Linked List (II):

Polynomials

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline


Outline

Goal

Represent the polynomial:

$$a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \cdots + a_0 x^{e_0}.$$

Goal

Represent the polynomial:

$$a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \cdots + a_0 x^{e_0}.$$

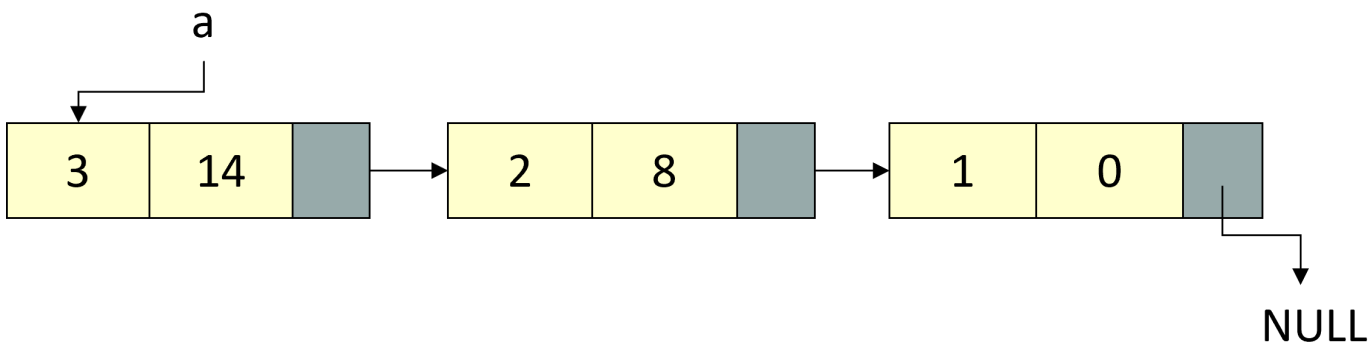
- **Idea:** Represent each term as a node containing
  - coefficient field
  - exponent field
  - pointer to the next term

Declaration

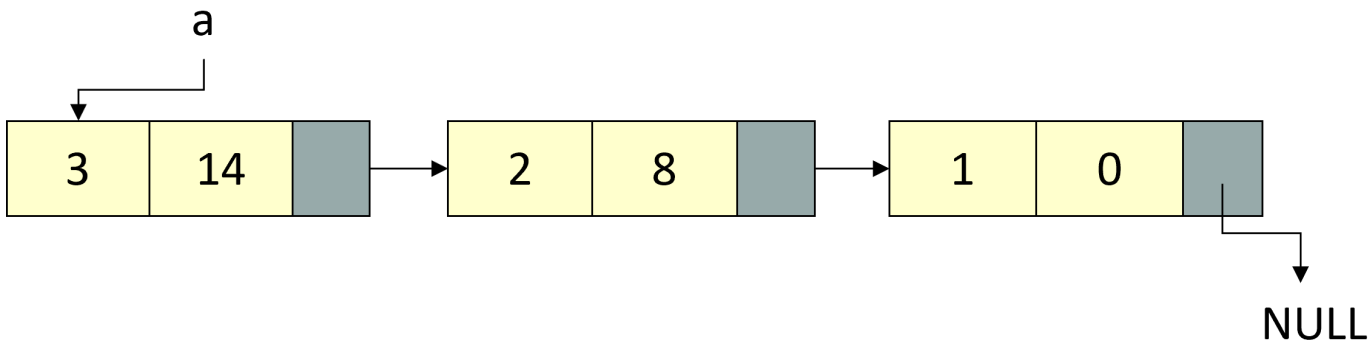
```
typedef struct polyNode *polyPointer;
struct polyNode {
    int coef;
    int expon;
    polyPointer link;
};
polyPointer a, b;
```

