数据结构
Data Structure

2017年秋季学期

刘鹏远

链表其余操作的实现

线性结构应用

- 集合合并
- 有序表合并
- 多项式表示与运算
- 约瑟夫问题

前插法建立单链表的实现(初始化+建立)

- 从一个空表(带/不带头)开始，重复读入数据：
  - 生成新结点
  - 将读入数据存放到新结点的数据域中
  - 将该新结点插入到链表的前端
- 直到读入结束符为止。

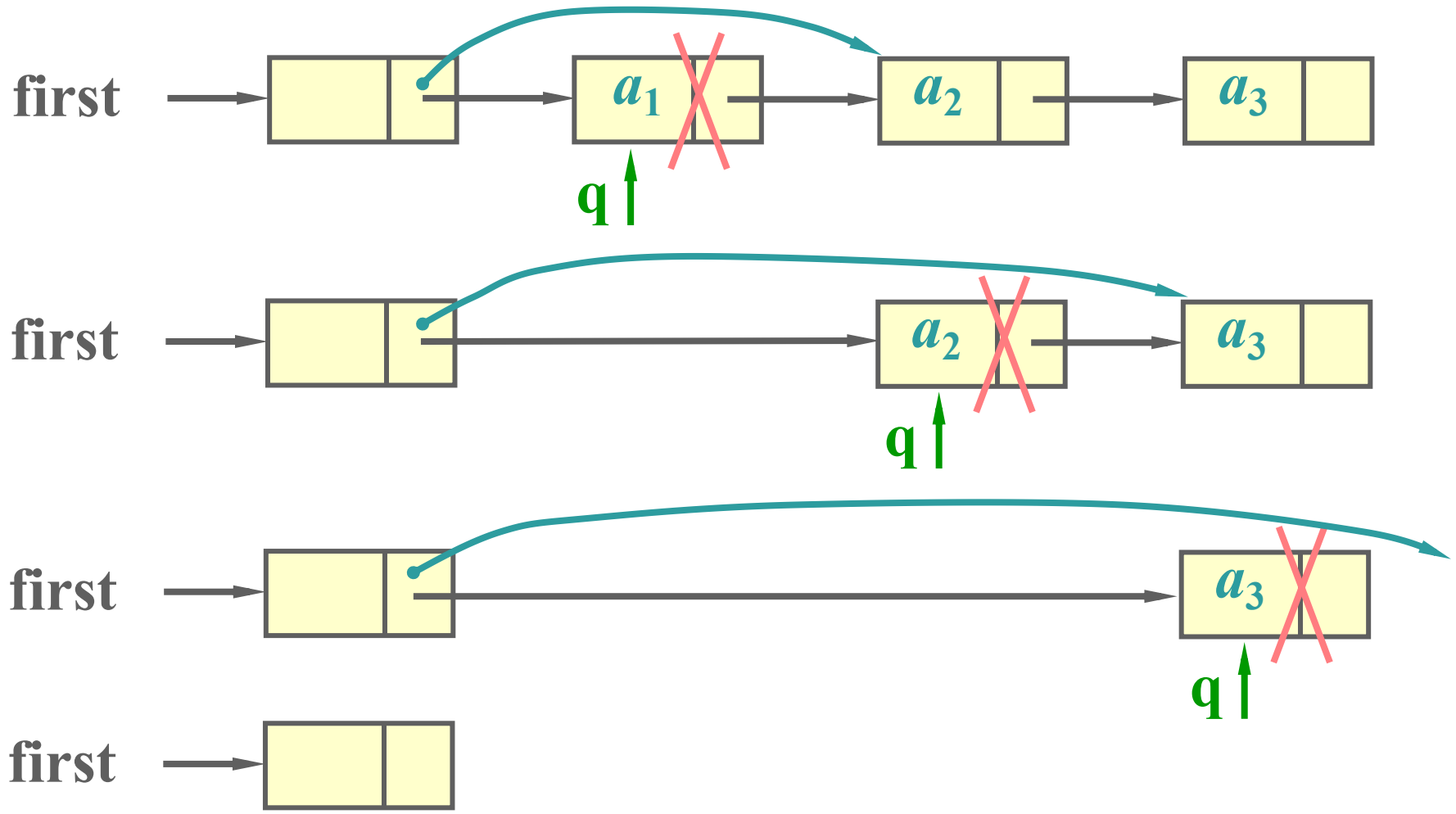
```
link_list Create_List_Fore ( void ) {
    char ch;   list_node* new_node;
    link_list  first = (link_list)malloc(sizeof(list_node));
    first->next = NULL;
    while ((ch = getchar()) != '\n') {
        new_node = (link_list)malloc(sizeof(list_node));
        new_node->data = ch;
        new_node->next = first->next;  //头插法插入步骤
        first->next = new_node;
    }
    return first;
}  //主程序可用link_list L = Create_List_Fore()得到新建链表
```

后插法建立单链表的实现(初始化+建立)

- 每次将新结点加在插到链表的表尾；
- 设置一个尾指针 r，总是指向表中最后一个结点，新结点插在它的后面；
- 尾指针 r 初始时置为指向头结点地址。

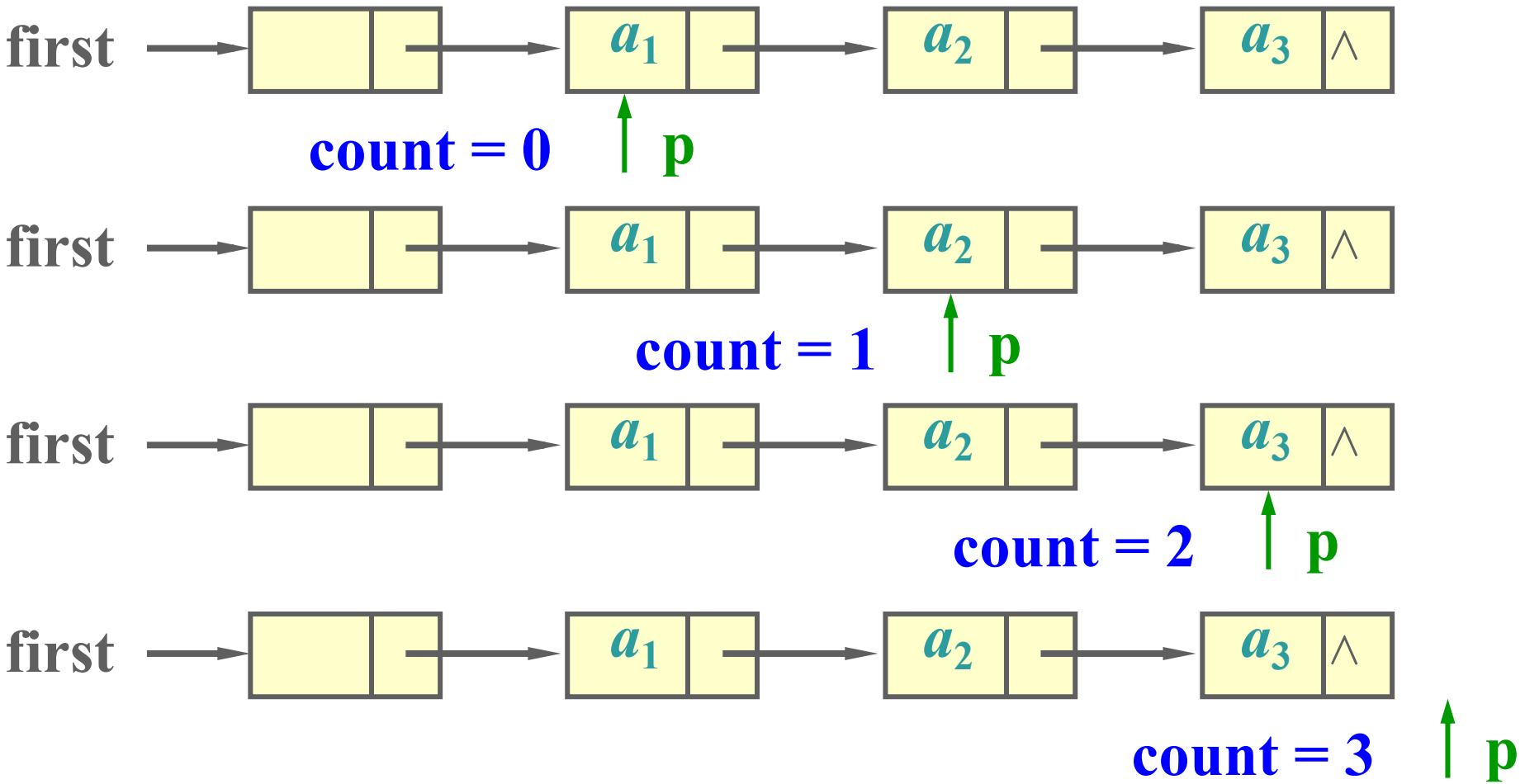
```
link_list Create_List_Back ( void ) {
    link_list  first = (link_list)malloc(sizeof(list_node));
     char ch;   list_node* new_node, *rear=first;
    first->next = NULL;
    while ((ch = getchar()) != '\n') {
        new_node = (link_list)malloc(sizeof(list_node));
        new_node->data = ch;
        rear->next = new_node;//尾插法插入步骤
         rear = new_node;              //始终指向尾节点
    }rear->next = NULL;               //尾节点指空
    return first;
}  //主程序可用link_list L = Create_List_Back()得到新建链表
```

