
                start = tmp;
        }
        else
        {
                ptr = start;
                while(ptr->link!=NULL && ptr->link->expo >= ex)
                        ptr = ptr->link;
```

```
                tmp->link = ptr->link;
                ptr->link = tmp;
        }
        return start;
}/*End of insert()*/

struct node *insert(struct node *start,float co,int ex)
{
        struct node *ptr,*tmp;
        tmp = (struct node *)malloc(sizeof(struct node));
        tmp->coef = co;
        tmp->expo = ex;
        if(start == NULL)       /*If list is empty*/
        {
                tmp->link = start;
                start = tmp;
        }
        else    /*Insert at the end of the list*/
        {
                ptr = start;
                while(ptr->link!=NULL)
                        ptr = ptr->link;
                tmp->link = ptr->link;
                ptr->link = tmp;
        }
        return start;
}/*End of insert()*/

void display(struct node *ptr)
{
        if(ptr == NULL)
        {
                printf("Zero polynomial\n");
                return;
        }
        while(ptr!=NULL)
        {
                printf("(%.1fx^%d)", ptr->coef,ptr->expo);
                ptr = ptr->link;
                if(ptr!=NULL)
                        printf(" + ");
                else
                        printf("\n");
        }
}/*End of display()*/

void poly_add(struct node *p1,struct node *p2)
{
        struct node *start3;
        start3 = NULL;
        while(p1!=NULL && p2!=NULL)
        {
                if(p1->expo > p2->expo)
                {
                        start3 = insert(start3,p1->coef,p1->expo);
                        p1 = p1->link;
                }
                else if(p2->expo > p1->expo)
                {
                        start3 = insert(start3,p2->coef,p2->expo);
                        p2 = p2->link;
                }
                else if(p1->expo == p2->expo)
                {
                        start3 = insert(start3,p1->coef+p2->coef,p1->expo);
```

```
                p1 = p1->link;
                p2 = p2->link;
            )
        )
        /*if poly2 has finished and elements left in poly1*/
        while(p1!=NULL)
        {
            start3 = insert(start3,p1->coef,p1->expo);
            p1 = p1->link;
        }
        /*if poly1 has finished and elements left in poly2*/
        while(p2!=NULL)
        {
            start3 = insert(start3,p2->coef,p2->expo);
            p2 = p2->link;
        }
        printf("Added polynomial is : ");
        display(start3);
}/*End of poly_add() */

void poly_mult(struct node *p1, struct node *p2)
{
        struct node *start3;
        struct node *p2_beg = p2;

        start3 = NULL;
        if(p1 == NULL || p2 == NULL)
        {
            printf("Multiplied polynomial is zero polynomial\n");
            return;
        }
        while(p1!=NULL)
        {
           p2 = p2_beg;
           while(p2!=NULL)
           {
             start3 = insert_s(start3,p1->coef*p2->coef,p1->expo+p2->expo);
             p2 = p2->link;
           }
           p1 = p1->link;
        }
        printf("Multiplied polynomial is : ");
        display(start3);
}/*End of poly_mult()*/
```

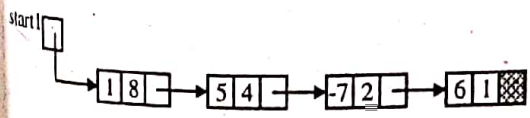
3.9.1 Creation of polynomial linked list

The function `create()` is very simple and calls another function `insert_s()` that inserts a node in polynomial linked list. The function `insert_s()` is similar to that of sorted linked lists. The only difference is that here our list is in descending order based on the exponent.

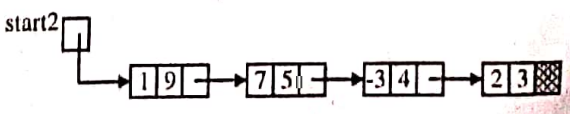
3.9.2 Addition of 2 polynomials

The procedure for addition of 2 polynomials represented by linked lists is somewhat similar to that of merging. Let us take two polynomial expression lists and make a third list by adding them.

Poly1- $x^8 + 5x^4 - 7x^2 + 6x$



Poly2- $x^9 + 7x^5 - 3x^4 + 2x^3$



The pointers p1 and p2 will point to the current nodes in the polynomials which will be added. The process of addition is shown in the figure 3.46. Both polynomials are traversed until one polynomial finishes. We can have three cases-

1. If (p1->expo) > (p2->expo)
The new node that is added to the resultant list has coefficient equal to p1->coef and exponent equal to p1->expo. After this we will make p1 point to the next node of polynomial1.

2. If (p2->expo) > (p1->expo)
The new node that is added to the resultant list has coefficient equal to p2->coef and exponent equal to p2->expo. After this we will make p2 point to the next node of polynomial2.

3. If (p1->expo) == (p2->expo)
The new node that is added to the resultant list has coefficient equal to (p1->coef + p2->coef) and exponent equal to p1->expo(or p2->expo). After this we will make p1 and p2 point to the next nodes of polynomial1 and polynomial 2 respectively. The procedure of polynomial addition is shown in figure 3.46.

