CHAPTER 3

Lists, Stacks, and Queues

§1 Abstract Data Type (ADT)

【Definition】 Data Type = { Objects } ∪ { Operations }

〖Example〗 int = { 0, ±1, ±2, · · ·, INT_MAX, INT_MIN } ∪ { +, −, ×, ÷, %, · · · }

【Definition】 An Abstract Data Type (ADT) is a data type that is organized in such a way that the specification on the objects and specification of the operations on the objects are separated from the representation of the objects and the implementation on the operations.

§2 The List ADT

❖ **ADT:**

Objects: (item$_0$, item$_1$, ··· , item$_{N-1}$)

Operations:

- Finding the length, N, of a list.
- Printing all the items in a list.
- Making an empty list.
- Finding the k-th item from a list, $0 \le k < N$.
- Inserting a new item after the k-th item of a list, $0 \le k < N$.
- Deleting an item from a list.
- Finding next of the current item from a list.
- Finding previous of the current item from a list.

Why after?

1. Simple Array implementation of Lists

array[i] = item$_i$

Sequential mapping

Address	Content
……	……
array+i	item$_i$
array+i+1	item$_{i+1}$
……	……

MaxSize has to be estimated.

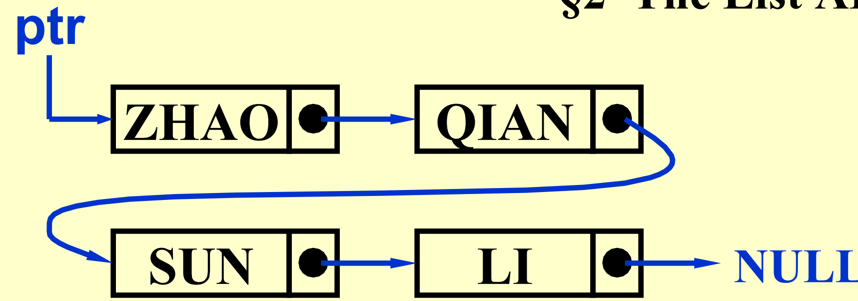
Find_Kth takes O(1) time.

Insertion and **Deletion** not only take O(N) time, but also involve a lot of data movements which takes time.

2. Linked Lists

Address	Data	Pointer
0010	SUN	1011
0011	QIAN	0010
0110	ZHAO	0011
1011	LI	NULL

Head pointer ptr = 0110



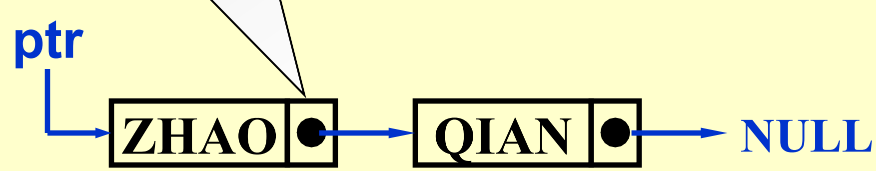
Initialization:

```
typedef  struct  list_node  *lis...
typedef  struct  list_node  {
     char        data [ 4 ] ;
     list_ptr    next ;
} ;
list_ptr   ptr ;
```

To link 'ZHAO' and 'QIAN':

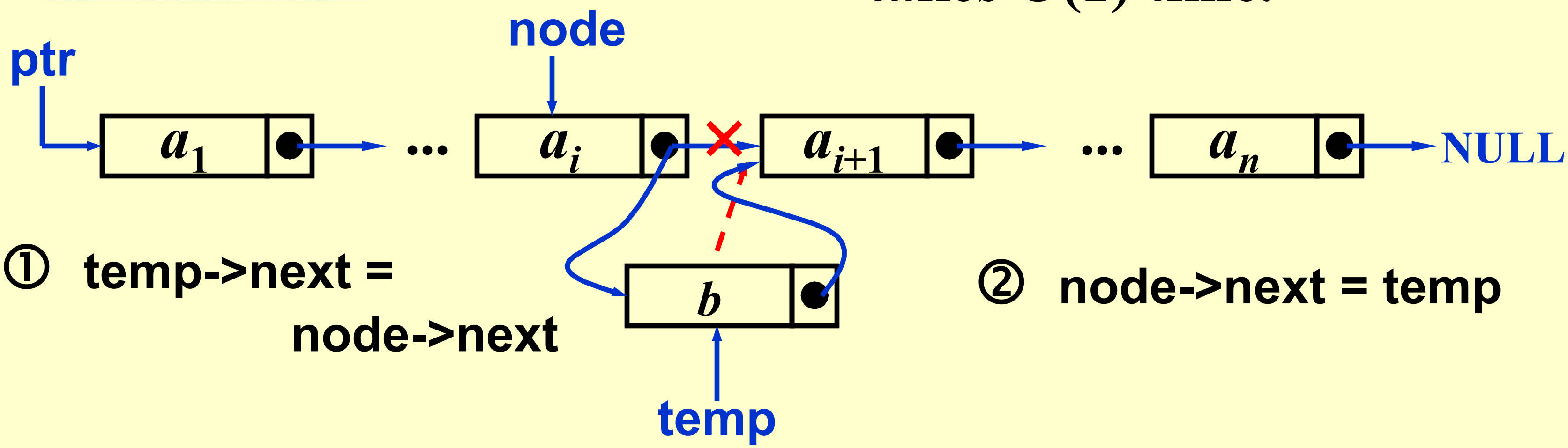
```
...N2 ;
...izeof(struct list_node));
...of(struct list_node));
...
...1 ;
ptr = ...
```