单链表清空



```
Status Clear_List(link_list first){
    link_node * q;
    while(first->next){
        q = first->next;
        first->next = q->next;
        free(q);
    }
}
```

求单链表表长


first
a_1
a_2
a_3
∧
count = 0
p
first
a_1
a_2
a_3
∧
count = 1
p
first
a_1
a_2
a_3
∧
count = 2
p
first
a_1
a_2
a_3
∧
count = 3
p

```
int Length_List(link_list first){
        link_node * p=first->next;
        int len = 0;
        while(p){
                p = p->next;
                len++;
        }
        return len;
}
```

链表其余操作的实现

线性结构应用

- 集合合并
- 有序表合并
- 多项式表示与运算
- 约瑟夫问题

应用一：集合合并。　　顺序表实现

还记得从哪端插入集合元素吗？(还记得顺序表吗？)

```
void union(seqlist* La, seqlist Lb) //伪码
{
 // 将所有在顺序表Lb中但不在La中的数据元素插入到La中
    La_len = Length_List(La);//m
    Lb_len = Length_List(Lb);//n
    int i;
    for (i = 1;  i <= Lb_len;  i++) //n次
```

```
    {
        i f(!find_elem(La, Lb[i], equal))//O（m）
            insert(La, ++La_len, Lb[i]);
                //表尾插入(append),不移动元素，O(1)
        }
} //O（m*n）
//?why use equal?
```

实际实现时，由于一般是数值或者char类型，可以直接比较，因此也可以去掉这个函数参数。

C++中也可以对“=”进行重载。

单链表实现　也要考虑哪端插入数据的问题

```
void union(List La, List Lb)//伪码
{
 // 将所有在单链表Lb中但不在La中的数据插入到La中
    p = Lb->next;
    while (p) //n次  {
        e = p->data;
        p = p->next;
```

```
        if(!find(La, e, equal))//O（m）
            insert_list(La, 1, e); //表头插入，O(1)
        }
}
//O（m*n）
```

需要先实现单链表的find操作

应用二：有序线性表合并

顺序表实现　（与p26. 算法 2.7略不同）

```
void merge_list(List La, List Lb, List* Lc){//伪码
i=j=k=1;
La_len = La.length;    Lb_len = Lb.length;
While ((i<=La_len)&&(j<=Lb_len))
     {  a=La[i]; b=Lb[j]);
        if (a<=b){insert(Lc, k++, a); i++;}
        else { insert(Lc, k++, b); j++;}
  }
```

```
  while (i<=La_len)
  {
        a= La[i++]; insert(Lc, k++, a);
  }
  while (j<=Lb_len)
  {
        b=Lb[j++]; insert(Lc,k++, b);
  }
}//O(La.length+Lb.length)insert位置为尾插入
//程序结束时，k的值=?
```

单链表实现　　p31. 算法2.12　　画图写算法

```
void merge_list(List La, List Lb, List Lc)//伪码
{ pa = La->next; pb = Lb->next;Lc=pc = La;
//a,b指向La,Lb首元素，c指向Lc当前结点
     while(pa&&pb) {
        if(pa->data<=pb->data)
                {pc->next = pa; pc = pa;pa=pa->next;}
        else{pc->next = pb; pc = pb;pb=pb->next;}
    }
    pc->next = pa?pa:pb;//链接剩余链表元素
    free(Lb);
}//O(La.length+Lb.length)
```

应用--多项式 (Polynomial)表示与运算

$$P_n(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n = \sum_{i=0}^{n} c_i x^i$$

n 阶一元多项式 $P_n(x)$ ， 有 $n+1$ 项。

系数 $c_0, c_1, c_2, \ldots, c_n$

指数 0, 1, 2, …, n。按升幂排列

----可抽象成线性表

ADT定义可参考P40页

$$P_n(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0$$

多项式求值的时间复杂度？

求第i项的值要执行 i 次乘法，总共执行：

n+(n-1)+…+1 = n(n+1)/2 次乘法，n 次加法

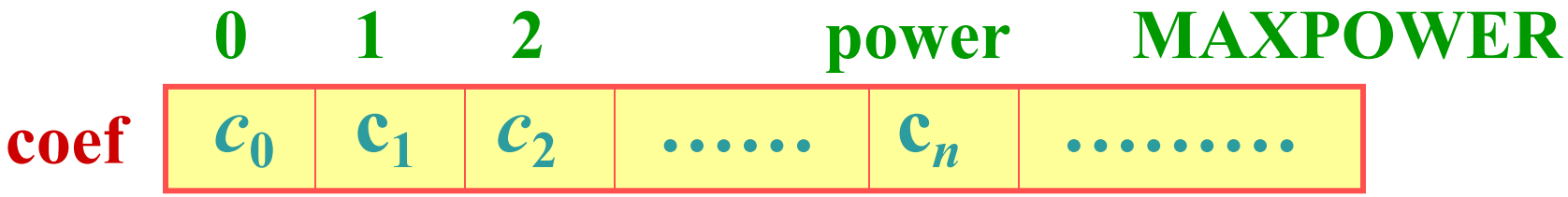
O（n^2）可否降低？

$$= ((\ldots((c_n x + c_{n-1})\, x + c_{n-2})\, x + \ldots)\, x + c_1)\, x + c_0$$

从最内层开始计算，逐层向外，直到求出最后的解，总共执行 n 次乘法，n 次加法。

多项式的存储表示

第一种顺序表示：静态数组表示（动态顺序表也类似）




```
#define MAXPOWER  100     //最大允许阶数
  typedef struct Polynomial {          //多项式结构定义
        int power;                     //实际阶数，定尾
        float coef [MAXPOWER  +1];   //系数数组
  };
```