Examples

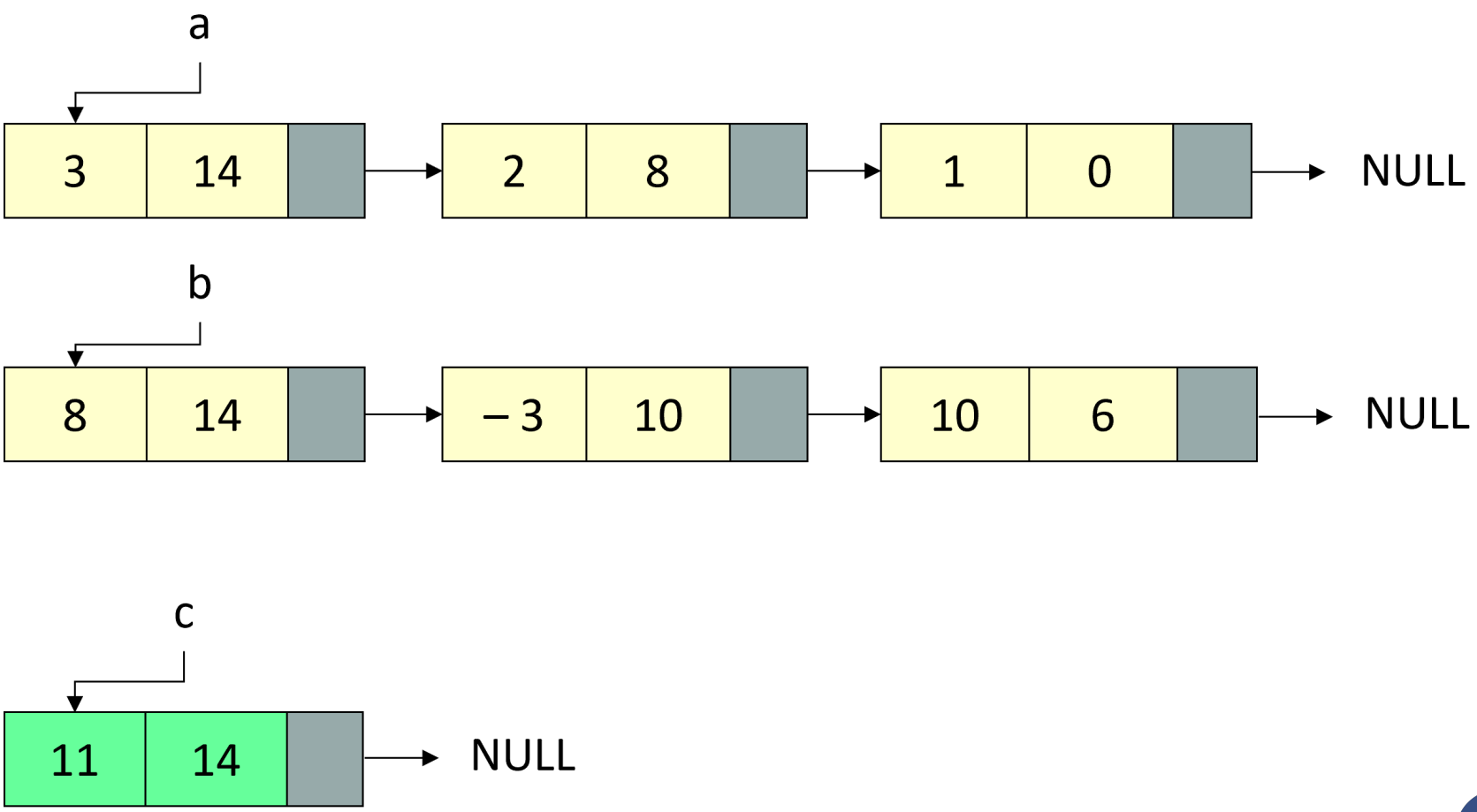
- $a = 3x^{14} + 2x^8 + 1$



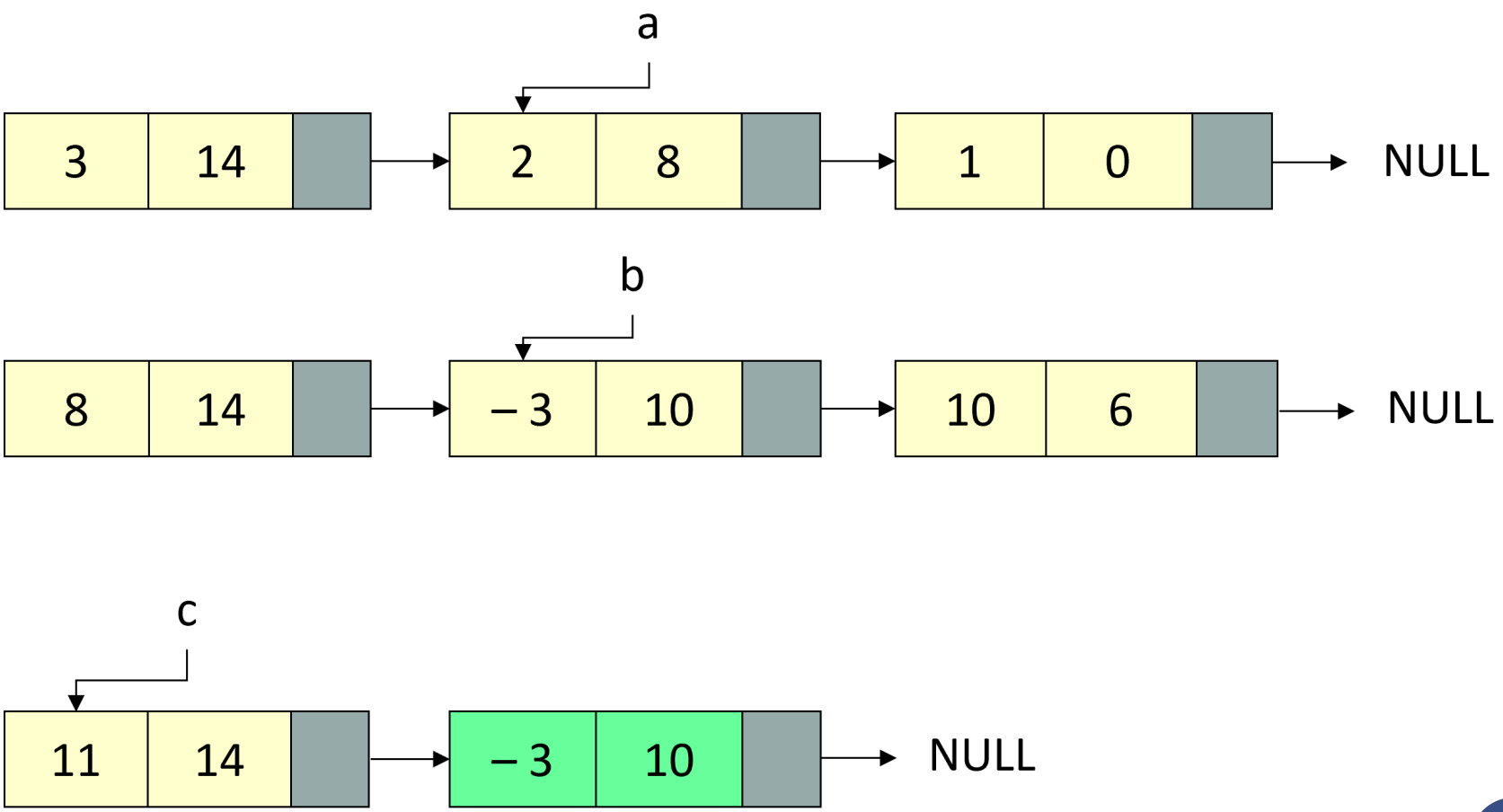
- $b = 8x^{14} - 3x^{10} + 10x^6$



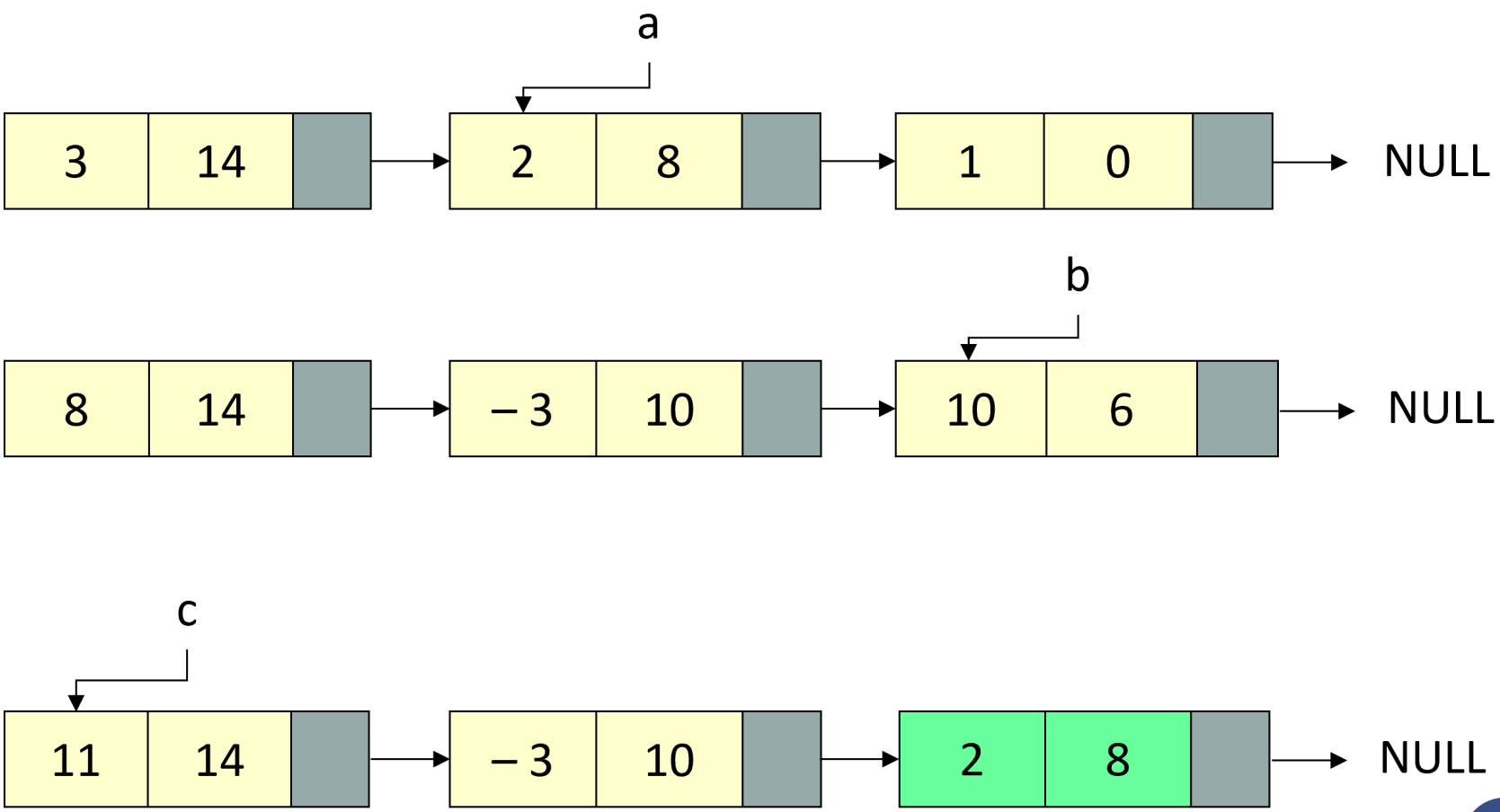
Generating the first three terms of $c = a + b$



Generating the first three terms of $c = a + b$



Generating the first three terms of $c = a + b$



Addition of Two Polynomials

```
polyPointer polyAdd(polyPointer a, polyPointer b) { /* return a polynomial which is the sum of a and b */
    polyPointer front, rear, temp;
    int sum; rear = (polyPointer)malloc(sizeof(*rear));
    if (IS_FULL(rear)) { printf("The memory is full\n"); exit(1) }
    front = rear;
    while (a && b) {
        switch (COMPARE(a->expon, b->expon)) {
            case -1: /* a->expon < b->expon */
                attach(b->coef, b->expon, &rear);
                b = b->link;
                break;
            case 0: /* a->expon = b->expon */
                sum = a->coef + b->coef;
                if (sum)
                    attach(sum, a->expon, &rear);
                a = a->link; b = b->link; break;
            case 1: /* a->expon > b->expon */
                attach(a->coef, a->expon, &rear);
                a = a->link;
        }
        /* copy rest of list a and then list b*/
        for (; a; a = a->link) attach(a->coef, a->expon, &rear);
        for (; b; b = b->link) attach(b->coef, b->expon, &rear);
        rear->link = NULL:
        /* delete extra initial node */
        temp = front; front = front->link; free(temp);
        return front;
    }
}
```

