Chap 4. Linked Lists (2)

Contents

Chapter 1. Basic Concepts

Chapter 2. Arrays And Structures

Chapter 3. Stacks And Queues

Chapter 4. Linked Lists

Chapter 5. Trees (Midterm exam)

Chapter 6. Graphs

Chapter 7. Sorting

Chapter 8. Hashing (Final exam)

Contents

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

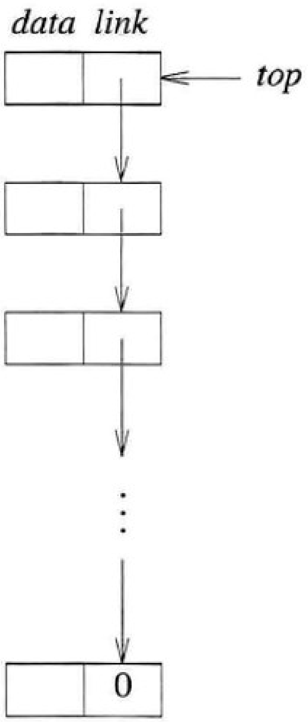
4.5 Additional List Operations

4.6 Equivalence Classes

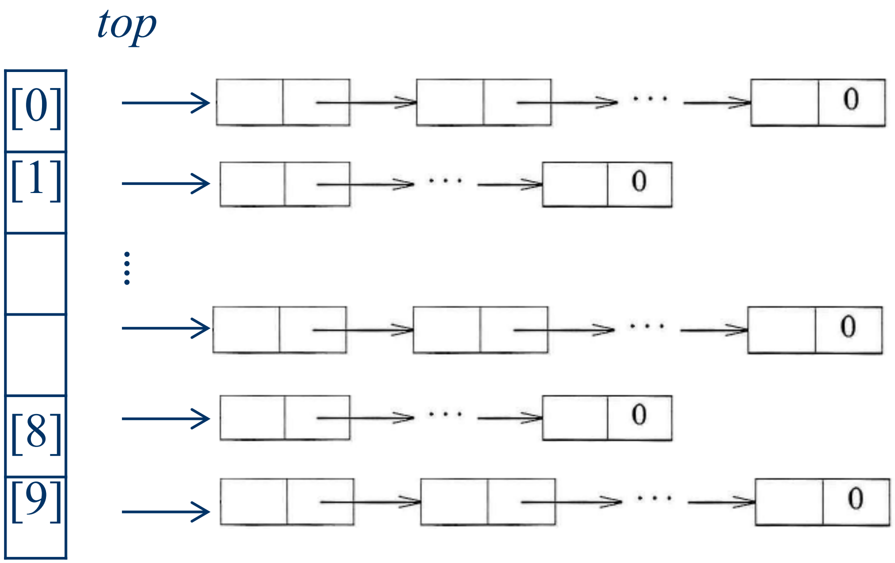
4.7 Sparse Matrices

4.8 Doubly Linked Lists

4.3 Linked Stacks And Queues



(a) Linked stack

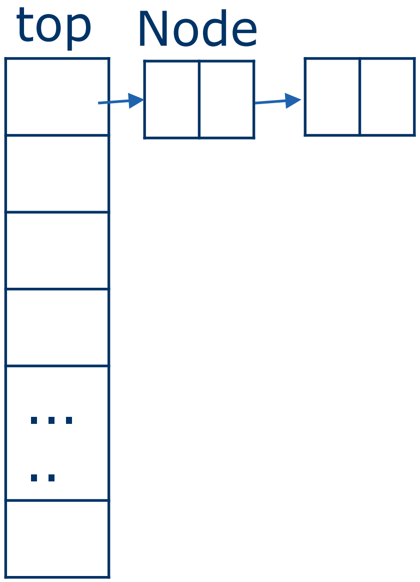


(b) Linked multi stack

4.3 Linked Stacks And Queues

- Representing *n≤MAX_STACKS* stacks simultaneously

```
#define MAX-STACKS 10 /* maximum number of stacks */
typedef struct {
        int key;
        /* other fields */
        } element;
typedef struct stack *stackPointer;
typedef struct stack {
        element data;
        stackPointer link;
        } ;
stackPointer top[MAX-STACKS];
```

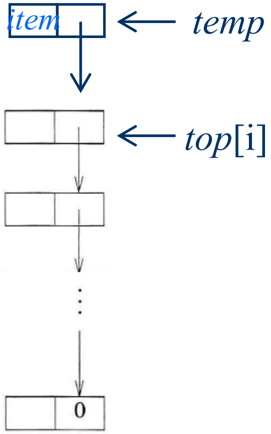


- *top [i] =NULL, 0≤ i <MAX –STACKS* //Initial conditions for the stacks
- *top* [i] ==*NULL* iff the ith stack is empty //Boundary condition for the *i*th stack

4.3 Linked Stacks And Queues

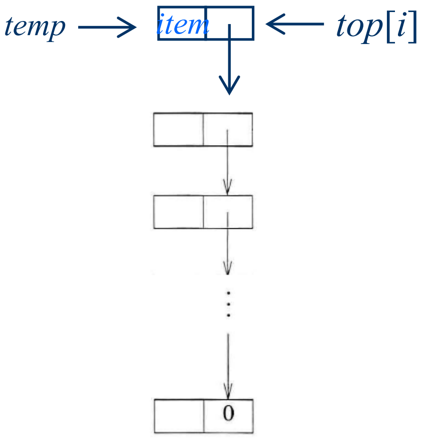
```
void push(int i, element item)
{/* add item to the ith stack */
        stackPointer temp;
        MALLOC(temp, sizeof(*temp) );
        temp->tdata = item;
        temp->tlink = top[i];
        top[i] = temp;
}
```

Program 4.5: Add to a linked stack



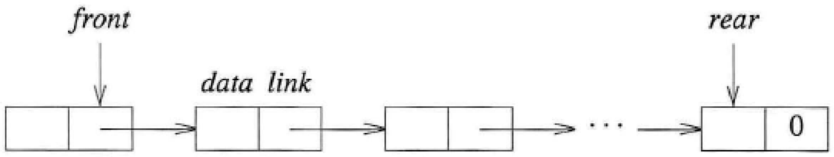
4.3 Linked Stacks And Queues

```
element pop(int i)
{/* remove top element from the ith stack */
        element item;
        stackPointer temp= top[i];
        if (!temp)
                return stackEmpty();
        item = temp->data;
        top[i] = temp->link;
        free (temp);
        return item;
}
```

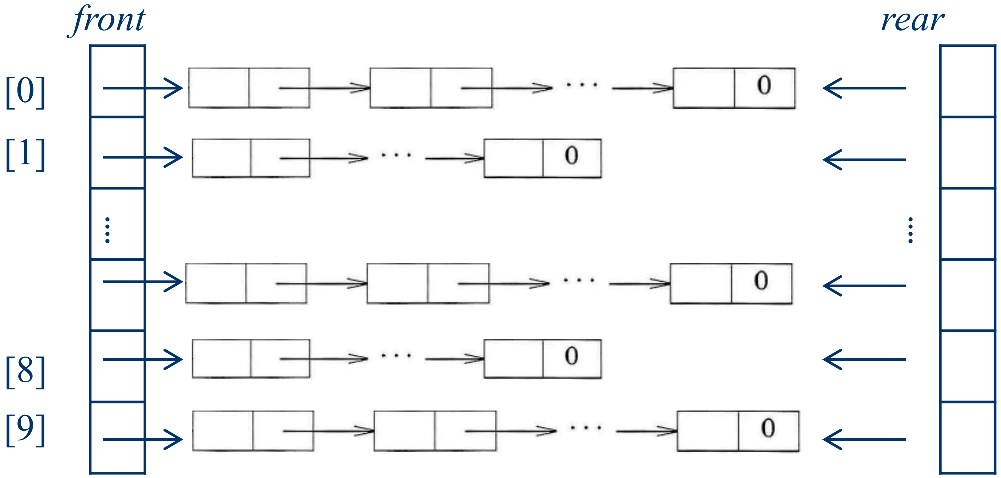


Program 4.6: Delete from a linked stack

4.3 Linked Stacks And Queues



(a) Linked queue



(b) Linked multi queue

4.3 Linked Stacks And Queues

- Representing *n≤MAX_QUEUES* queues simultaneously

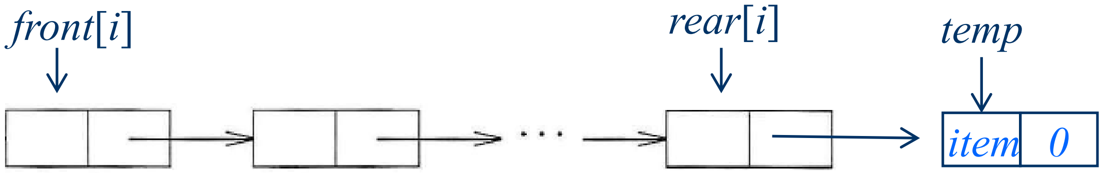
```
#define MAX-QUEUES 10 // maximum number of queues
typedef struct queue *queuePointer;
typedef struct queue {
       element data;
       queuePointer link;
       } ;
queuePointer front[MAX-QUEUES], rear[MAX-QUEUES];
```

- *front[i] =NULL,* 0 ≤ i <*MAX-QUEUES* Initial conditions for the queues
- *front[i]* =NULL iff the *ith* queue is empty Boundary condition for the *i*th queue

4.3 Linked Stacks And Queues

```
void addq(i, item)
{/* add item to the rear of queue i */
        queuePointer temp;
        MALLOC(temp, sizeof(*temp));
        temp->data = item;
        temp->link = NULL;
        if (front [i])
                rear[i]->link =temp;
        else
                front[i] =temp;
        rear[i] =temp;
}
```

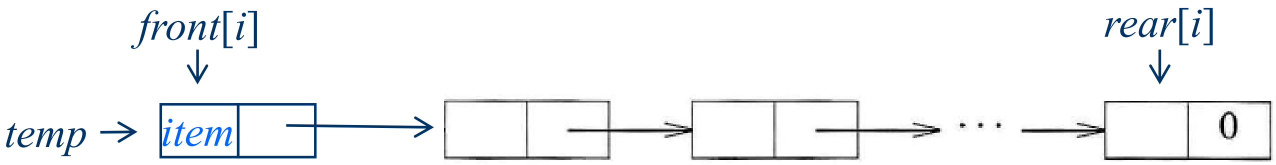
- **Program 4.7: Add to the rear of a linked queue**