$P_n(x)$ 可以表示为：

Polynimial pl;

pl.power = n;

pl.coef[i] = a_i, $0 \leqslant i \leqslant n$

在这种存储表示中，x^i 的系数 c_i 存放于 coef[i]，可以简化如相加等各项操作，实现非常简单。

该方法的缺陷？

但对于指数不全的多项式如

$$P_{101}(x) = 3 + 5x^{50} - 14x^{101}$$

coef[] 数组大小有 102 个元素，其中只有 3 个元素非零，不经济。计算起来也不科学。

一般只适用于指数连续排列的多项式（稠密的，少缺项的）。

对稀疏的多项式，如何做较好？

第二种顺序表示：只保存非零系数项（最常用）

```
typedef struct elem_type {        //多项式的项定义
    float coef;                   //系数
    int exp;                      //指数
}elem_type;
typedef struct Polynomial {       //多项式定义
    int max_size;                 //数组最大保存项数
    int count;                    //实际项数
    elem_type * elem;             //项组
}poly;                  //就是一般顺序表，只是数据元素有变化
```

	0	1	2		*i*		*m*
coef	a_0	a_1	a_2	……	a_i	……	a_m
exp	e_0	e_1	e_2	……	e_i	……	e_m

保存系数 a_i 和指数 e_i。 ploy pl;初始化为

```
pl.max_size = MAXSIZE;
pl.elem = (elem_type *)malloc(pl.max_size*sizeof(elem_type));
    pl.count = n;//实际项数
    pl.elem[i].coef = ai;
    pl.elem[i].exp = ei;                    0≤i≤n
```

第三种：多个多项式共存一个数组（顺序存储）

```
#define MAX_NUMBER 5;        //最大多项式个数
typedef struct Polynomial {    //多项式定义
        int max_size;                  //数组最大保存项数
        int[MAX_NUMBER] start, finish;       //多项式始末位置
        int m;当前多项式个数
        elem_type * elem;              //项组
}poly;
```

对第i个多项式 pl，pl.start[i] 和 pl.finish[i]分别指明其开始存放位置和最后存放位置。多项式按照指数递增的方式存放。

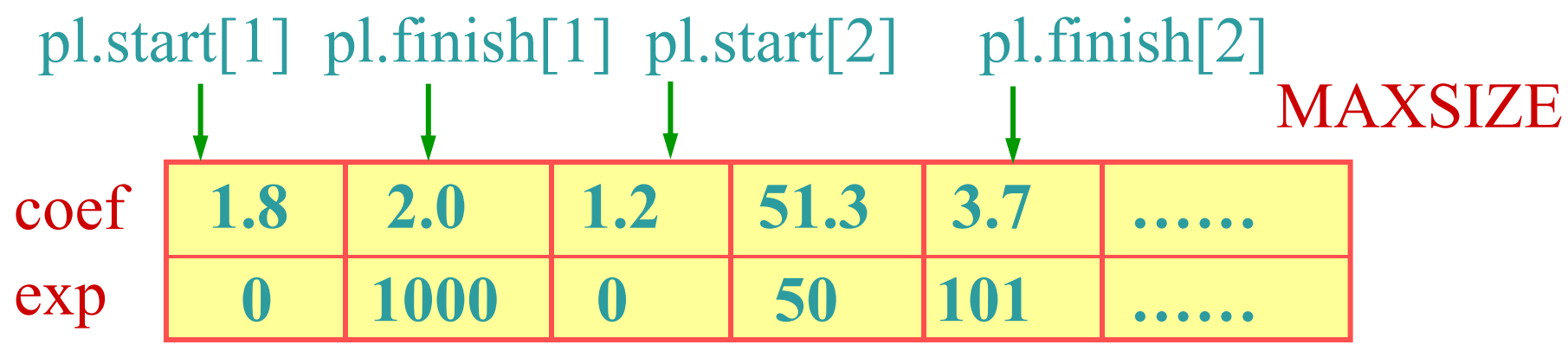
pl.finish[m]+1指明新多项式在项数组中可存放的起始位置。

例如，有两个多项式

$$A(x) = 2.0x^{1000}+1.8$$

$$B(x) = 1.2 + 51.3x^{50} + 3.7x^{101}$$

存放在pl.elem中，如图



设有两个多项式 A 和 B 相加，结果多项式另存于 C。

扫描两个相加多项式，若都未检测完：

1、若当前被检测项指数相等，系数相加。若未变成 0，则将结果加到结果多项式。

2、若当前被检测项指数不等，将指数小者加到结果多项式。

若有一个多项式已检测完，将另一个多项式剩余部分复制到结果多项式。

前提是有序存储的(类似有序线性表合并？)

以顺序表示中较为常用的第2种方式为例：

```
void Add (poly A, poly B,  poly C ) {//伪码
    int i =j =k= 1;
    float tmp;
    while ( i <= A.count&& j <= B.count )
{
        if ( A.elem[i].exp == B.elem[j].exp) )      //对应项指数相等情形
         {
                tmp = A.elem[i].coef + B.elem[j].coef;  //系数加
                if ( tmp ) {C.elem[k].coef = tmp;
  C.elem[k].exp=A.elem[i].exp; C.count++;}
                i++;  j++; k++;
        } //C数组是否满判定略
```

```
else if ( A.elem[i].exp > B.elem[j].exp )
{// C建立新项
        C.elem[k].coef=B.elem[j].coef;
        C.elem[k].exp=B.elem[j].exp;
        C.count++;          j++; k++;     }
else
{   C建立新项
        C.elem[k].coef=A.elem[i].coef;
        C.elem[k].exp=A.elem[i].exp;
        C.count++;          i++; k++;      }
```

```
for (  ; i <= A.count; i++ )    //多项式 A 未检测完
{
        C.elem[k].coef=A.elem[i].coef;
        C.elem[k].exp=A.elem[i].exp;
        C.count++; k++;
}
for (  ; j <= B.count; j++ )    //多项式 B 未检测完
{
        C.elem[k].coef=B.elem[j].coef;
        C.elem[k].exp=B.elem[j].exp;
        C.count++; k++;
}}}
```

多项式的链表表示

在多项式的链表表示中每个结点 data 的构成为：

elem_type { float coef; int exp; };