```
while(p1!=NULL && p2!=NULL)
{
        if(p1->expo > p2->expo)
        {
                p3_start = insert(p3_start,p1->coef,p1->expo);
                p1 = p1->link;
        }
        else if(p2->expo > p1->expo)
        {
                p3_start = insert(p3_start,p2->coef,p2->expo);
                p2 = p2->link;
        }
        else if(p1->expo == p2->expo)
        {
                p3_start = insert (p3_start,p1->coef+p2->coef,p1->expo);
                p1 = p1->link;
                p2 = p2->link;
        }
}
```

The above loop will terminate when any of the polynomial will finish. Now we have to add the remaining nodes of the unfinished polynomial to the resultant list. If polynomial 2 has finished, then we will put all the terms of polynomial 1 in the resultant list as-

```
while(p1!=NULL)
{
        p3_start = insert(p3_start,p1->coef,p1->expo);
        p1 = p1->link;
}
```

If polynomial 1 has finished, then we will put all the terms of polynomial 2 in the resultant list as-

```
while(p2!=NULL)
{
        p3_start = insert(p3_start,p2->coef,p2->expo);
        p2 = p2->link;
}
```

We can see the advantage of storing the terms in descending order of their exponents. If it was not so then we would have to scan both the lists many times.

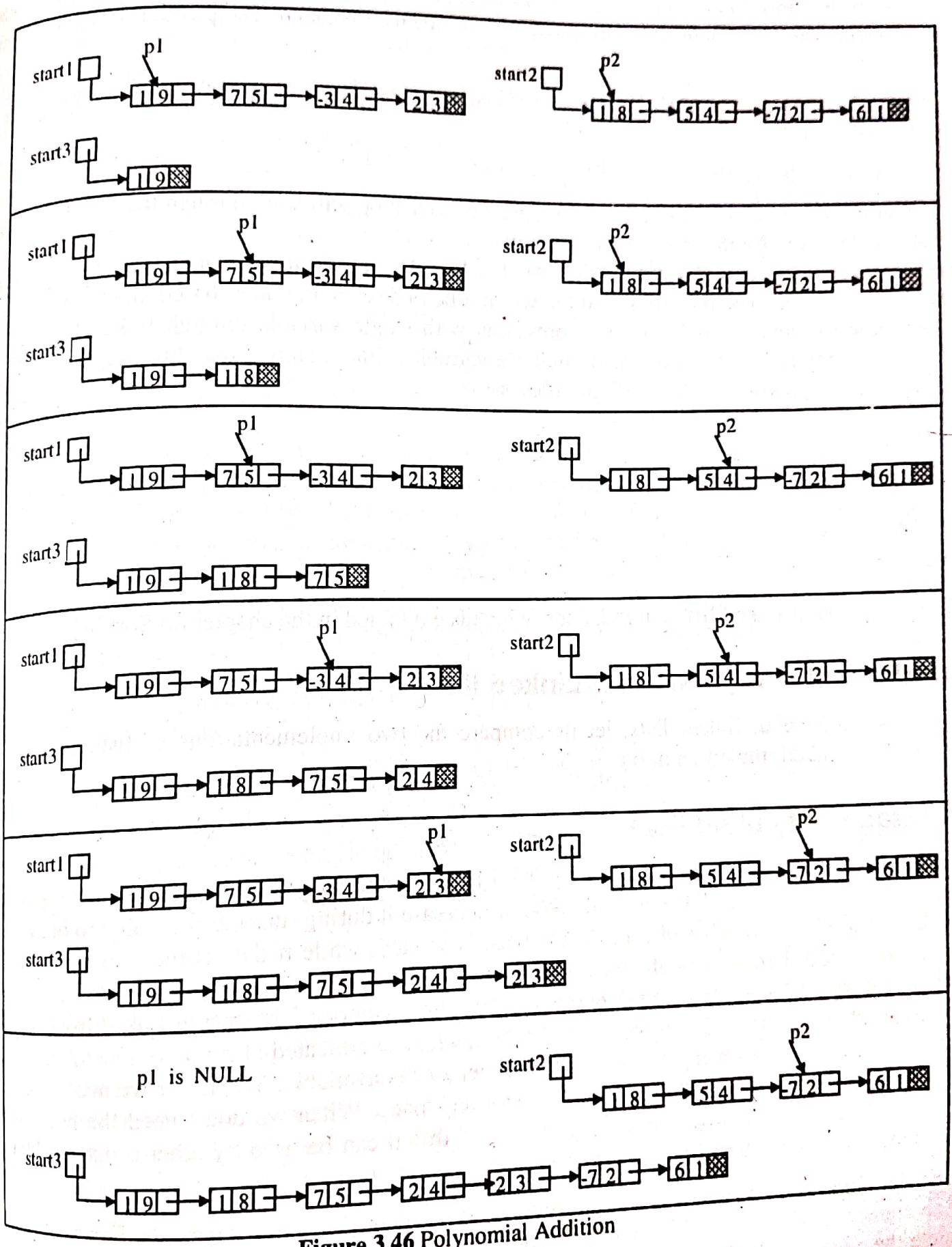


Figure 3.46 Polynomial Addition