Locations of the nodes may change on different runs.



Insertion

takes O(1) time.



① temp->next = node->next

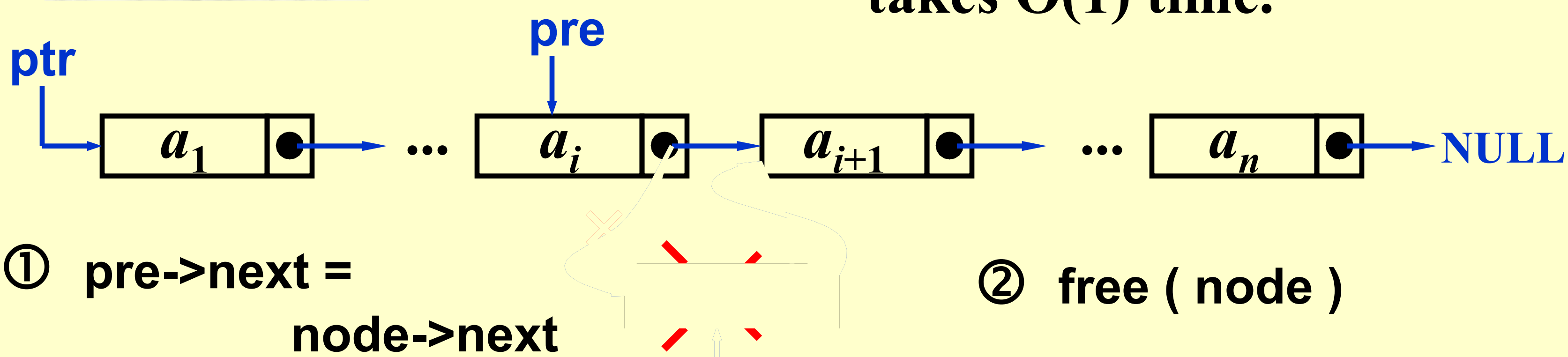
② node->next = temp

Question: What will happen if the order of the two steps is reversed?

Question: How can we insert a new first item?

Deletion

takes O(1) time.



① pre->next = node->next

② free (node)

Question: How can we delete the first node from a list?

Answer: We can add a dummy head node to a list.

Read programs in Figures 3.6-3.15 for detailed implementations of operations.

Doubly Linked Circular Lists

```
typedef struct node *node_ptr ;
typedef struct node {
    node_ptr llink;
    element   item;
    node_ptr rlink;
} ;
```

…n … Then I'll have to

…n the 1st node again.

…y, why do I wanna

…de?

…node

…ue.