则该链表可表示为：

优点是：

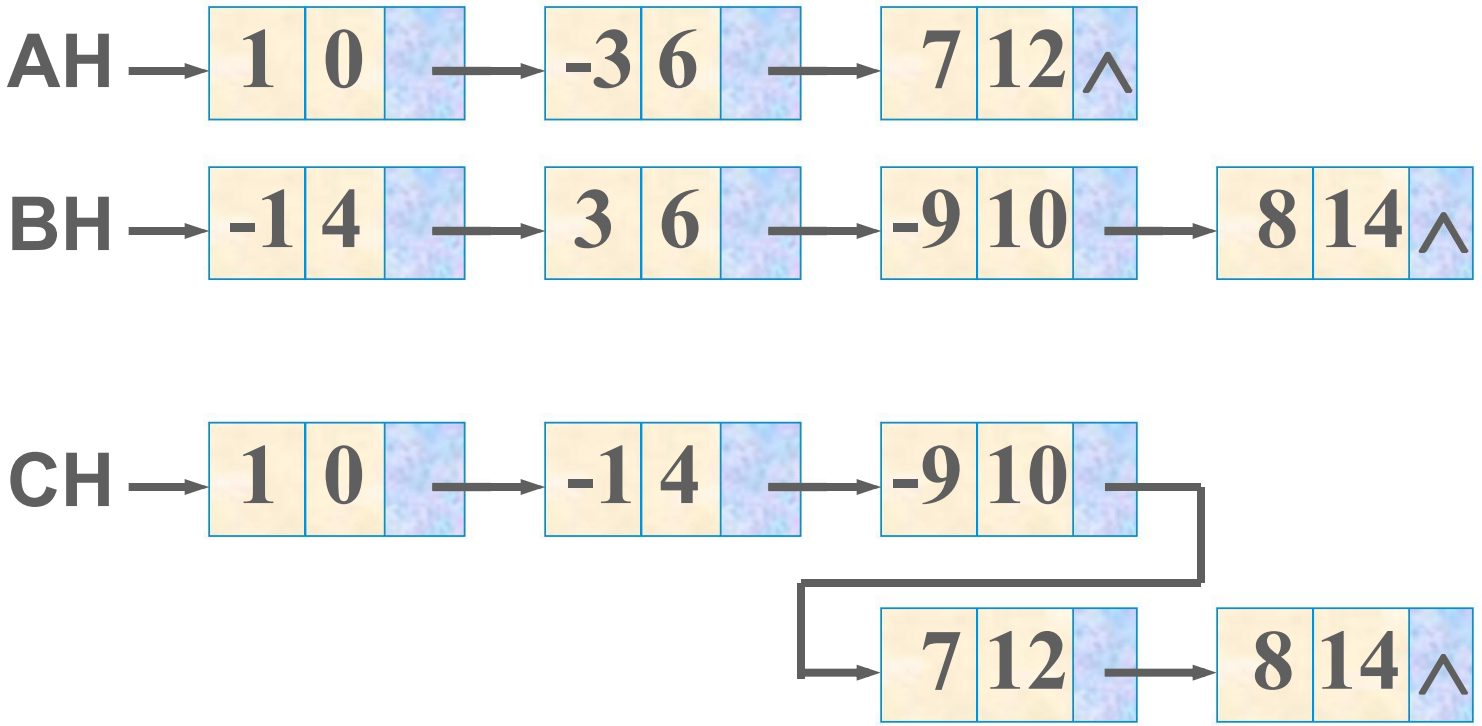
多项式的项数可以动态地增长。

插入、删除方便，无须移动元素。

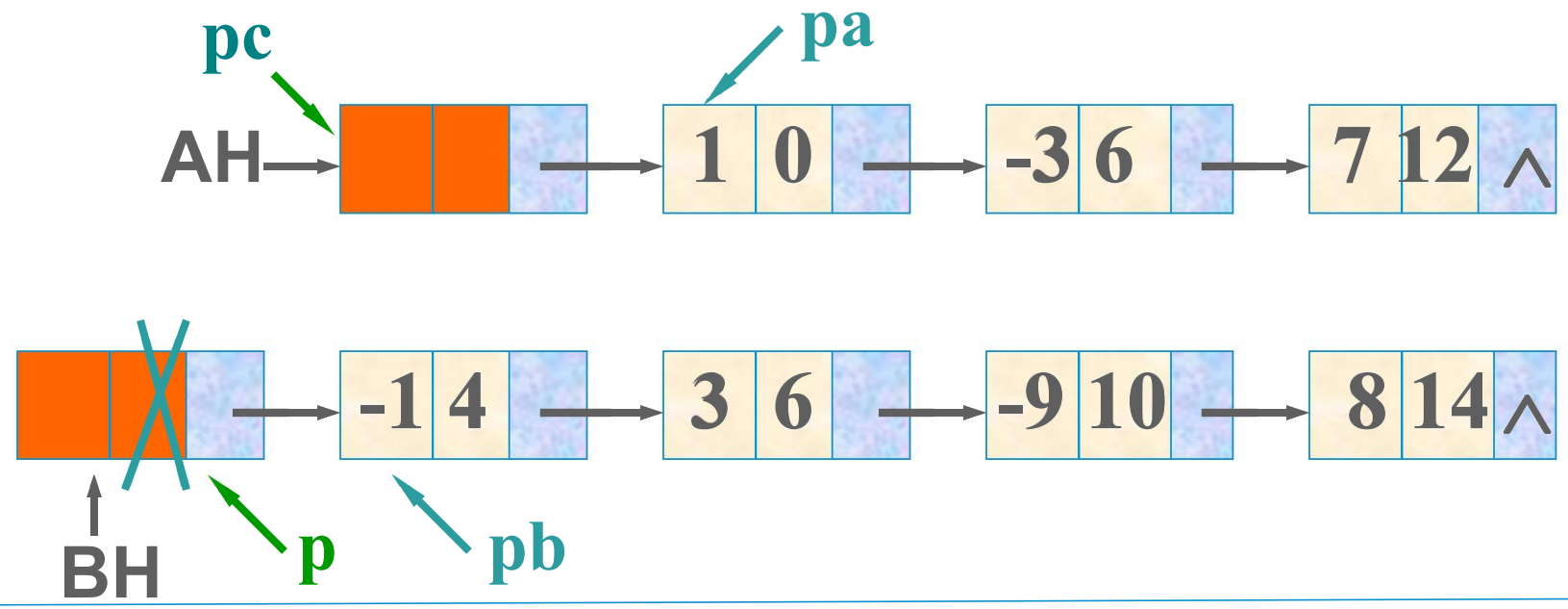
多项式链表的相加