4.3 Linked Stacks And Queues

```
element deleteq(int i)
{/* delete an element from queue i */
            element item;
      queuePointer temp= front[i];
      if (!temp)
            return queueEmpty();
      item = temp->data;
      front[i]= temp->link;
      free (temp) ;
      return item;
}
```

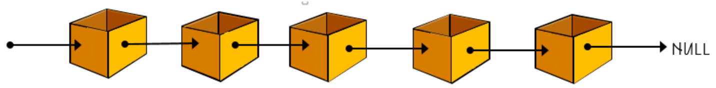
- **Program 4.8: Delete from the front of a linked queue**



Circular List Representation

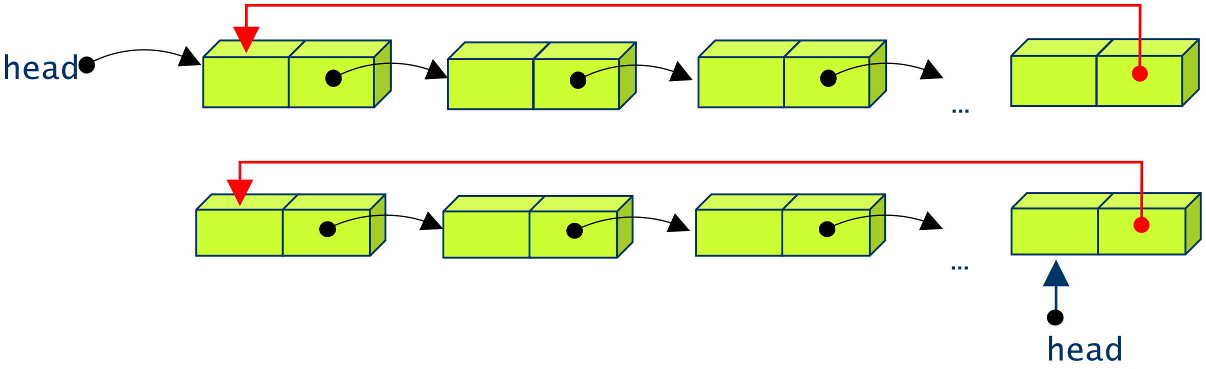
Chain

- A singly linked list in which the last node has a null link



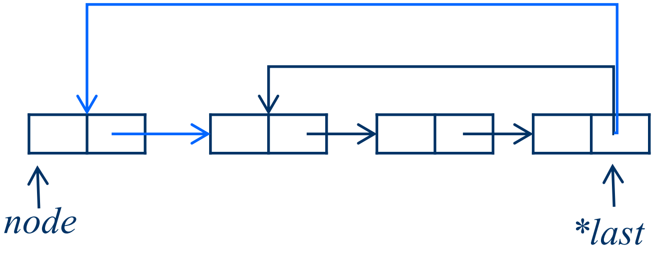
Circular list

- The link filed of the last node points to the first node in the list



Operations For Circularly Linked Lists

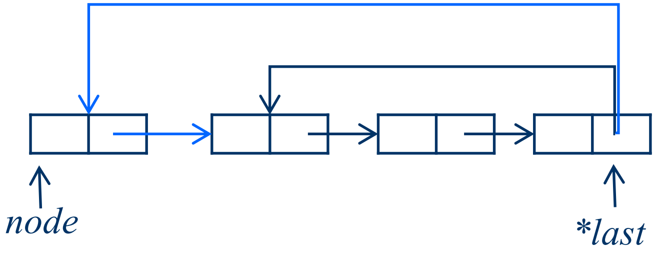
```
void insertFront(listPointer *last, listPointer node)
{/* insert node at the front of the circular list whose
     last node is last */
     if (!(*last)) {
     /* list is empty, change last to point to new entry */
          *last = node;
          node->link = node;
     }
     else {
     /* list is not empty, add new entry at front */
          node->link = (*last) ->link;
          (*last) ->link = node;
          }
}
```



Program 4.18: Inserting at the front of a list

Operations For Circularly Linked Lists

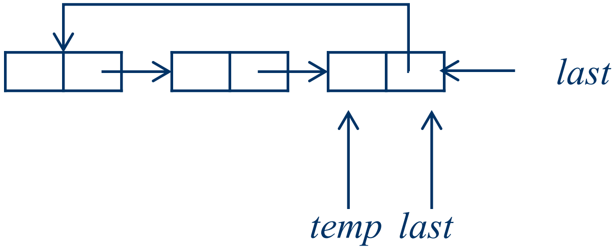
```
void insertLast(listPointer *last, listPointer node)
{/* insert node at the front of the circular list whose
        last node is last */
        if (!(*last)) {
        /* list is empty, change last to point to new entry */
                *last = node;
                node->link = node;
        }
        else {
        /* list is not empty, add new entry at front */
                node->link = (*last) ->link;
                (*last) ->link = node;
                *last = node;
}
```



Program 4.18: Inserting at the front of a list

Operations For Circularly Linked Lists

```
int length(listPointer last)
{/* find the length of the circular list last */
        listPointer temp;
        int count = 0;
        if (last)
                temp = last;
                do {
                count++;
                temp = temp->link;
                } while (temp !=last);
        }
        return count;
}
```



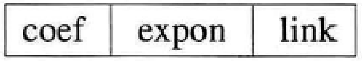
- **Program 4.19: Finding the length of a circular list**

4.4 Polynomials

4.4.1 Polynomial Representation

- **Polynomial**
- $A(x) = a_{m-1}x^{e_{m-1}} + \; \ldots\ldots \; + a_0x^{e_0}$
  - $e_{m-1} > e_{m-2} \ldots > e_1 > e_0 \geq 0$
  - a_i : nonzero coefficients
  - e_i : nonnegative integer exponents
- **Representation of Polynomial**

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



coef	expon	link

4.4 Polynomials

❖ Representation of polynomials

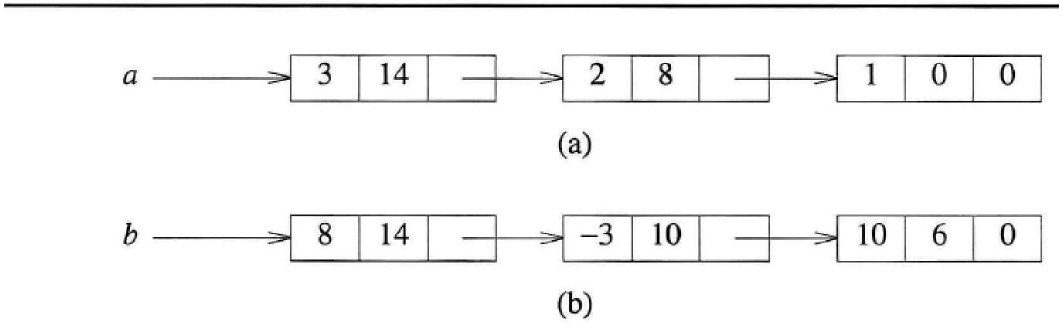


Figure 4.12: Representation of $3x^{14}+2x^{8}+1$ and $8x^{14}-3x^{10}+10x^{6}$

4.4.2 Adding Polynomials

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

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

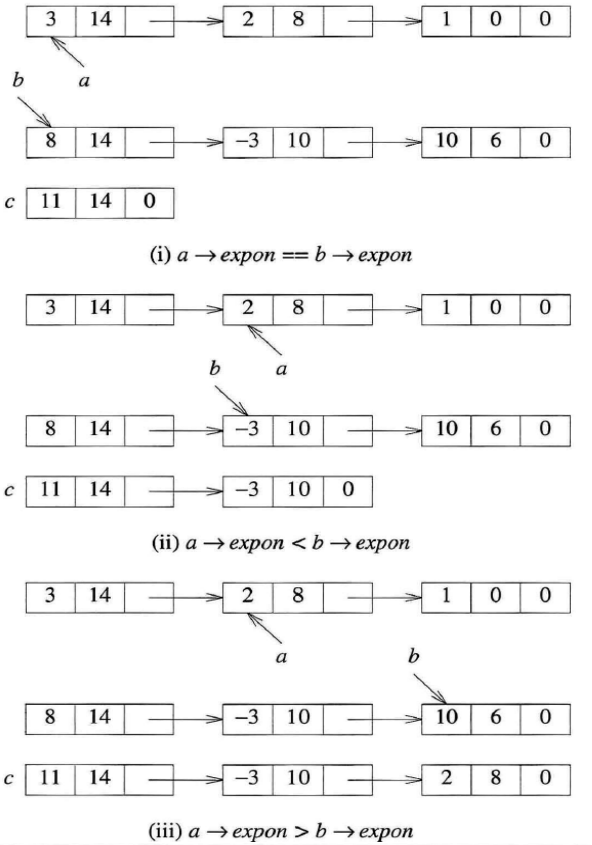
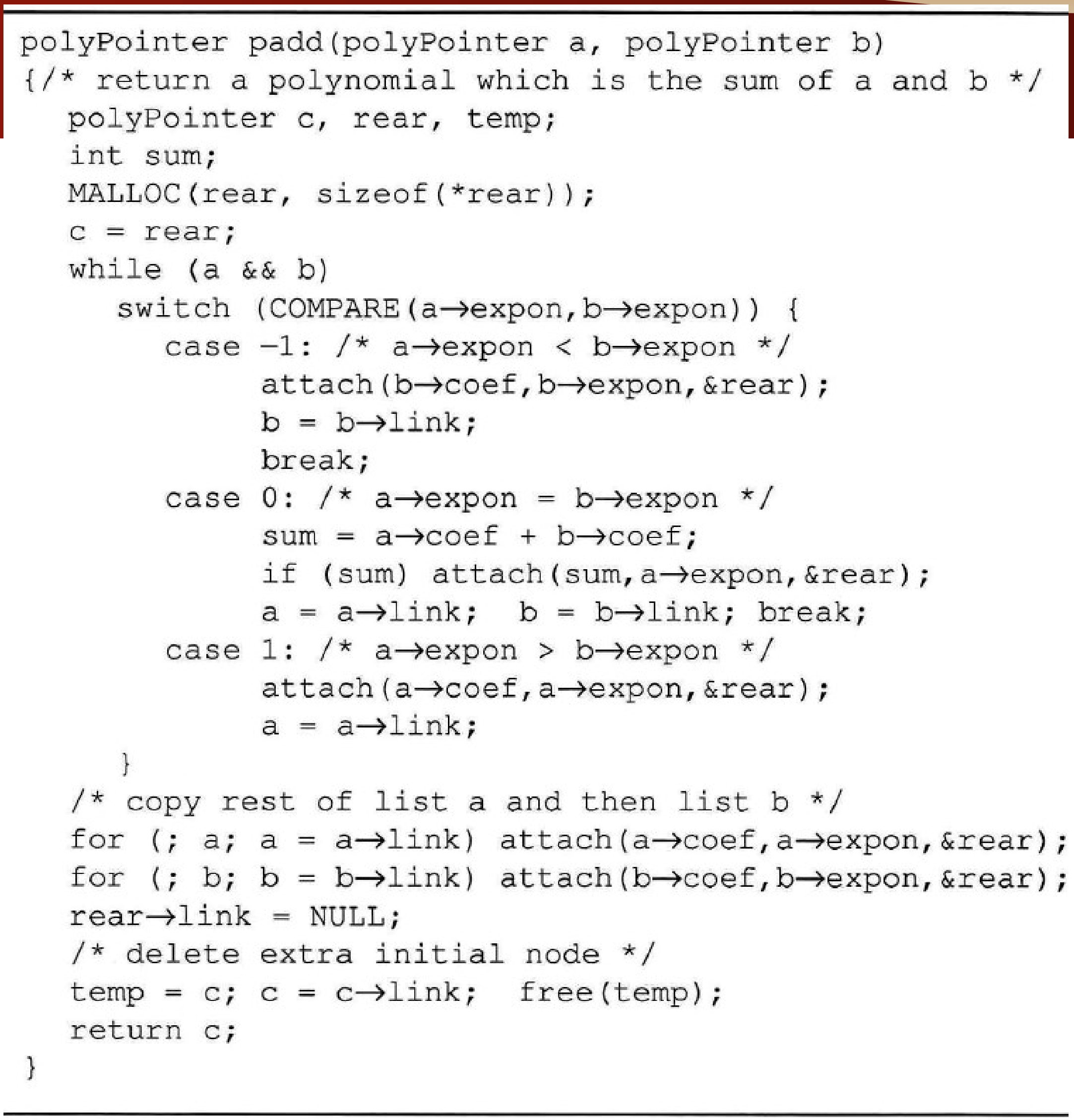


