Case Study: Polynomial ADT

Polynomial Representation using Linked Lists

Data Structures

Polynomial ADT

A single variable polynomial can be generalized as:

$$f(x) = \sum_{i=0}^{n} a_i x^i$$

An example of a single variable polynomial:

$4x^6 + 10x^4 - 5x + 3$

Remark: the order of this polynomial is 6
(look for highest exponent)

Polynomial ADT (continued)

By definition of ADT:

A set of values and a set of allowable operations on those values.

We can now operate on this polynomial the way we like…

Polynomial ADT

- What kinds of operations?

Here are the most common operations on a polynomial:

- Add & Subtract
- Multiply
- Differentiate
- Integrate
- etc…

Polynomial ADT (continued)

- Why implement this?

Calculating polynomial operations by hand can be very cumbersome. Take differentiation as an example:

$$d(23x^{9} + 18x^{7} + 41x^{6} + 163x^{4} + 5x + 3)/dx$$

$$= (23*9)x^{(9-1)} + (18*7)x^{(7-1)} + (41*6)x^{(6-1)} + \ldots$$

Polynomial ADT (continued)

- How to implement this?

There are different ways of implementing the polynomial ADT:

- Array (not recommended)
- Linked List (preferred and recommended)

• Polynomial ADT (continued)

- Array Implementation:
- $p1(x) = 8x^3 + 3x^2 + 2x + 6$
- $p2(x) = 23x^4 + 18x - 3$

p1(x)

6	2	3	8
0		2	

p2(x)

-3	18	0	0	23
0		2		4

Index represents exponents

Polynomial ADT (continued)

- This is why arrays aren't good to represent polynomials:

- $p3(x) = 16x^{21} - 3x^{5} + 2x + 6$

6	2	0	0	-3	0	…………	0	16

<u>WASTE OF SPACE!</u>

Polynomial ADT (continued)

- Advantages of using an Array:

  - only good for non-sparse polynomials.
  - ease of storage and retrieval.

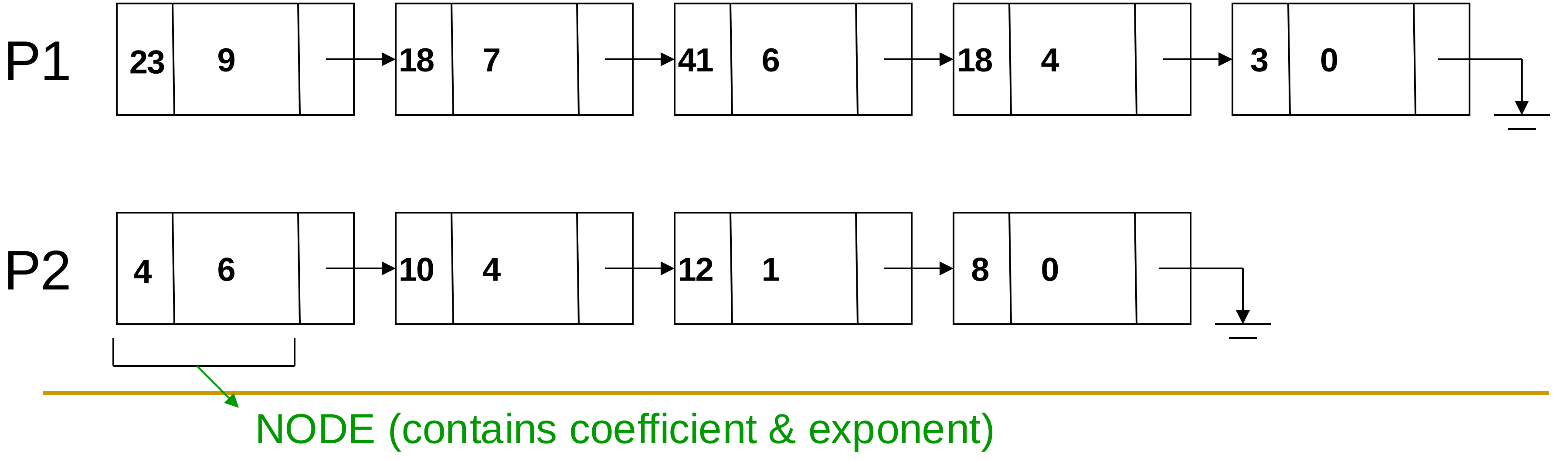
- Disadvantages of using an Array:

  - have to allocate array size ahead of time.
  - huge array size required for sparse polynomials. Waste of space and runtime.