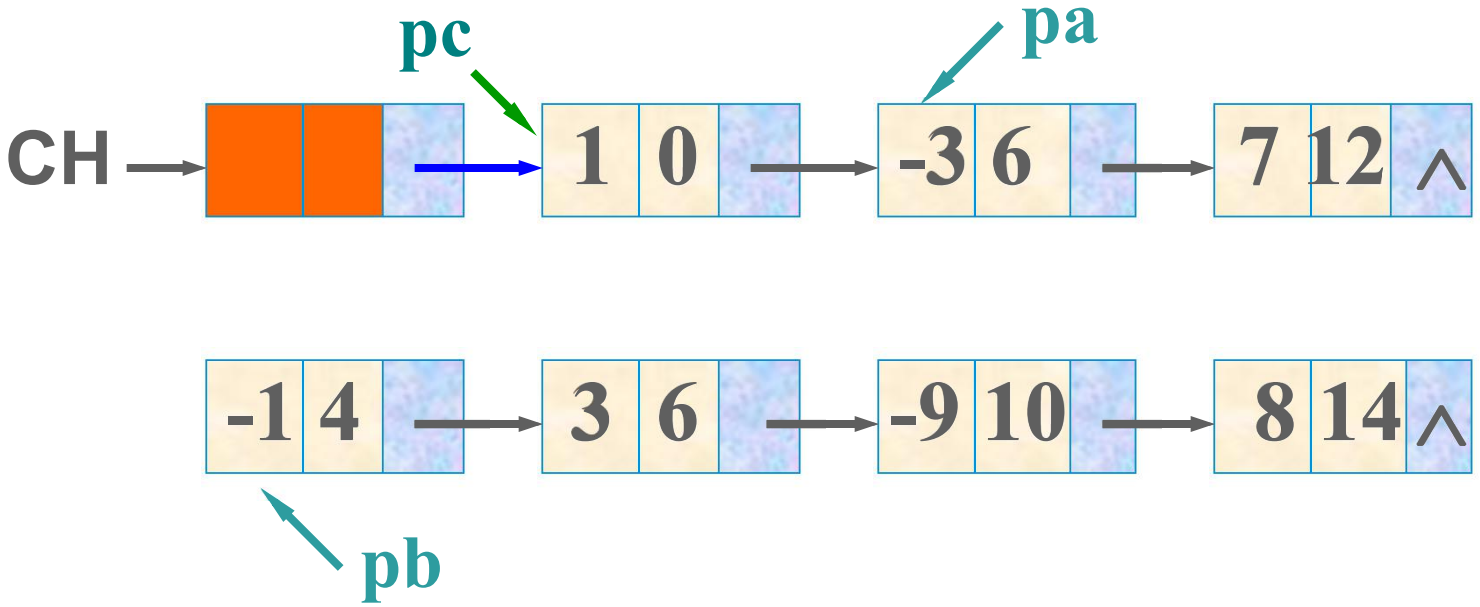
$$AH = 1 - 3x^6 + 7x^{12}$$

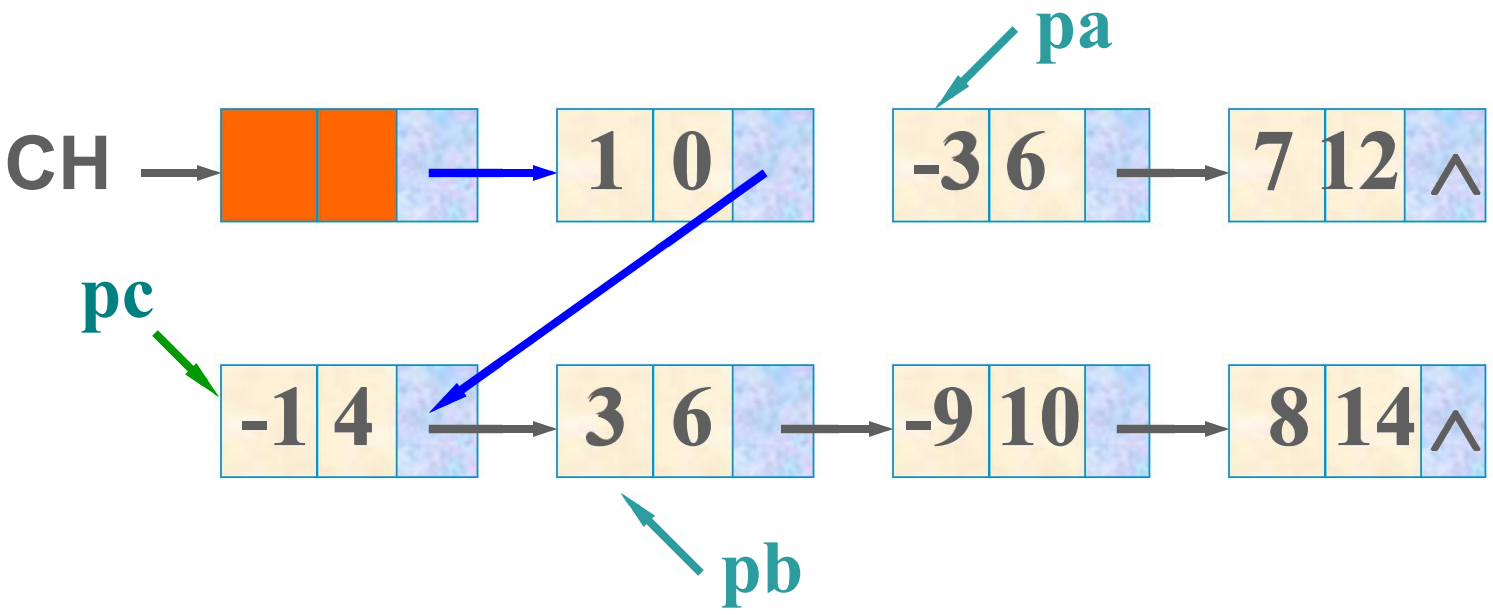
$$BH = -x^4 + 3x^6 - 9x^{10} + 8x^{14}$$