Figure 4.13: Generating the first three terms of $c = a + b$



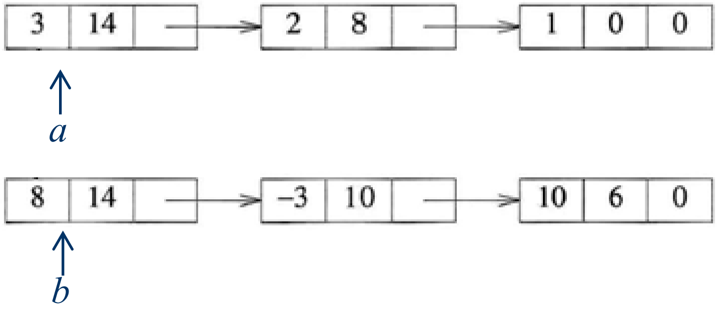
```
polyPointer padd(polyPointer a, polyPointer b)
{/* return a polynomial which is the sum of a and b */
  polyPointer c, rear, temp;
  int sum;
  MALLOC(rear, sizeof(*rear));
  c = rear;
  while (a && b)
    switch (COMPARE(a→expon,b→expon)) {
      case -1: /* a→expon < b→expon */
           attach(b→coef,b→expon,&rear);
           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 = c; c = c→link;  free(temp);
  return c;
}
```

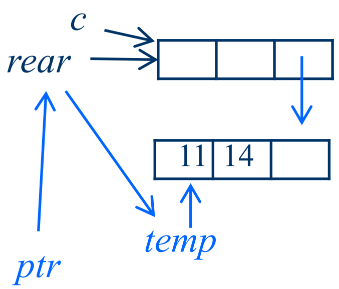
Program 4.9: Add two polynomials

4.4.2 Adding Polynomials

```
void attach(float coefficient, int exponent,
                  polyPointer *ptr)
{/* create a new node with coef = coefficient and expon =
    exponent, attach it to the node pointed to by ptr.
    ptr is updated to point to this new node */
   polyPointer temp;
   MALLOC(temp, sizeof(*temp));
   temp→coef = coefficient;
   temp→expon = exponent;
   (*ptr)→link = temp;
   *ptr = temp;
}
```

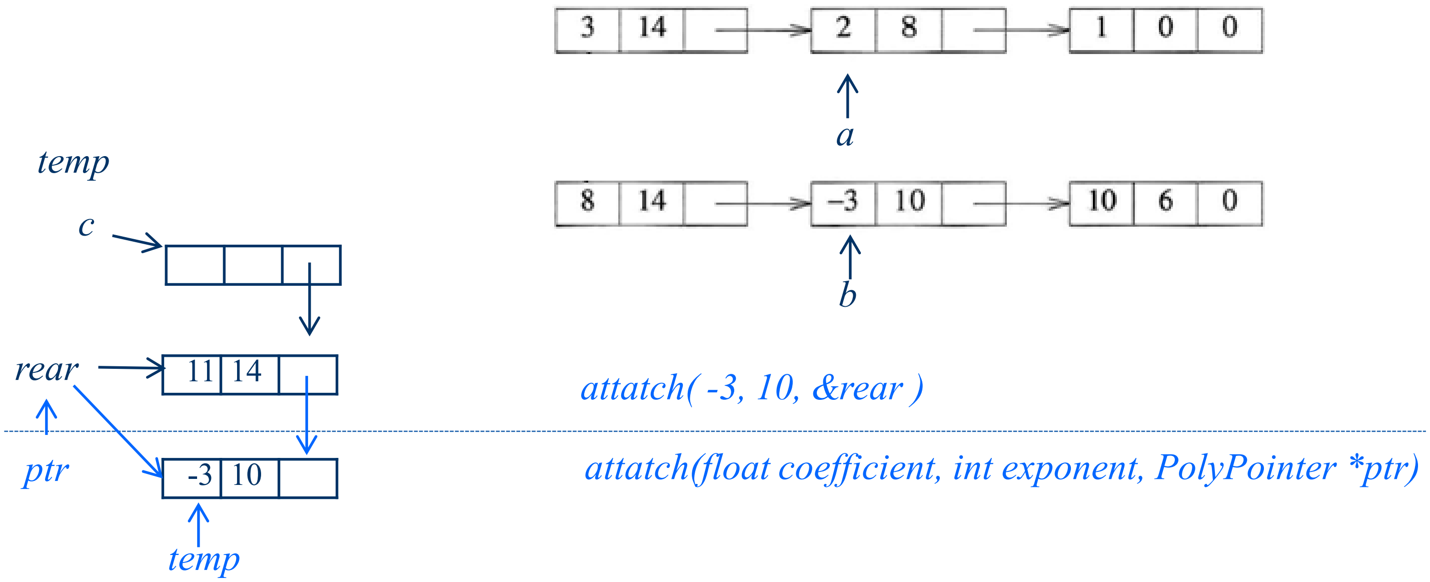
Program 4.10: Attach a node to the end of a list





attatch(11, 14, &rear)

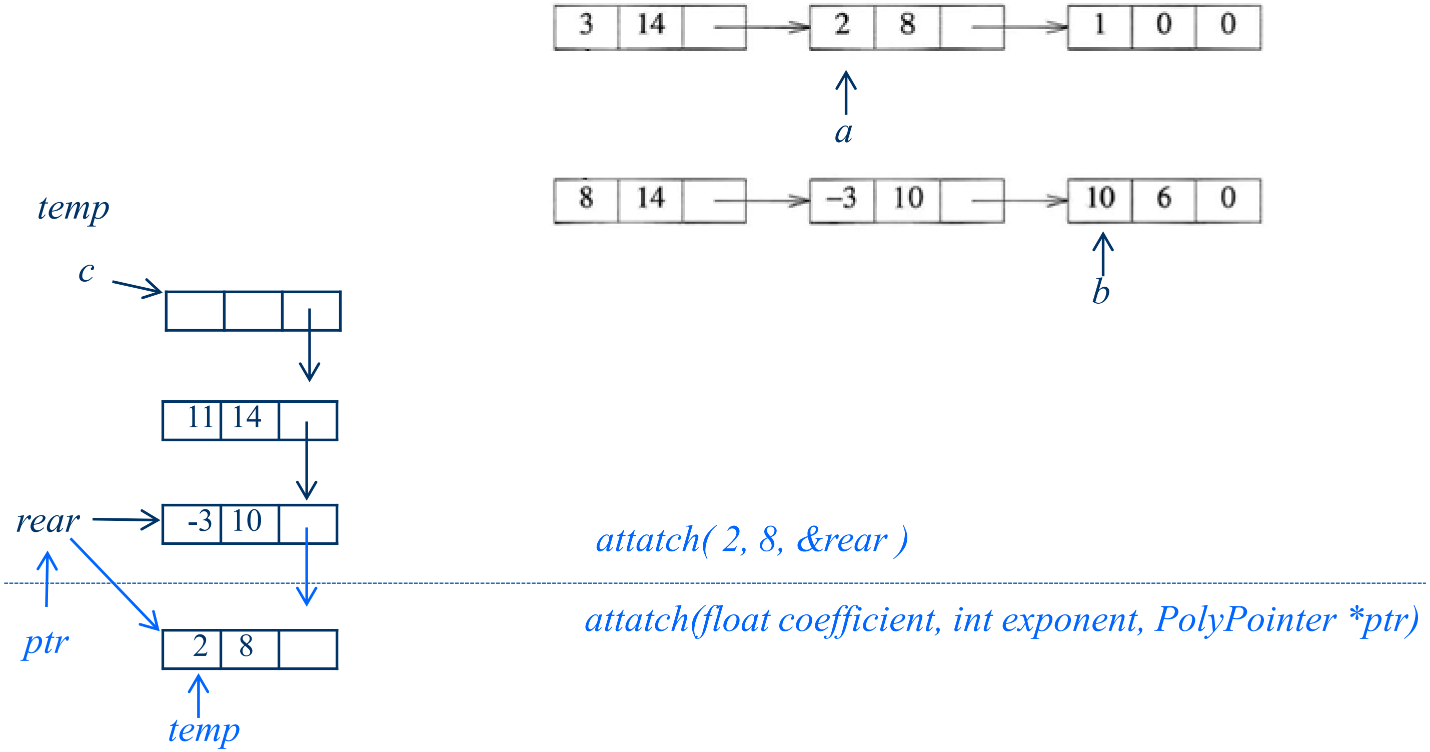
4.4.2 Adding Polynomials



attatch(-3, 10, &rear)