Polynomial ADT (continued)

- Linked list Implementation:

- $p1(x) = 23x^9 + 18x^7 + 41x^6 + 18x^4 + 3$
- $p2(x) = 4x^6 + 10x^4 + 12x + 8$



Polynomial ADT (continued)

- Advantages of using a Linked list:

  - save space (don't have to worry about sparse polynomials) and easy to maintain
  - don't need to allocate list size and can declare nodes (terms) only as needed

- Disadvantages of using a Linked list :
  - can't go backwards through the list
  - can't jump to the beginning of the list from the end.

Polynomial ADT (continued)

- Adding polynomials using a Linked list representation: (storing the result in p3)

To do this, we have to break the process down to cases:

- Case 1: exponent of p1 > exponent of p2
  - Copy node of p1 to end of p3.

[go to next node]

- Case 2: exponent of p1 < exponent of p2
  - Copy node of p2 to end of p3.

[go to next node]

Polynomial ADT (continued)

- Case 3: exponent of p1 = exponent of p2
  - Create a new node in p3 with the same exponent and with the sum of the coefficients of p1 and p2.

Polynomials

$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \ldots + a_0 x^{e_0}$$

Representation

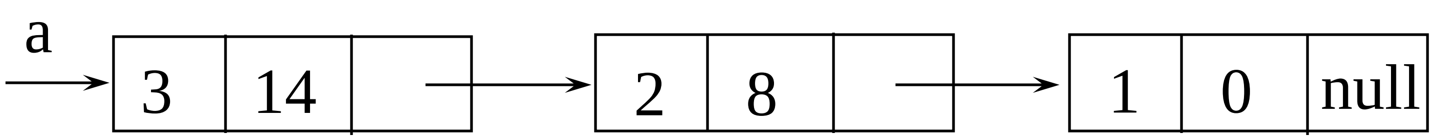
```
struct polynode {
      int coef;
      int exp;
      struct polynode * next;
  };
```

```
typedef struct polynode polynode;
polynode *a, *b;
```