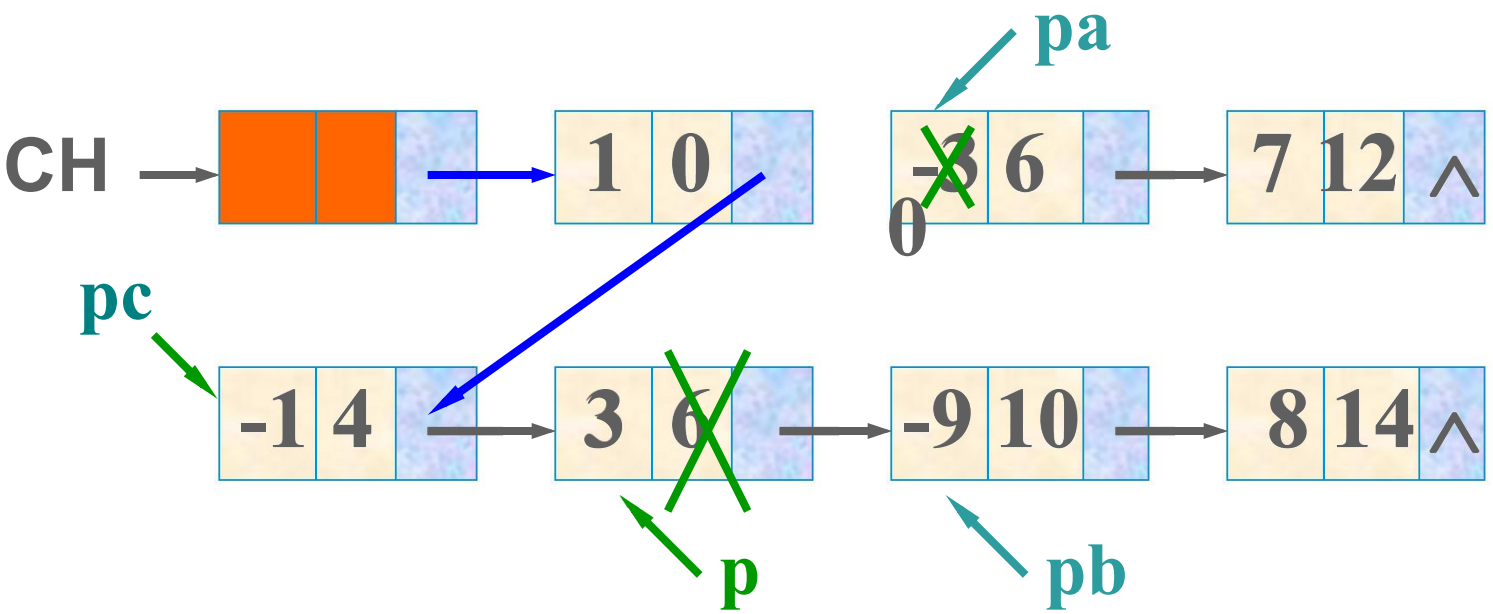


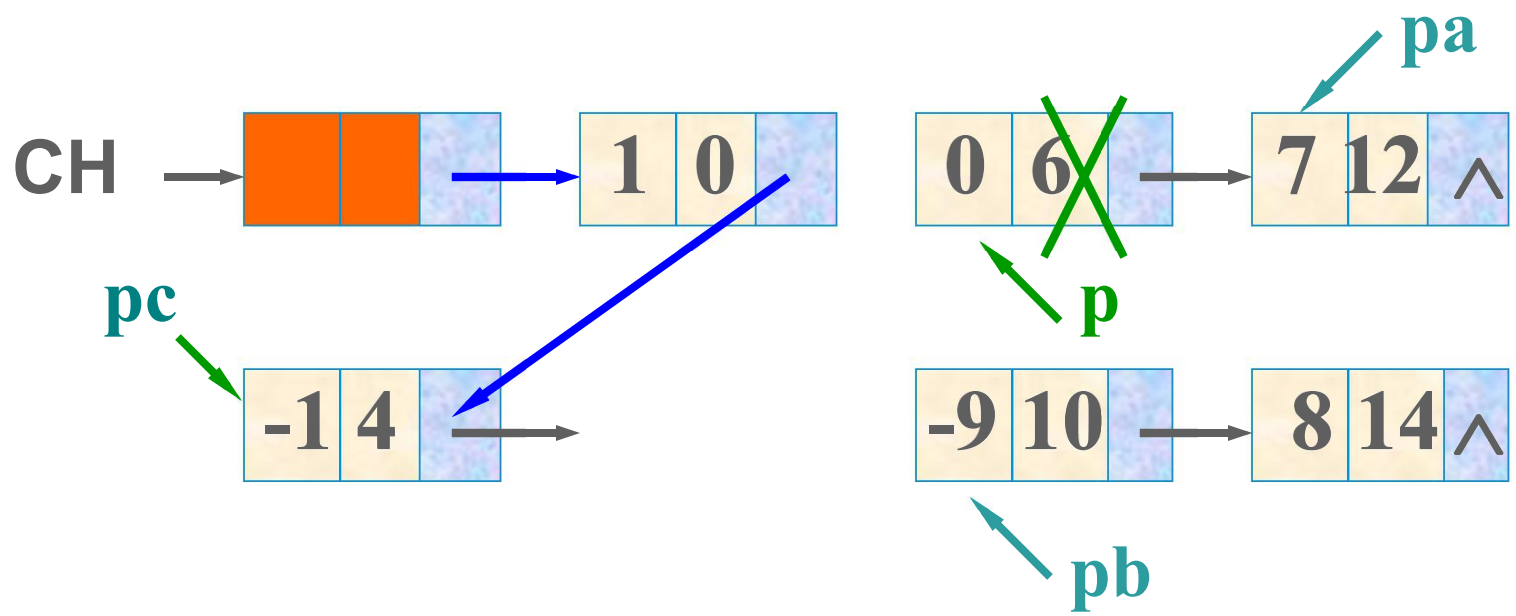
p的用处是为了删除节点

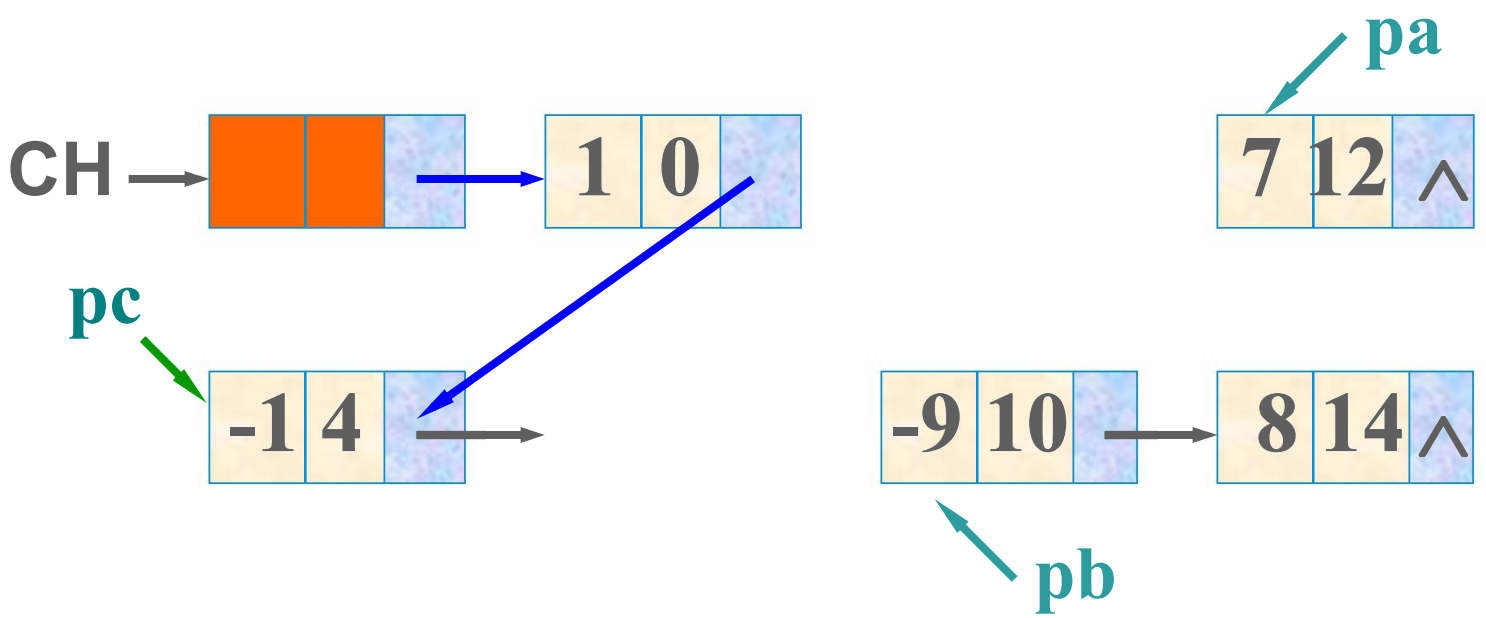


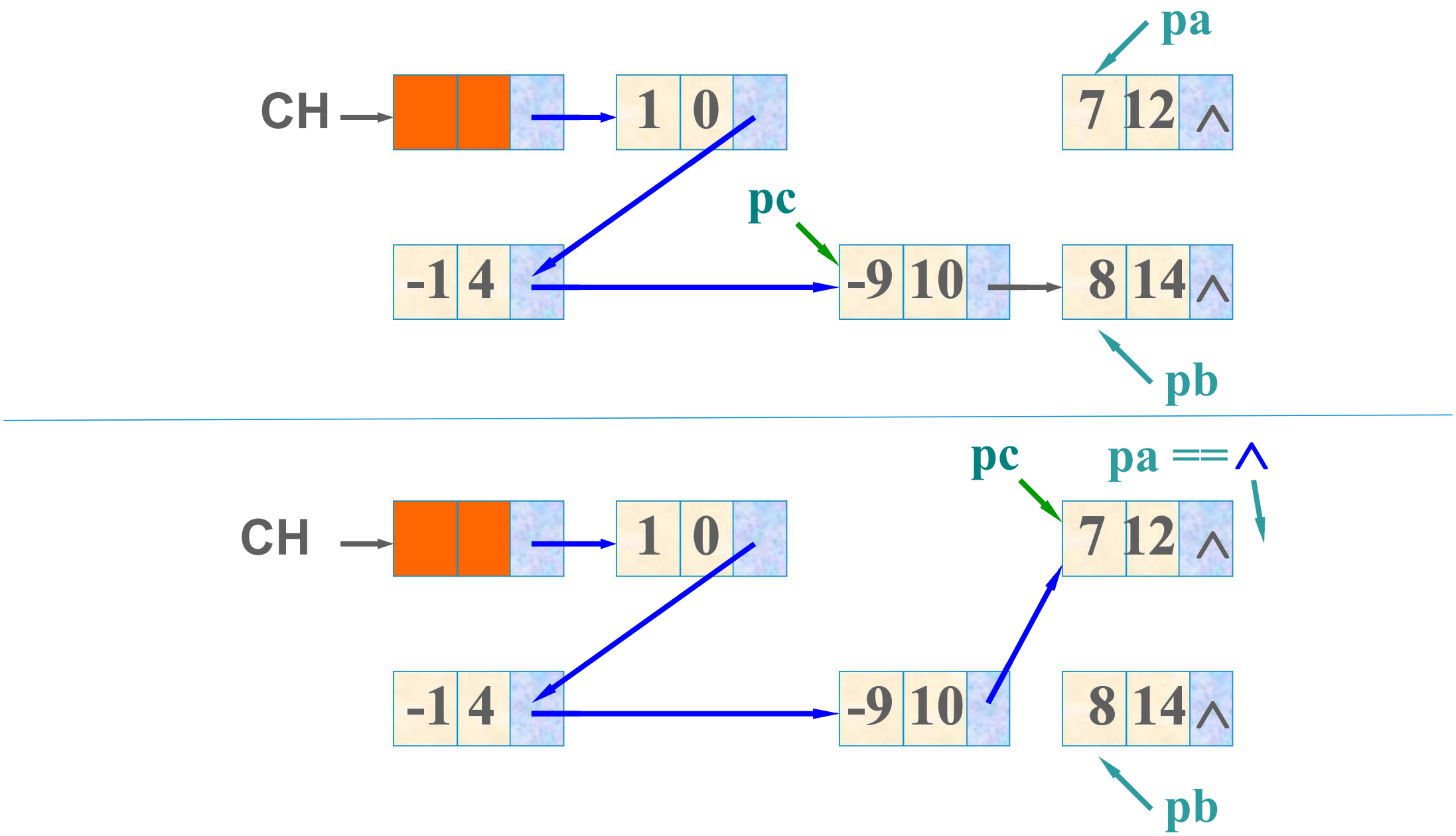


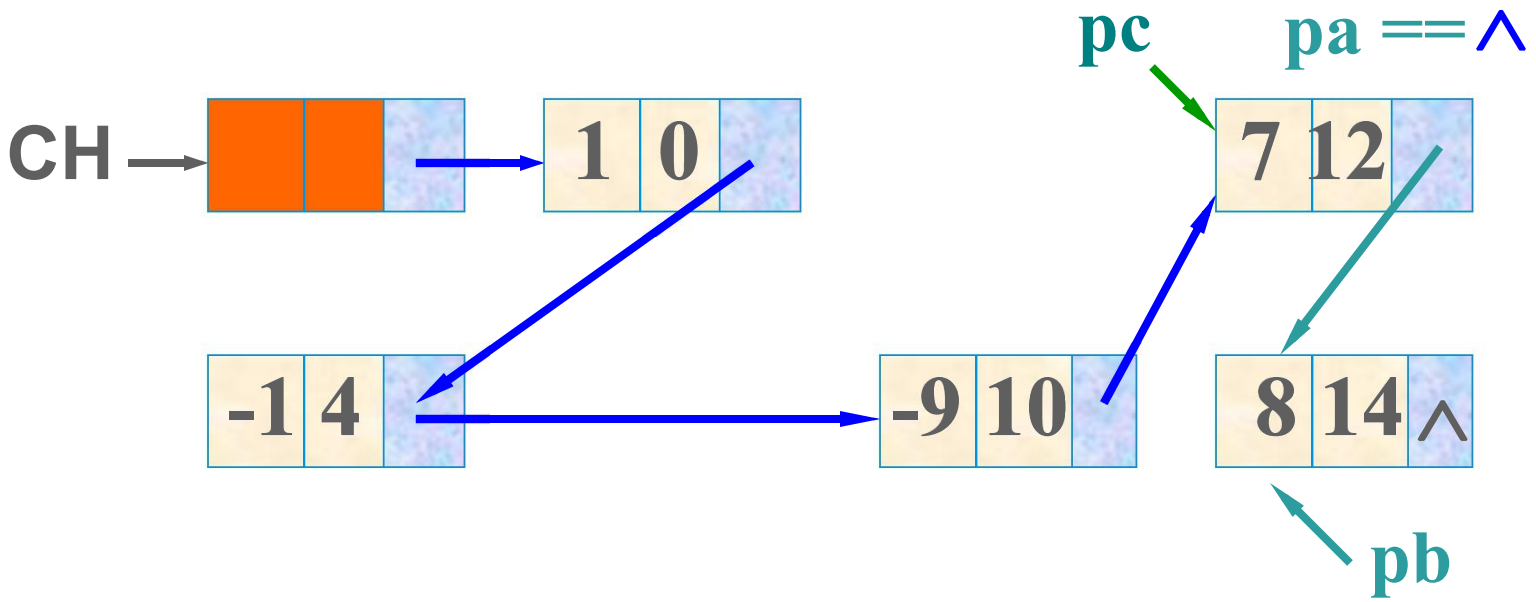

pa
CH
1 0
-3 6
7 12
pc
-1 4
3 6
-9 10
8 14
pb