Attach a new node to the end of a list

```
void attach(float coefficient, int exponent, polyPointer *ptr) {
/* create a new node with coef = coefficient and expon = exponent,
attach it to the node pointed by ptr and update ptr to point to this new node */
    polyPointer temp;
    temp = (polyPointer)malloc(sizeof(*temp));
    if (IS_FULL(temp)) {
        printf("The memory is full\n");
        exit(1);
    }
    temp->coef = coefficient;
    temp->expon = exponent;
    (*ptr)->link = temp;
    *ptr = temp;
}
```

Analysis of `polyAdd`

Consider

$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \cdots + a_0 x^{e_0},$$
$$B(x) = b_{m-1}x^{f_{m-1}} + b_{m-2}x^{f_{m-2}} + \cdots + b_0 x^{f_0}$$

Analysis of `polyAdd`

Consider

$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \cdots + a_0 x^{e_0},$$
$$B(x) = b_{m-1}x^{f_{m-1}} + b_{m-2}x^{f_{m-2}} + \cdots + b_0 x^{f_0}$$

- coefficient additions:

$$0 \leq \#\text{coefficient additions} \leq \min\{m, n\}$$

Analysis of `polyAdd`

Consider

$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \cdots + a_0 x^{e_0},$$
$$B(x) = b_{m-1}x^{f_{m-1}} + b_{m-2}x^{f_{m-2}} + \cdots + b_0 x^{f_0}$$

- coefficient additions:

$$0 \leq \#\text{coefficient additions} \leq \min\{m, n\}$$

- exponent comparisons:
  - In each iteration, either pointer a or b or both move to the next term(s).
  - The maximum number of exponent comparisons is $m + n$
- create new nodes for C:

Analysis of `polyAdd`

Consider

$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \cdots + a_0 x^{e_0},$$
$$B(x) = b_{m-1}x^{f_{m-1}} + b_{m-2}x^{f_{m-2}} + \cdots + b_0 x^{f_0}$$

- coefficient additions:

$$0 \leq \#\text{coefficient additions} \leq \min\{m, n\}$$

- exponent comparisons:
  - In each iteration, either pointer a or b or both move to the next term(s).
  - The maximum number of exponent comparisons is $m + n$
- create new nodes for C:
  - The maximum number of terms in C is

Analysis of `polyAdd`

Consider

$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \cdots + a_0 x^{e_0},$$
$$B(x) = b_{m-1}x^{f_{m-1}} + b_{m-2}x^{f_{m-2}} + \cdots + b_0 x^{f_0}$$

- coefficient additions:

$$0 \leq \#\text{coefficient additions} \leq \min\{m, n\}$$

- exponent comparisons:
  - In each iteration, either pointer a or b or both move to the next term(s).
  - The maximum number of exponent comparisons is $m + n$
- create new nodes for C:
  - The maximum number of terms in C is $O(m + n)$.

Outline

Pointers

- C provides extensive supports for pointers.

> &: address operator
>
> *: dereferencing (indirect) operator

```
int i, *pi; // i:integer variable; pi: a pointer to an integer.
pi = &i;    // pi gets the address of i.
i = 10; // assign the value 10 to i
*pi = 20; // assign the value 20 to i
if (pi == NULL) ... // or if  (!pi); test if the pointer is null.
```


Dynamically Allocated Storage

- C provides a mechanism, called heap, for allocating storage at run-time.
  - `malloc` or `calloc`: dynamic memory allocation.
  - `free`: free the memory previously (dynamically) allocated.

```
int i, *pi;
float f, *pf;
pi = (int *) malloc(sizeof(int));
pf = (float *)malloc(sizeof(float));
*pi = 1024; *pf = 3.14;
free(pi);
free(pf);
```


Discussions