*attatch(float coefficient, int exponent, PolyPointer *ptr)*

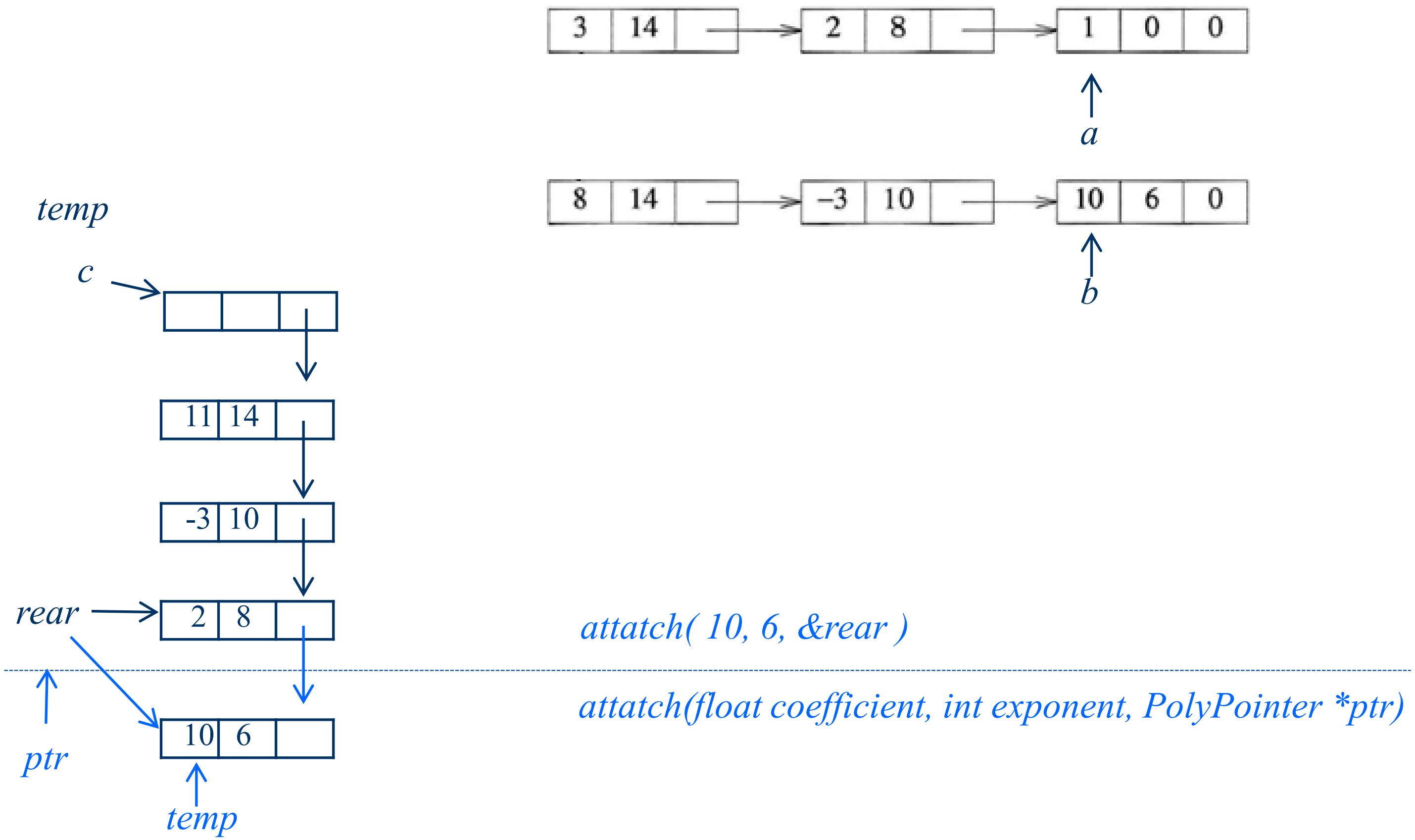
4.4.2 Adding Polynomials



attatch(2, 8, &rear)

*attatch(float coefficient, int exponent, PolyPointer *ptr)*

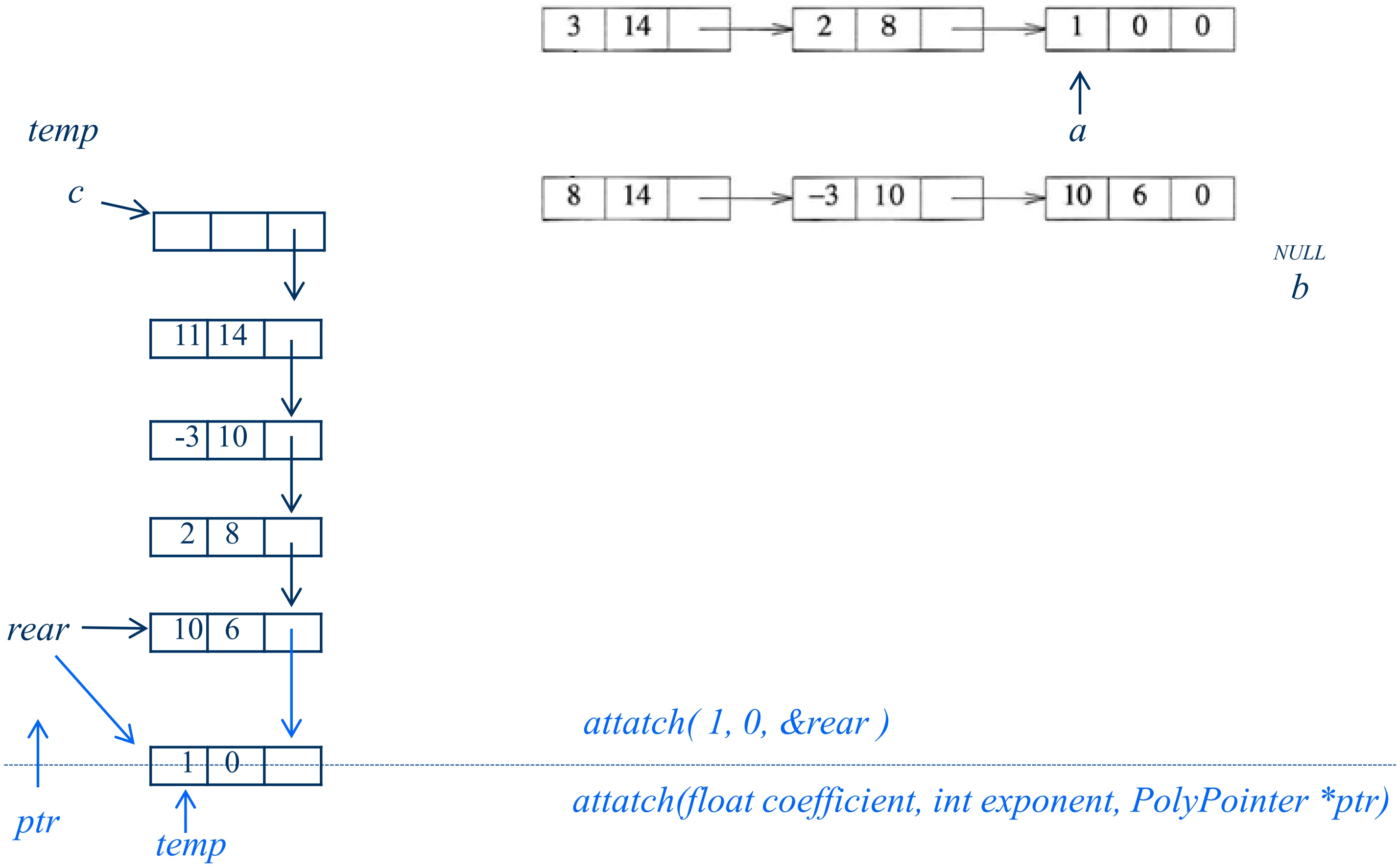
4.4.2 Adding Polynomials



attatch(10, 6, &rear)

*attatch(float coefficient, int exponent, PolyPointer *ptr)*

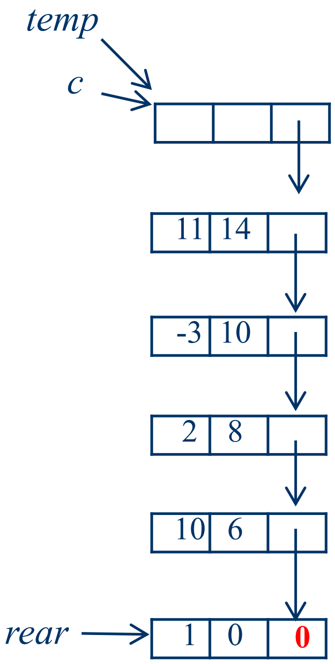
4.4.2 Adding Polynomials

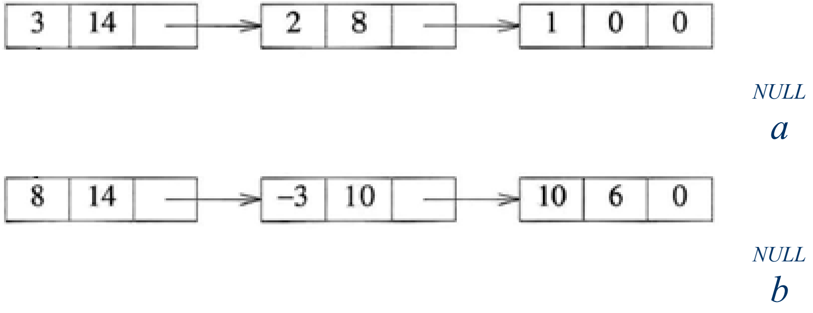


attatch(1, 0, &rear)

*attatch(float coefficient, int exponent, PolyPointer *ptr)*

4.4.2 Adding Polynomials





4.4.2 Adding Polynomials

Analysis of *padd*

- Three cost measures for this algorithm

(1)Coefficient additions

$A(x) = a_{m-1}x^{e_{m-1}} + \quad ..... \quad +a_0x^{e_0}$

$\mathrm{B}(x) = b_{n-1}x^{f_{n-1}} + \quad ..... \quad +b_0x^{f_0}$

where a_i , b_i $\neq 0$ *and* $e_{m-1} > e_{m-2} \ldots > e_1 > e_0 \geq 0,$ $f_{n-1} > f_{n-2} \ldots > f_1 > f_0 \geq 0$