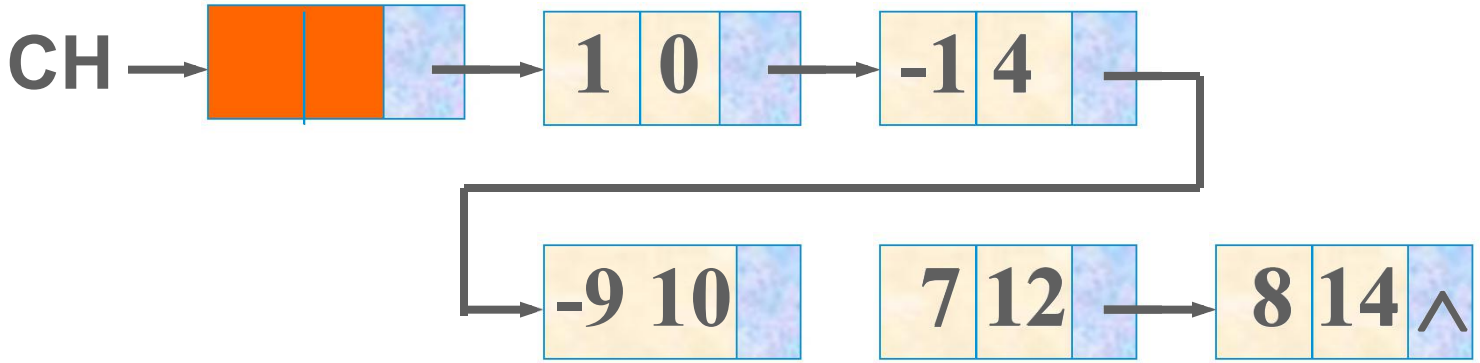

pa
CH
1 0
-3 6
0
7 12
pc
-1 4
3 6
-9 10
8 14
p
pb


CH
1 0
pa
0 6
7 12 ^
pc
p
-1 4
-9 10
8 14 ^
pb


CH
1 0
pa
7 12 ^
pc
-1 4
-9 10
8 14 ^
pb


pa
CH
1 0
7 12 ^
pc
-1 4
-9 10
8 14 ^
pb
pc
pa == ^
CH
1 0
7 12 ^
-1 4
-9 10
8 14 ^
pb