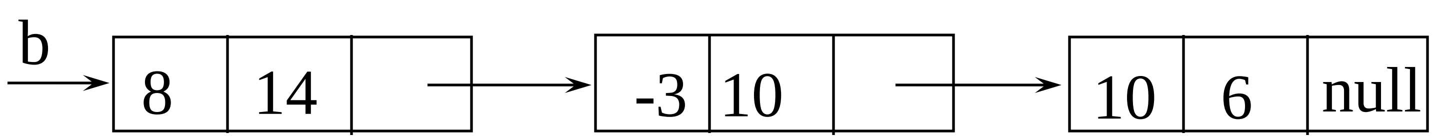
coef	exp	next

Example

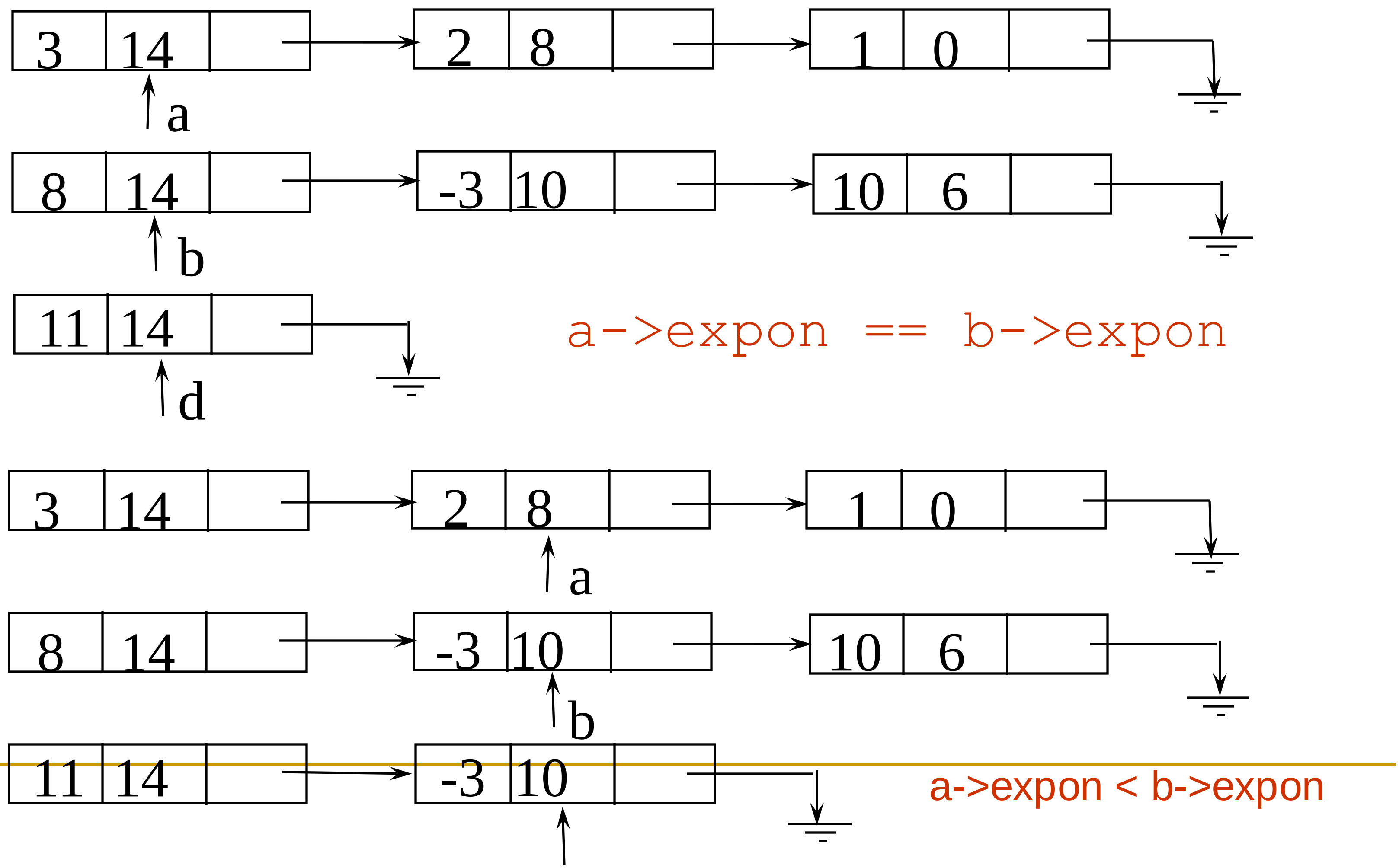
$$a = 3x^{14} + 2x^{8} + 1$$

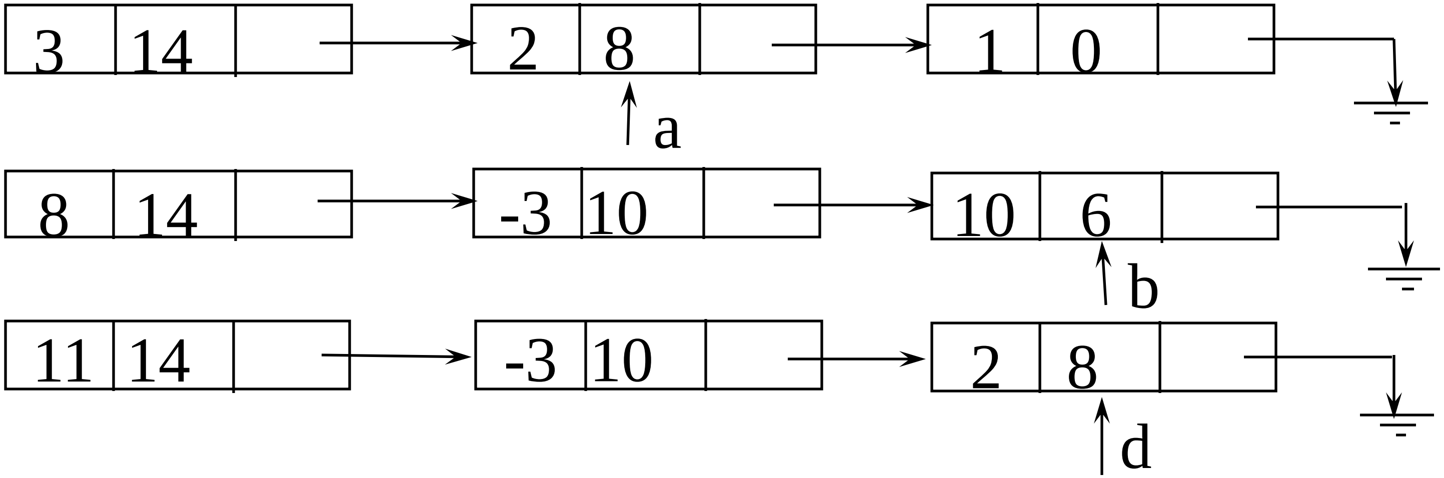


$$b = 8x^{14} - 3x^{10} + 10x^{6}$$



Adding Polynomials





`a->expon > b->expon`

C Program to implement polynomial Addition

```
struct polynode
{
   int coef;
   int pow;
   struct polynode *next;
};
typedef struct polynode poly;
```

```
void main()
{
    poly *poly1=NULL, *poly2=NULL, *poly3=NULL;
    int x, y;
    char choice;
    printf("Enter 1st Polynomial\n");
    do{
        printf("Enter Coefficient\n");
        scanf("%d", &x);
        printf("Enter Exponent\n");
        scanf("%d", &y);
        insert_node(x,y,&poly1);
        printf("Do you want to enter more terms: Y/N\n");
        scanf("%c", &choice);
    }while(choice != 'N');
```

```
printf("Enter 2nd Polynomial\n");
do{
    printf("Enter Coefficient\n");
    scanf("%d",&x);
    printf("Enter Exponent\n");
    scanf("%d",&y);
    insert_node(x, y, &poly2);
    printf("Do you want to enter more terms: Y/N\n");
    scanf("%c",&choice);
    }while(choice != 'N');
```

```
    printf("Entered Polynomials are:\n);
    display(poly1);
    printf("\n");
    display(poly2);
    poly3=(poly *)malloc(sizeof(poly));
    add_polynomials(poly1, poly2, poly3);
    printf(Resultant Polynomial after addition :\n");
    display(poly3);
}
```

```
void display(poly *a)
{
    while(a != NULL)
    {
        printf(“%dx^%d”, a->coef, a->pow);
        if(a->next != NULL)
            printf(“ + ”);
        a = a->next;
    }
}
```

```
void insert_node(int x, int y, poly **temp)
{
        poly *r, *z;
        z = *temp;
        r =(poly *)malloc(sizeof(poly));
        r->coeff = x;
        r->pow = y;
        r->next = NULL;
        if(z == NULL)
                *temp = r;
        else
        {
                while(z->next != NULL)
                        z = z->next;
                z->next = r;
        }