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

4.4.2 Adding Polynomials

(2) Exponent comparisons

- One comparison on each iteration of the `while` loop
- The number of iterations is bounded by *m* + *n*

  ex) *m*+*n*-1 iterations, for example *m* == *n*

  $e_{m-1} > f_{n-1} > e_{m-2} > f_{n-2} \ldots > e_1 > f_1 > e_0 > f_0$

  다항식 a의 지수

  다항식 b의 지수

(3) Creations of new nodes for *c*

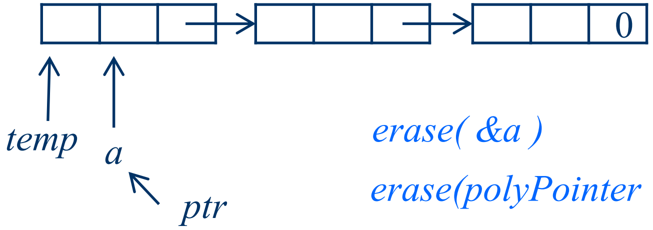
- The maximum number of terms in *c* is *m* + *n*

From (1)~(3),

- the total time complexity is **O (*m* + *n*)**

4.4.3 Erasing Polynomials

```
void erase(polyPointer *ptr)
{/* erase the polynomial pointed to by ptr */
      polyPointer temp;
      while ( *ptr) {
            temp = *ptr;
            *ptr = (*ptr)->link;
            free (temp);
      }
}
```



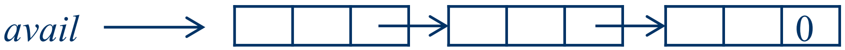
erase(&a)

*erase(polyPointer *ptr)*

4.4.4 Circular List Representation of Polynomials

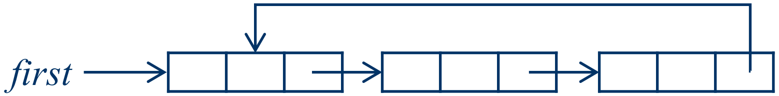
Available space list

- *A chain of nodes that have been "freed"*
- Use *getNode* and *retNode*, instead of *malloc* & *free*



When maintaining it,

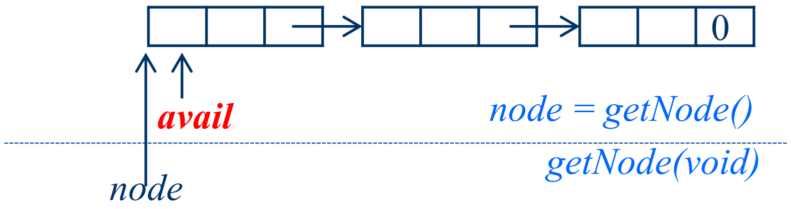
- we can obtain an efficient erase algorithm for circular list.



4.4.4 Circular List Representation of Polynomials

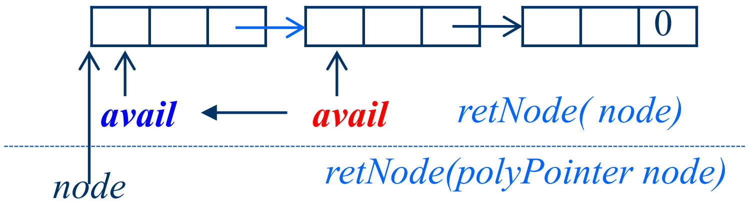
```
polyPointer getNode(void)
{/* provide a node for use */
        polyPointer node;
        if (avail) {
                node = avail;
                avail = avail->link;
        else
        }
                MALLOC(node, sizeof(*node));
        return node;
}
```

Program 4.12: *getNode* function



```
void retNode(polyPointer node)
{/* return a node to the available list */
        node->link = avail;
        avail = node;
}
```

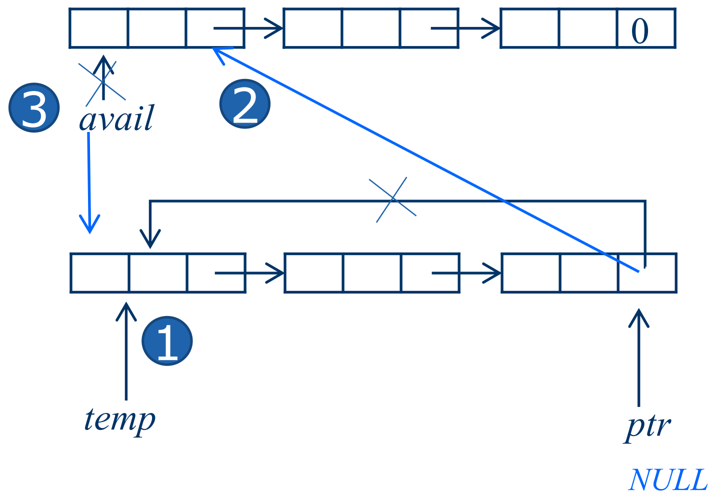
Program 4.13: *retNode* function



4.4.4 Circular List Representation of Polynomials

```
void cerase(polyPointer *ptr)
{/* erase the circular list pointed to by ptr */
        polyPointer temp;
        if ( *ptr) {
                temp = (*ptr)->link;
                (*ptr)->link = avail;
                avail = temp;
                *ptr = NULL;
        }
}
```

Program 4.14: Erasing a circular list



cerase(&first)

4.4.4 Circular List Representation of Polynomials

- A header node is added to easily handle polynomial addition in a circular

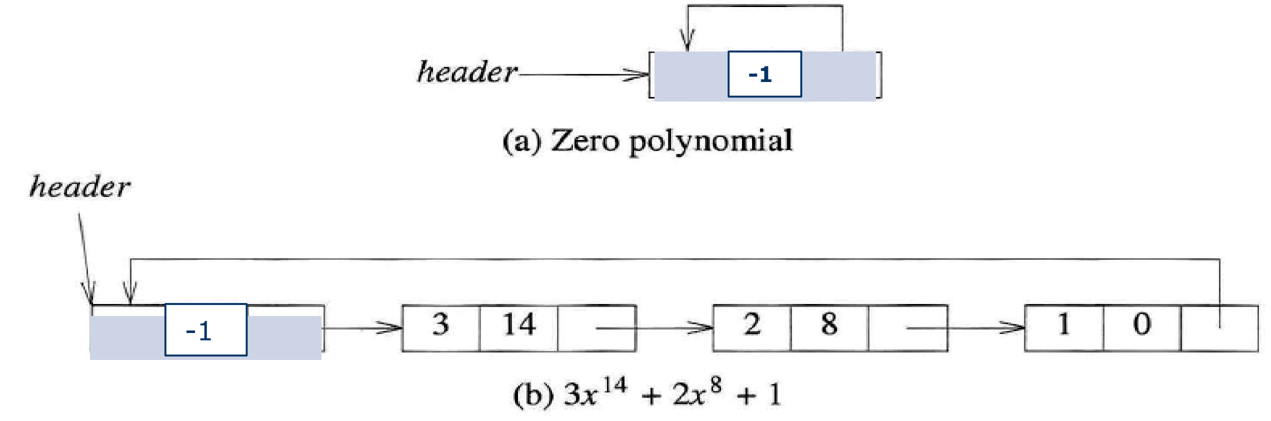
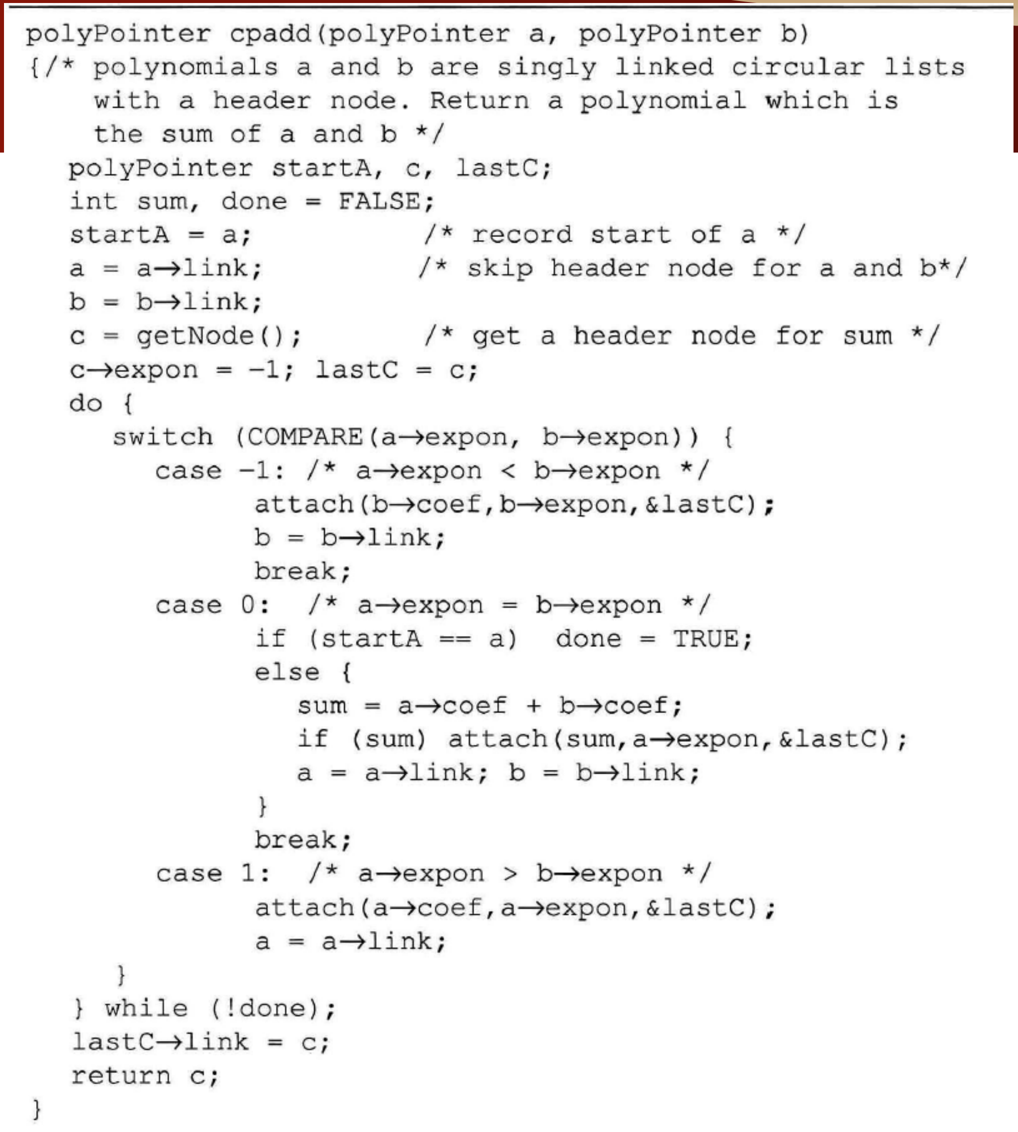