pc
pa == ∧
CH
1 0
7 12
-1 4
-9 10
8 14 ∧
pb


CH
1 0
-1 4
-9 10
7 12
8 14 ∧

```
typedef struct elem_type {        //多项式数据定义
        float coef;                       //系数
        int exp;                  //指数
}elem_type;

typedef struct PolyNode {     //多项式结点定义
        ElemType data;                    //结点数据
        struct PolyNode *next;    //结点指针
} poly_node,*poly_nomial;                 //多项式定义
```

```
void Add ( poly_nomial A, poly_nomial B,
                                poly_nomial *C ) {//伪码
  //两个带头结点的按升幂排列的多项式相加，返回
  //结果多项式链表的表头指针 C，释放 A 和 B 链表
    poly_node *pa, *pb, *pc, *p;   elem_type a, b;
    *C = pc = A;                    //当前结果指针，C返回
    pa = A->next;                  //多项式 A 的检测指针
    pb = B->next;                  //多项式 B 的检测指针
    free(B);                      //删去 B 的表头结点
```

```
while ( pa != NULL && pb != NULL ) {
    a = pa->data;  b = pb->data;      //a，b的data数据取出
    if ( a.exp == b.exp ) {
        a.coef = a.coef + b.coef;     //系数相加
        p = pb;   pb = pb->next;   free(p);
      //指数相等的结点仅保留一个加入结果链
        if ( a.coef ) {  //相加不为零, 加入 C 链
            pa->data = a;   pc->next= pa;  pc = pa;
            pa = pa->next;
        }
        else          //相加为零,该项不要
        { p = pa;  pa = pa->next;  free(p);}
    }
```

```
    else if ( pa->exp > pb->exp ) {
            pc->next= pb;   pc = pb;
             pb = pb->next; }
  else {
            pc->next= pa;  pc = pa;
            pa = pa->link;}
    }   //while结束
    if ( pa != NULL )   pc->next= pa;
    else       pc->next= pb;     //剩余部分链入 C 链
}
```

如果多项式是按照降幂排列，按照既定顺序求值即可。

$$((\ldots((c_n x + c_{n-1})\,x + c_{n-2})\,x + \ldots)\,x + c_1)\,x + c_0$$

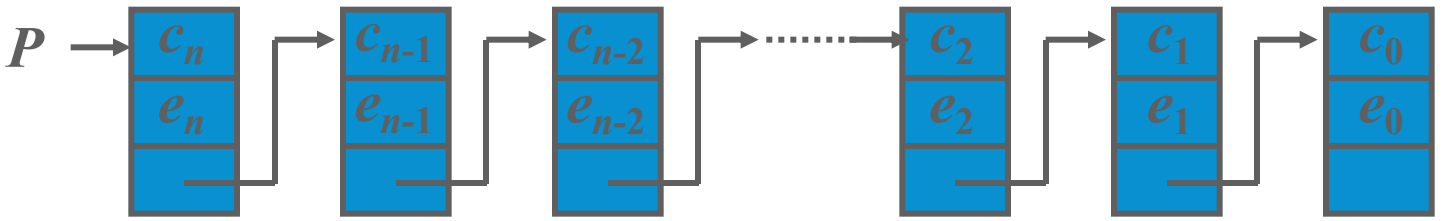
如果多项式有很多零系数项，如

$$25\,x^{101} + 15\,x^{54} + 18\,x^{17} + 2\,x^5 + 6$$

可以考虑改造一下公式，形如：

$$((\ldots((c_n x^{e_n - e_{n-1}} + c_{n-1})x^{e_{n-1} - e_{n-2}} + c_{n-2})x^{e_{n-2} - e_{n-3}} +$$

$$+ \ldots + c_1)x^{e_1 - e_0} + c_0$$



每项需要两个节点数据

然后，设计一个计算 x^i 的函数。

```
float Power ( float x, int i ) {
    float mul = x;
    for ( int j = 1; j < i; j++ ) mul * = x;
    return mul;
}//该函数大多数语言已经内置在库中
```

将指数大小将各项降幂链接，C语义可以直接使用 pow() 函数和上述公式计算多项式的值。#include <math.h>

```
float Evaluate ( poly_nomial pl, float x ) {//伪码
 //计算多项式 pl  在给定 x 时的值
      poly_node *p = pl->next;   //跳过表头
      float rst = p->data.coef;     elem_type a, b;
      if ( p->next== NULL )                  //只有一项
            return  rst*pow(x, p->data.exp);
      while ( p != NULL && p->next!= NULL ) {
            a = p->data;   b = p->next->data;
            rst = rst*Power(x, a.exp-b.exp)+b.coef;
            p = p->next;
      }
return rst;}
```

实际上，多项式求值，采用顺序结构即可(P40)

多项式乘法可以转化为加法（P43）

如果多项式链表按照指数大小将各项升幂链接怎么办？

对一元多项式，用“有序链表”进行表示与存储更好(P41)

- 应用四：约瑟夫问题模拟

据说著名犹太历史学家 Josephus有过以下的故事：在罗马人占领乔塔帕特后，39 个犹太人与Josephus及他的朋友躲到一个洞中，39个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式，41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止。Josephus要他的朋友先假装遵从，他将朋友与自己安排在第16个与第31个位置，于是逃过了这场死亡游戏。

后来衍生了一系列的游戏。

问题可形式化表达为：

N个人围成一圈，从第一个开始报数，第M个将被选择（死亡），直到剩下1（r）个，其余人都已被选择。求选择顺序。

逻辑上是相邻关系，因此可用线性表结构。

以顺序表及循环单链表为例来实现。

顺序表算法策略：

给定位置器；给定报数器;给定死亡计数器；

如果还没死够

向前位置器++，直到第一个没有被选中的位置

报数器++

如果报数器等于给定值，则

当前位置节点改为被选中，输出当前位置

报数器置0，死亡计数器++

可令每个节点data域表示是否被选，0表示没有，1表示有

顺序表算法伪码：

```
Status Josephus(int m, int n, int r)//总人数,定数,余人数
{       sqlist L;       init(&L);creat(L);
        int p = 0, nn=killed=0;//位置,报数器,已死
        while(killed< m-r)
        {       do{p= p%n+1;get_elem(L, p, &e);}
                while(e);
                if(++nn==m)
                {set_elem(L, p, 1); printf("%d",&p); nn=0;  killed++;}
        }
return OK;} //T=O？       m²
```

循环链表算法策略：

也可以仿照顺序表基本一致来实现。下面给出不同思路：

链表初始化，data域为位置序号。指针、计数器等初始化。

如果还可以选择

报数器++

如果报数器等于给定值，则

输出当前节点data域，删除该节点

报数器置0，死亡计数器++

定位下一个节点

前提是要实现循环链表(带头/不带头节点)的next操作。

（性能？）

循环链表算法伪码：

```
Status Josephus(int m, int n, int r)//总人数,定数,余人数
{c_link_list L; init(&L); creat(L);list_node *p=L->next,*q=L；
        int nn=k=0;//                报数器nn,已死k
    while(k<m-r)
    {           if(++nn==n){delete(L,q,&e);printf("%d",e);nn=0;k++;}
                else   q=p;
                p=next(L,q);
    }
return OK;
} //T=O？ mn     ？ 如何能同时输出每个人的姓名？
```

- 小节-----线性结构之线性表
  - 线性结构逻辑推理过程
  - 顺序表---利用一维数组实现
  - 链表---利用指针/游标实现（带头/不带头）
    - 单链表
    - 双向链表
    - 循环链表
    - 双向循环链表
    - 静态链表---数组+游标

ADT、逻辑结构、存储结构、操作、应用：

逻辑相邻

物理相邻

是否可随机存取

插入 删除 定位 移动元素 动态性

应用：

具体事物---->抽象表示

视需求选择结构与设计操作及算法

不同的时间复杂度

上机与作业：

0、单链表其余操作的实现（下午上机）

1、顺序表实现集合合并（下午上机）

2、链表实现有序表合并（下午上机）

3、顺序表、链表实现多项式相加（上机+作业）

自行设计主函数进行0-3的各项验证。

0-3作业上交时间，下周一凌晨4点。