UNIT I – INTRODUCTION

Data structures - Abstract data types - Primitive data structures- – Performance analysis – Space complexity – Time complexity – Asymptotic notations – Performance measurement – Array as an abstract data type – Polynomial as an abstract data type – Sparse matrix abstract data type – String abstract data type.

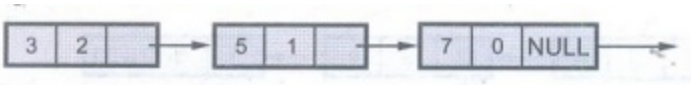
POLYNOMIAL AS AN ABSTRACT DATA TYPE

A polynomial has the main fields as coefficient, exponent. In the linked list it will have one more field called 'link' field to point to next term in the polynomial. If there are n terms in the polynomial then n such nodes have to be created.

The typical node will look like this.

Fig. 1.11.1 Node of polynomial

For example: To represent 3x2+5x+7 the link list will be



In each node, the exponent field will store the exponent corresponding to that term, the coefficient field will store coefficient corresponding to that term and the link field will point to the next term in the polynomial. Again for simplifying the algorithms such as addition of two polynomials we will assume that the polynomial terms are stored in descending order of exponents. The node structure for a singly linked list for representing a term of polynomial can be defined as follows:

```
typedef struct Pnode
{
        float coef;
        int exp;
struct node *next;
} p;
```

Advantages of linked representation over arrays :

1. Only one pointer will be needed to point to first term of the polynomial.
2. No prior estimation on number of terms in the polynomial is required. This results in flexible and more space efficient representation.
3. The insertion and deletion operations can be carried out very easily without movement of data.

Disadvantage of linked representation over arrays :

We can not access any term randomly or directly we have to go from the start node always.

Addition of Two Polynomials using Singly Linked List

Logic for polynomial addition by linked list :

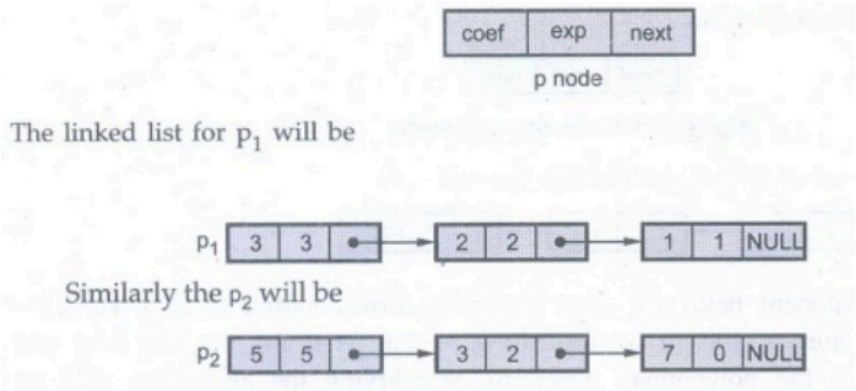
Step 1: First of all we create two linked polynomials.

For e.g.:

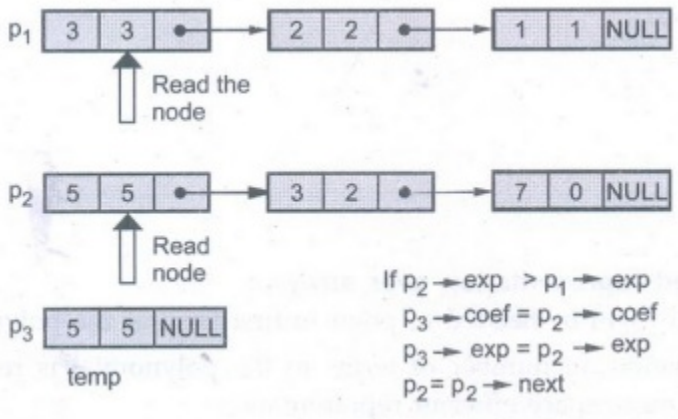
P1 = 3x3 +2x2 + 1x

P2 = 5x5 +3x2 +7

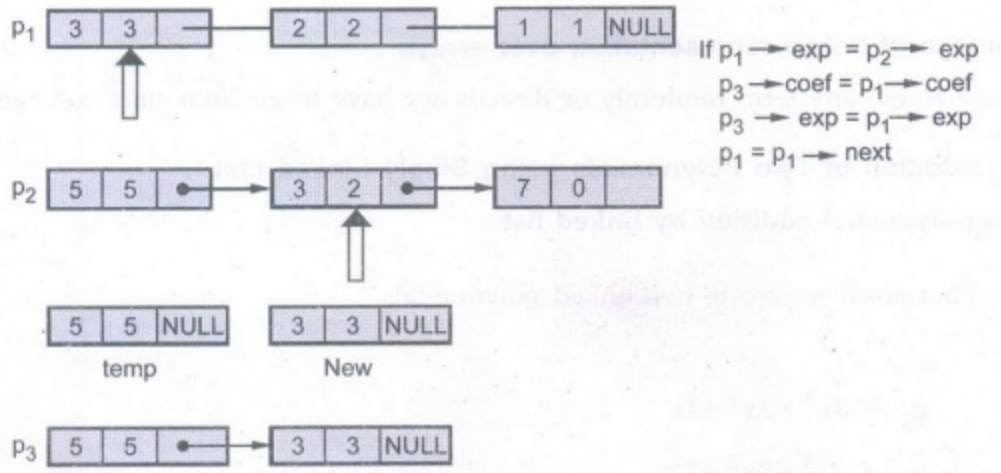
Each node in the polynomial will look like this.



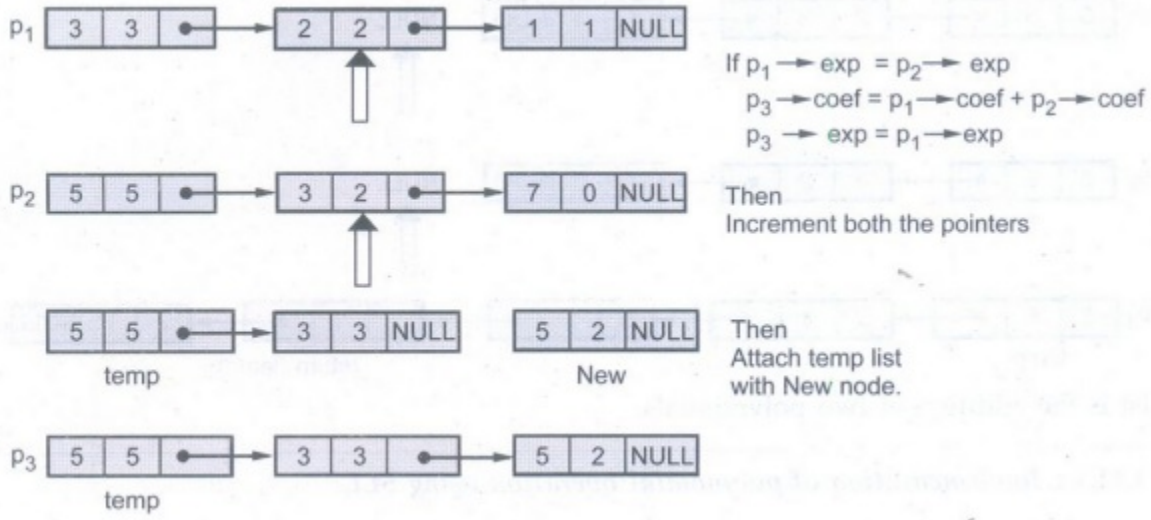
Step 2: For addition of two polynomials if exponents of both the polynomials are same then we add the coefficients. For storing the result we will create the third linked list say p3. The processing will be as follows:



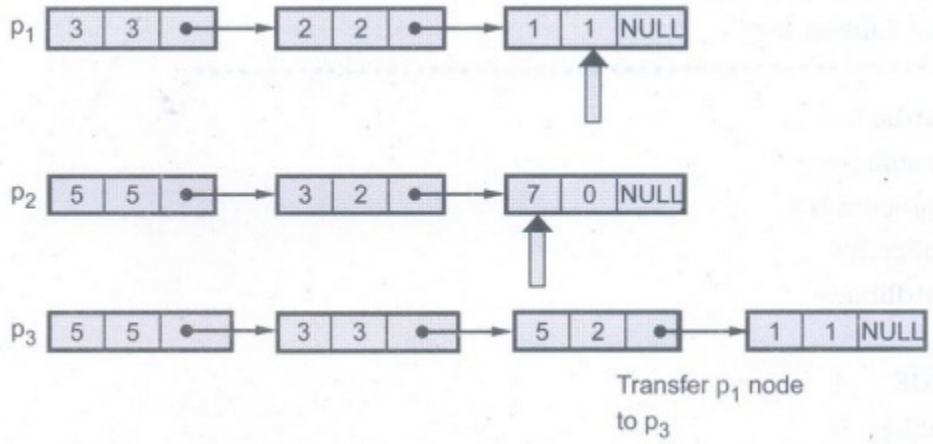
Step 3:



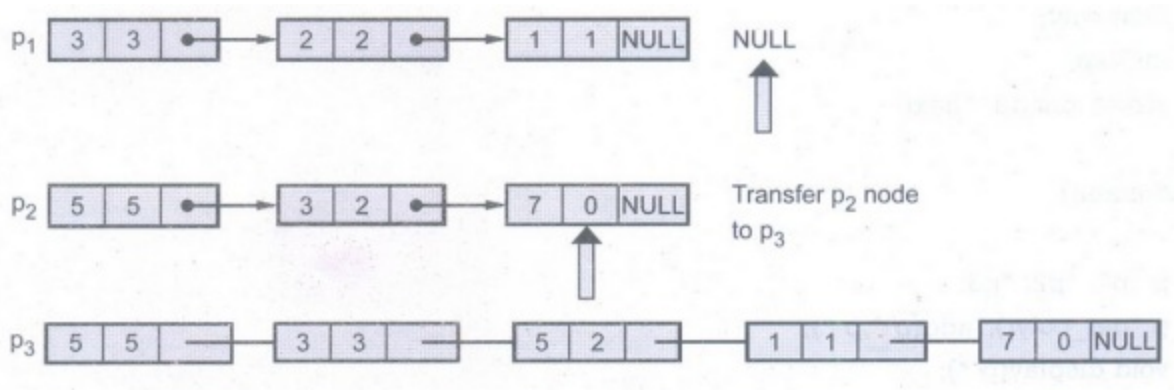
Step 4:



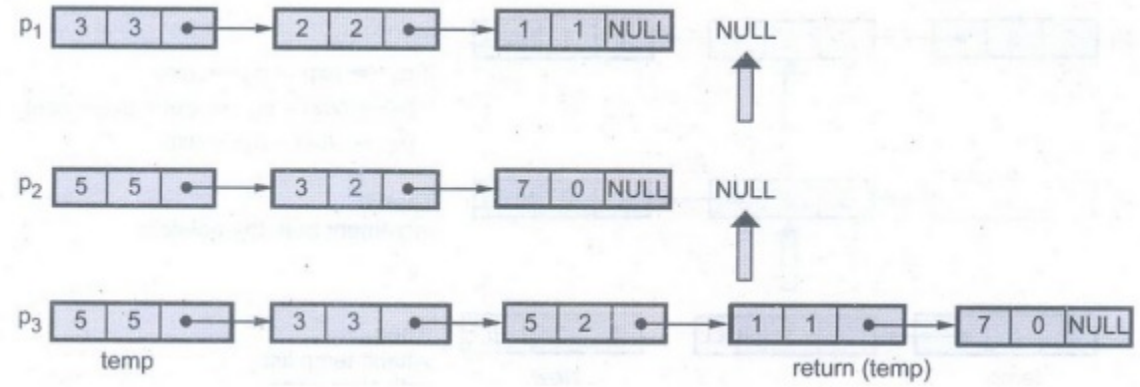
Step 5:



Step 6:



Finally



P3 list is the addition of two polynomials.

Function to create new node

```
void create_node(int x, int y, struct Node** temp)
{
   struct Node *r, *z;
   z = *temp;
   if (z == NULL) {
      r = (struct Node*)malloc(sizeof(struct Node));
      r->coeff = x;
      r->pow = y;
      *temp = r;
      r->next = (struct Node*)malloc(sizeof(struct Node));
      r = r->next;
      r->next = NULL;
   }
   else {
      r->coeff = x;
      r->pow = y;
```

```
        r->next = (struct Node*)malloc(sizeof(struct Node));
        r = r->next;
        r->next = NULL;
    }
}
```

Function Adding two polynomial numbers

```
void polyadd(struct Node* poly1, struct Node* poly2,
             struct Node* poly)
{
    while (poly1->next && poly2->next) {
        // If power of 1st polynomial is greater then 2nd,
        // then store 1st as it is and move its pointer
        if (poly1->pow > poly2->pow) {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }

        // If power of 2nd polynomial is greater then 1st,
        // then store 2nd as it is and move its pointer
        else if (poly1->pow < poly2->pow) {
            poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }

        // If power of both polynomial numbers is same then
        // add their coefficients
        else {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff + poly2->coeff;
            poly1 = poly1->next;
            poly2 = poly2->next;
        }

        // Dynamically create new node
```

```
        poly->next
            = (struct Node*)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
    while (poly1->next || poly2->next) {
        if (poly1->next) {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        if (poly2->next) {
            poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        poly->next
            = (struct Node*)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
}
```