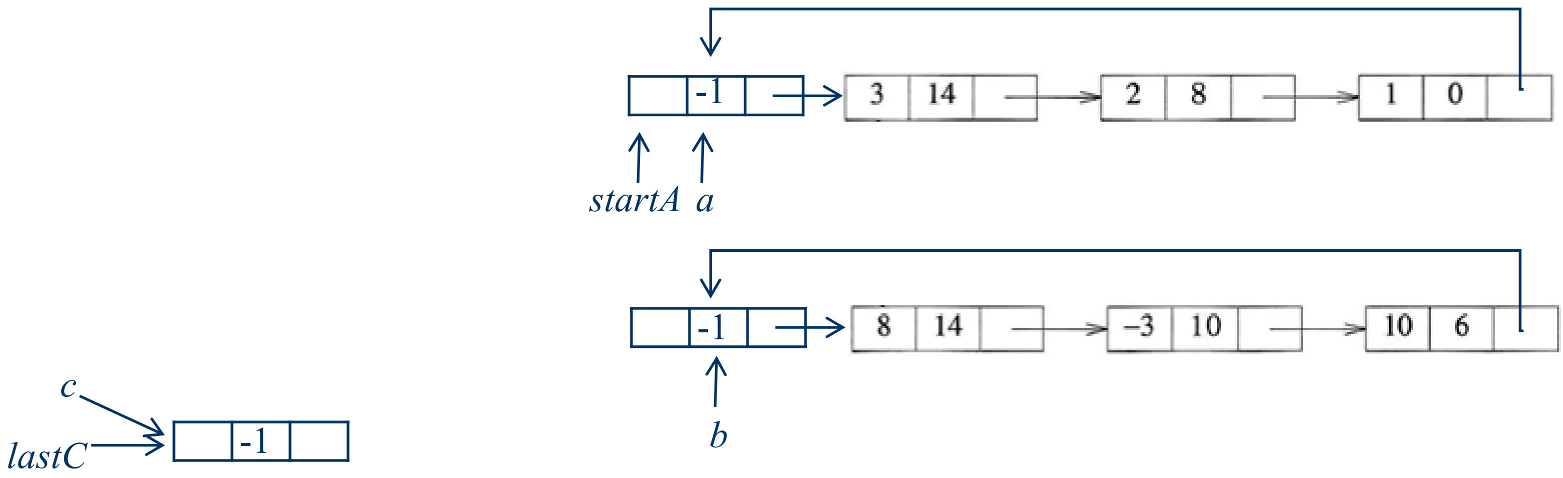
Figure 4.15: Example polynomials with header nodes



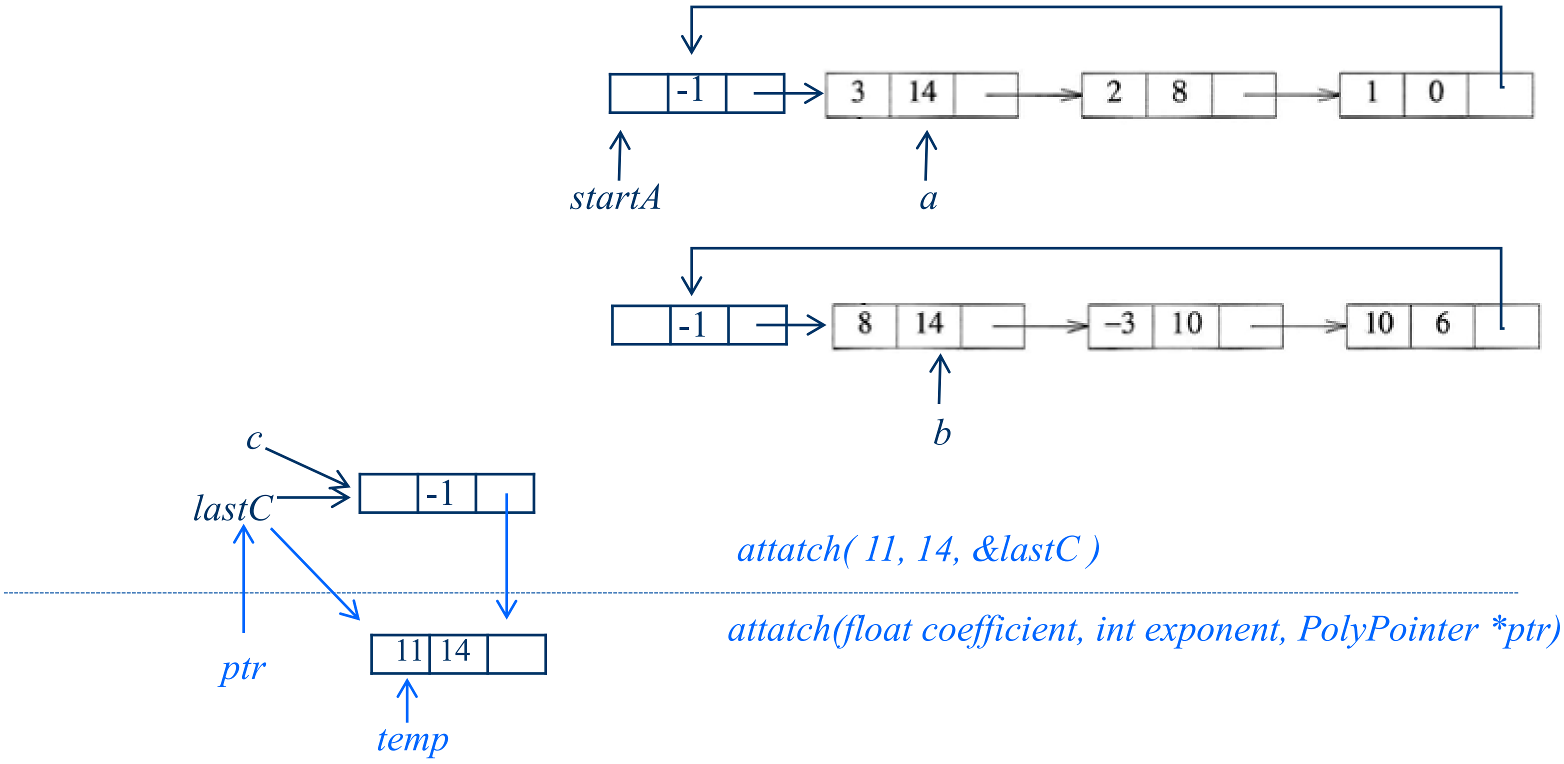
```
polyPointer cpadd(polyPointer a, polyPointer b)
{/* polynomials a and b are singly linked circular lists
    with a header node. Return a polynomial which is
    the sum of a and b */
   polyPointer startA, c, lastC;
   int sum, done = FALSE;
   startA = a;           /* record start of a */
   a = a→link;           /* skip header node for a and b*/
   b = b→link;
   c = getNode();        /* get a header node for sum */
   c→expon = -1; lastC = c;
   do {
      switch (COMPARE(a→expon, b→expon)) {
         case -1: /* a→expon < b→expon */
               attach(b→coef,b→expon,&lastC);
               b = b→link;
               break;
         case 0:  /* a→expon = b→expon */
               if (startA == a)  done = TRUE;
               else {
                  sum = a→coef + b→coef;
                  if (sum) attach(sum,a→expon,&lastC);
                  a = a→link; b = b→link;
               }
               break;
         case 1:  /* a→expon > b→expon */
               attach(a→coef,a→expon,&lastC);
               a = a→link;
      }
   } while (!done);
   lastC→link = c;
   return c;
}
```

Program 4.15: Adding two polynomials represented as circular lists with header nodes

4.4.4 Circular List Representation of Polynomials



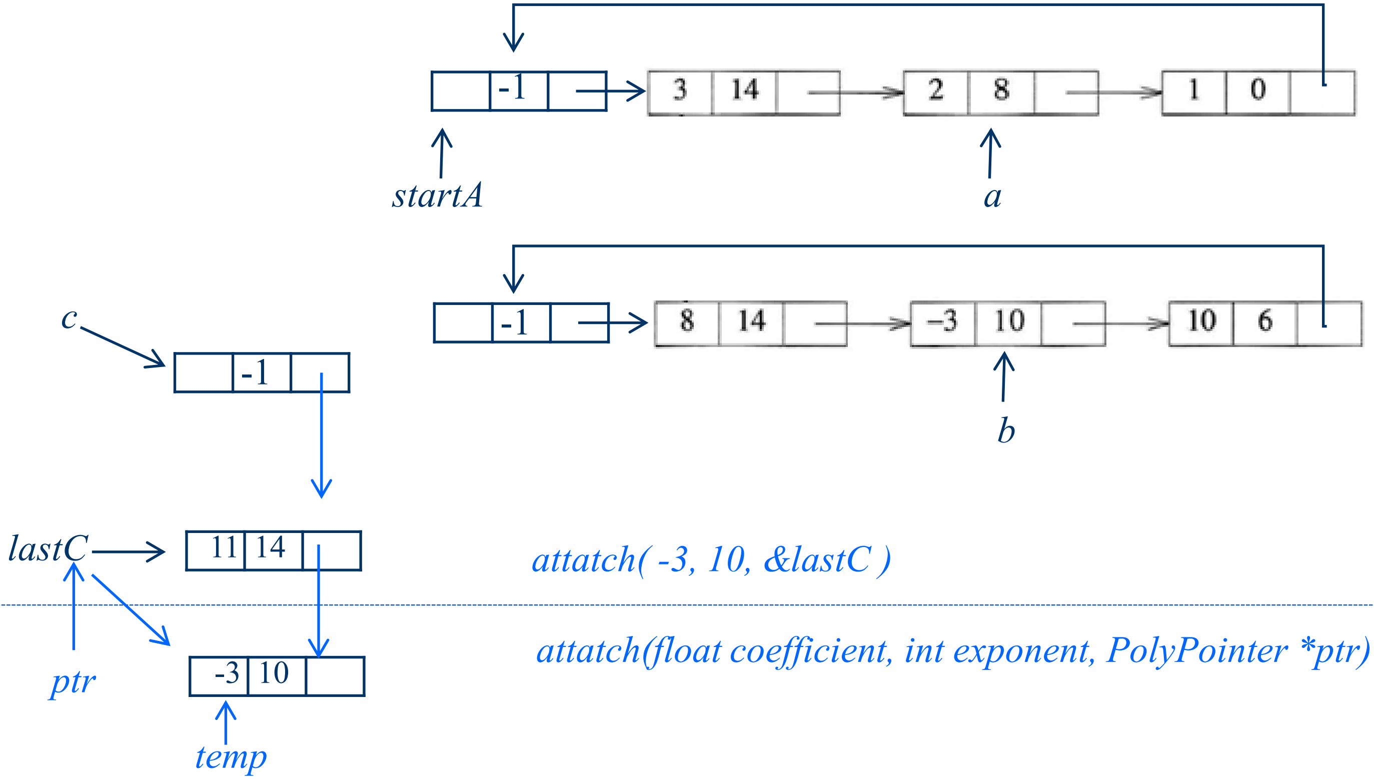
4.4.4 Circular List Representation of Polynomials



attatch(11, 14, &lastC)