}
```

```
void add_polynomials(poly *a, poly *b, poly *c)
{
          while(a && b)
          {
      // If power of 1st polynomial is greater then 2nd, then store 1st as it is
      // and move its pointer
                    if(a->pow > b->pow)
                    {
                              c->pow = a->pow;
                              c->coef = a->coef;
                              a = a->next;
                    }
```

```
     // If power of 2nd polynomial is greater then 1st, then store 2nd as it is
     // and move its pointer

    else if(a->pow < b->pow)
     {
        c->pow = b->pow;
        c->coef = b->coef;
        b = b->next;
     }
```

```
        // If power of both polynomial numbers is same then add
their coefficients
        else
        {
            c->pow = a->pow;
            c->coef = a->coef+b->coef;
            a = a->next;
            b = b->next;
        }

        // Dynamically create new node only when terms are still
left in any of the polynomial
            if(a || b)
            {
                        c->next = (poly *)malloc(sizeof(poly));
                        c = c->next;
                        c->next = NULL;
            }
    } //End of while
```

```
while(a || b)
  {
        if(a)
        {
                c->pow = a->pow;
                c->coef = a->coef;
                a = a->next;
        }
        if(b)
        {
                c>pow = b->pow;
                c->coef =b->coef;
                b = b->next;
        }
        if(a || b)
        {       c->next = (poly *)malloc(sizeof(poly));
                c = c->next;
                c->next = NULL; }
  }//End of while
} //End of add_polynomials
```

Complexity?

- What do you think about the time complexity of adding 2 polynomials of size ‘m’ and ‘n’ respectively?

O(m+n), where **m** is the length of 1st polynomial

n is the length of 2nd polynomial

Exercise

- Write a program to implement polynomial multiplication.