*attatch(float coefficient, int exponent, PolyPointer *ptr)*

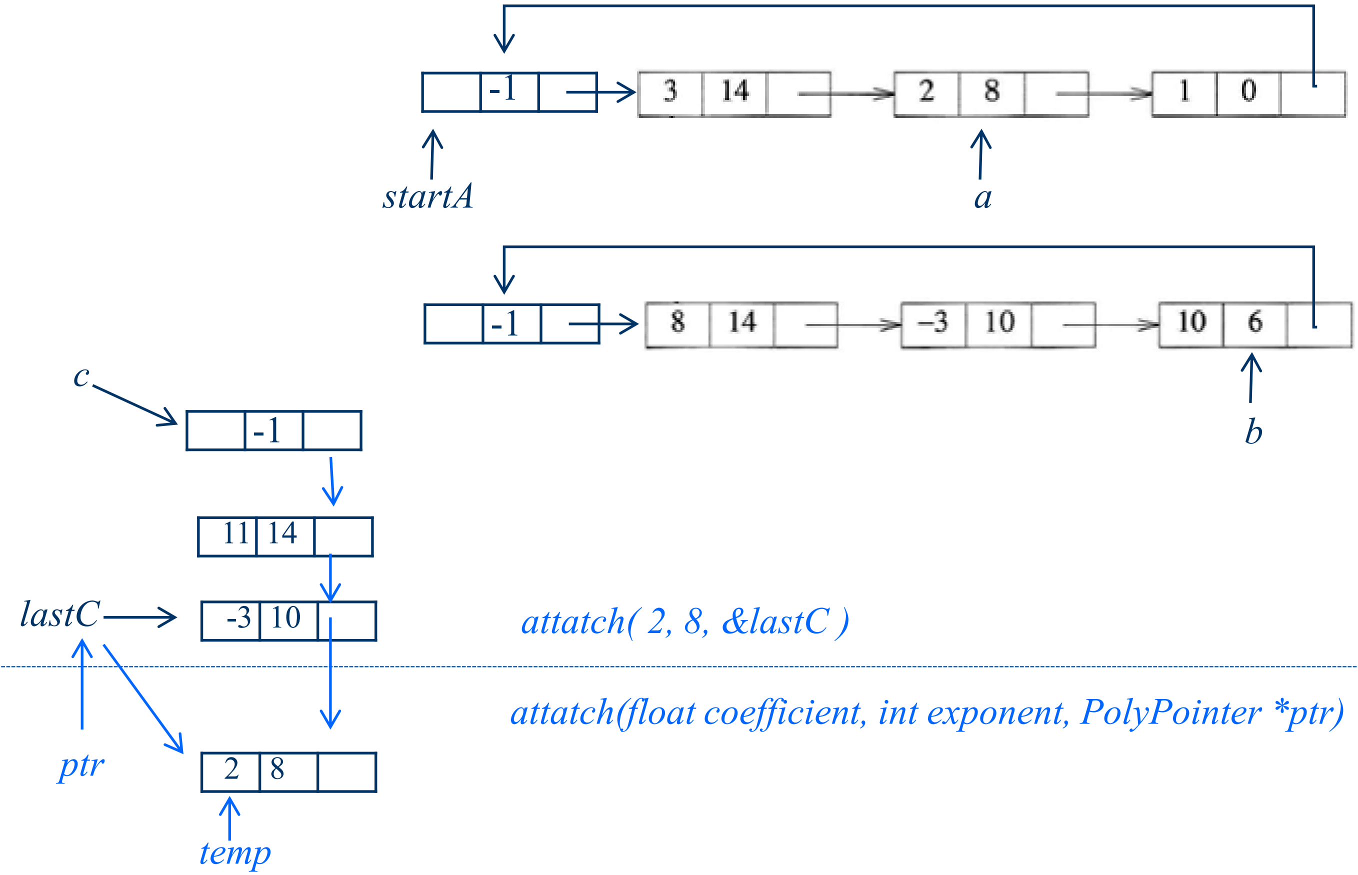
4.4.4 Circular List Representation of Polynomials



attatch(-3, 10, &lastC)

*attatch(float coefficient, int exponent, PolyPointer *ptr)*

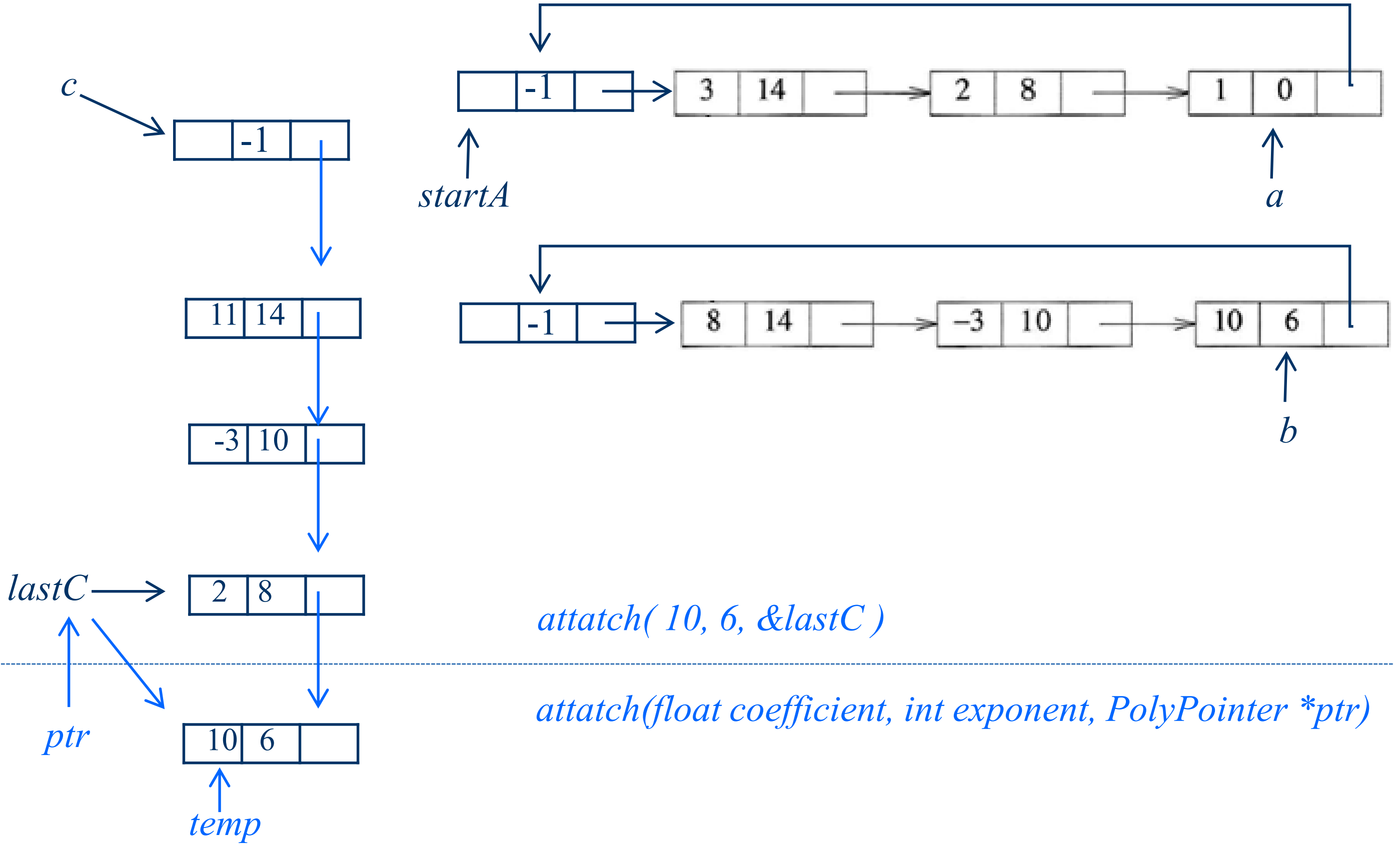
4.4.4 Circular List Representation of Polynomials



attatch(2, 8, &lastC)

*attatch(float coefficient, int exponent, PolyPointer *ptr)*

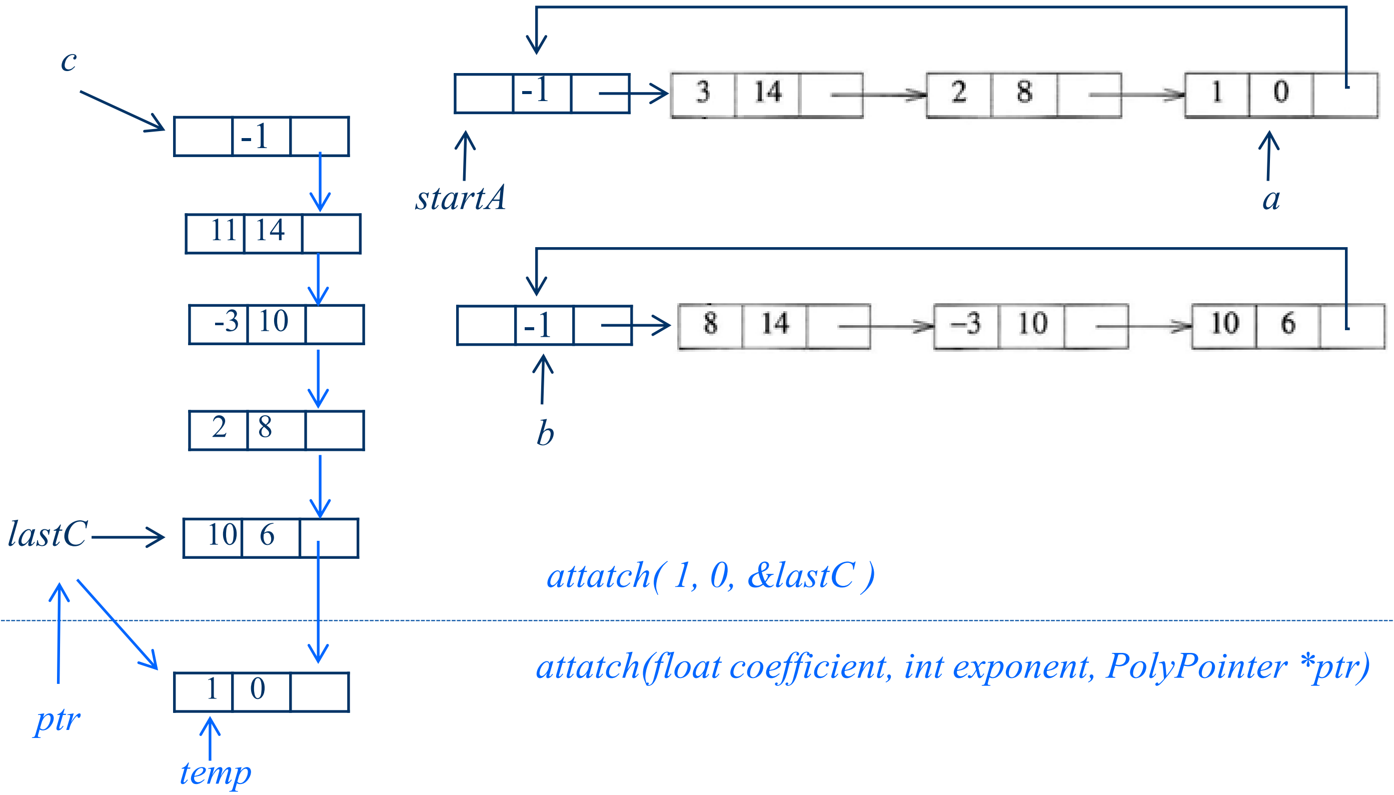
4.4.4 Circular List Representation of Polynomials



attatch(10, 6, &lastC)

*attatch(float coefficient, int exponent, PolyPointer *ptr)*

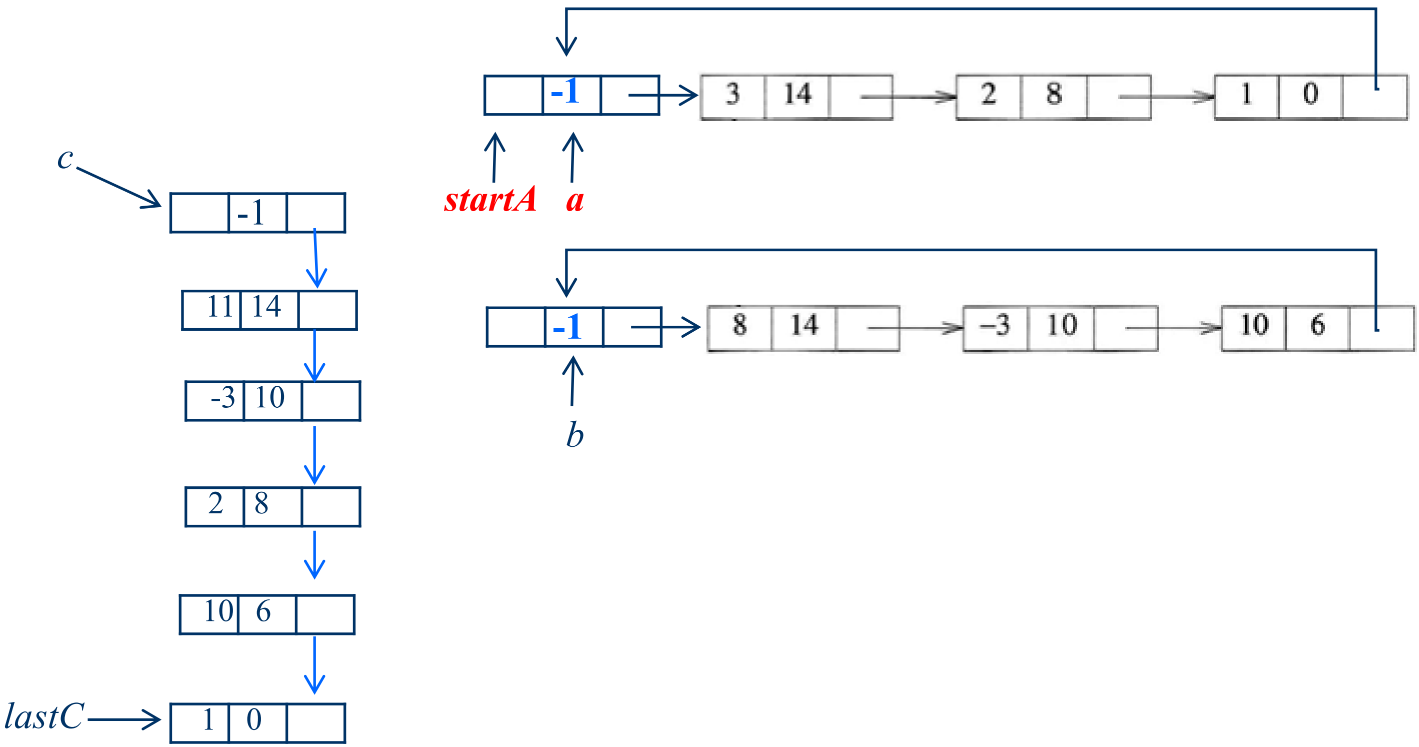
4.4.4 Circular List Representation of Polynomials



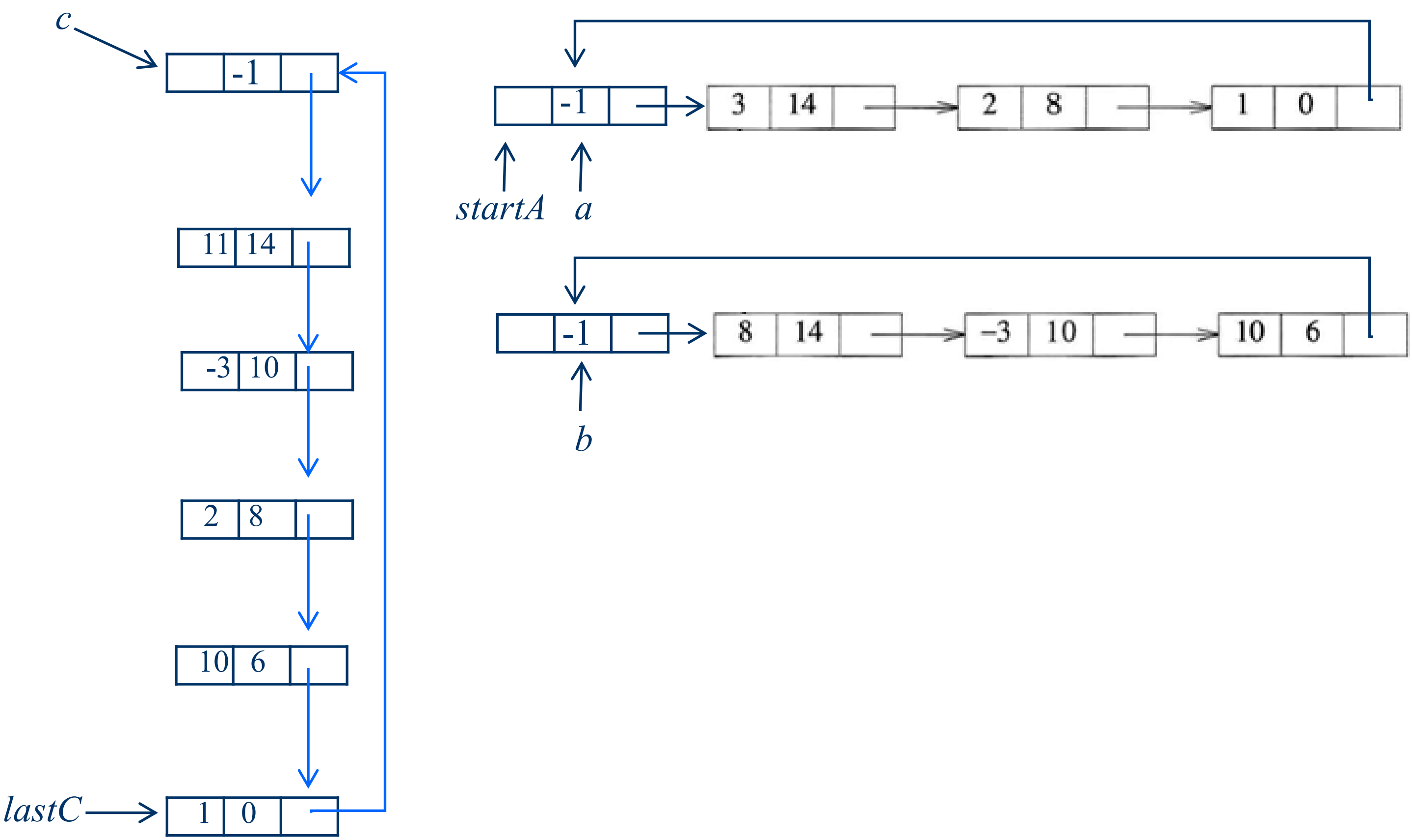
attatch(1, 0, &lastC)

attatch(float coefficient, int exponent, PolyPointer *ptr)

4.4.4 Circular List Representation of Polynomials



4.4.4 Circular List Representation of Polynomials


c
-1
11
14
-3
10
2
8
10
6
lastC
1
0
-1
3
14
2
8
1
0
startA
a
-1
8
14
–3
10
10
6
b