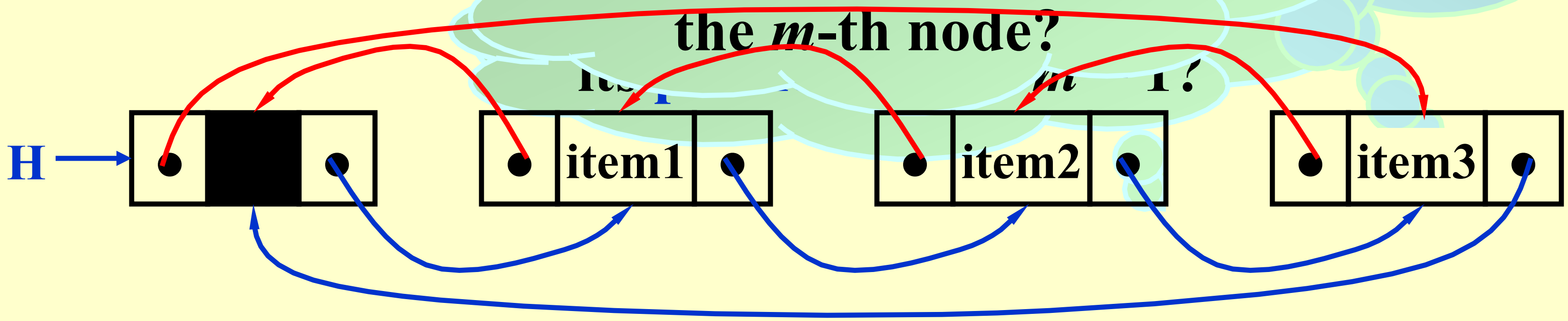
ptr = ptr->llink->rlink

= ptr->rlink->llink

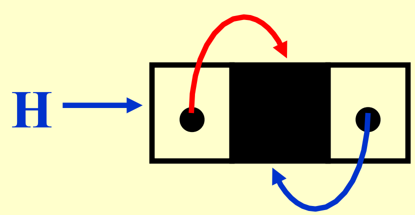
Why do you ask me? :-)

Maybe you wanna delete the *m*-th node?

A doubly linked circular list with head node:



An empty list :



Two Applications

✵ The Polynomial ADT

Objects : $P(x) = a_1 x^{e1} + \cdots + a_n x^{en}$; a set of ordered pairs of $< e_i , a_i >$ where a_i is the coefficient and e_i is the exponent. e_i are nonnegative integers.

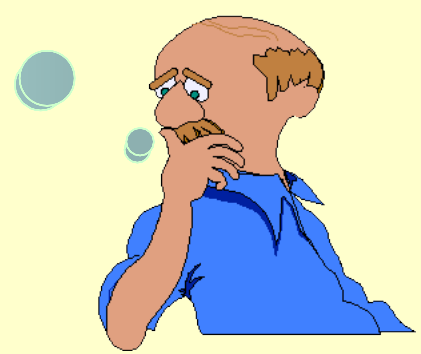
Operations:

- Finding degree, $\max \{ e_i \}$, of a polynomial.
- Addition of two polynomials.
- Subtraction between two polynomials.
- Multiplication of two polynomials.
- Differentiation of a polynomial.

【Representation 1】

```
typedef struct {
    int     CoeffArray [ MaxDegree + 1 ] ;
    int     HighPower;
} *Polynomial ;
```

Try to apply MultPolynomial (p.47)
On $P_1(x) = 10x^{1000}+5x^{14}+1$ and
$P_2(x) = 3x^{1990}-2x^{1492}+11x+5$
-- now do you see my point?

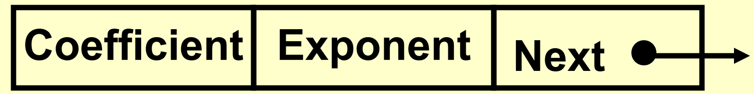


【Representation 2】

Given: $A(x) = a_{m-1}x^{e_{m-1}} + \cdots + a_0 x^{e_0}$

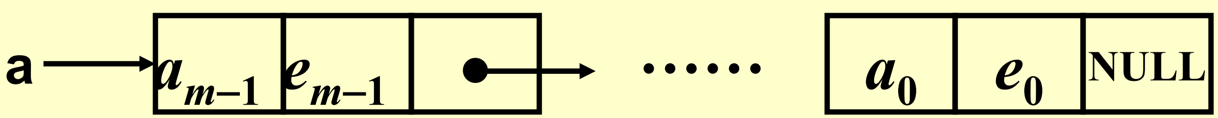
where $e_{m-1} > e_{m-2} > \cdots > e_0 \geq 0$ **and** $a_i \neq 0$ **for** $i = 0, 1, \cdots, m-1.$

We represent each term as a node



Declaration:

```c
typedef struct poly_node *poly_ptr;
struct poly_node {
    int      Coefficient ;  /* assume coefficients are integers */
    int      Exponent;
    poly_ptr Next ;
} ;
typedef poly_ptr a ;    /* nodes sorted by exponent */
```



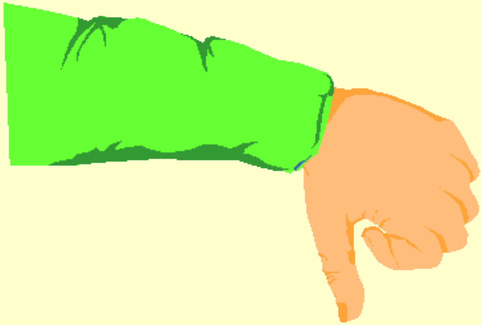
❈ Multilists

〖Example〗 Suppose that we have 40,000 students and 2,500 courses. Print the students' name list for each course, and print the registered classes' list for each student.

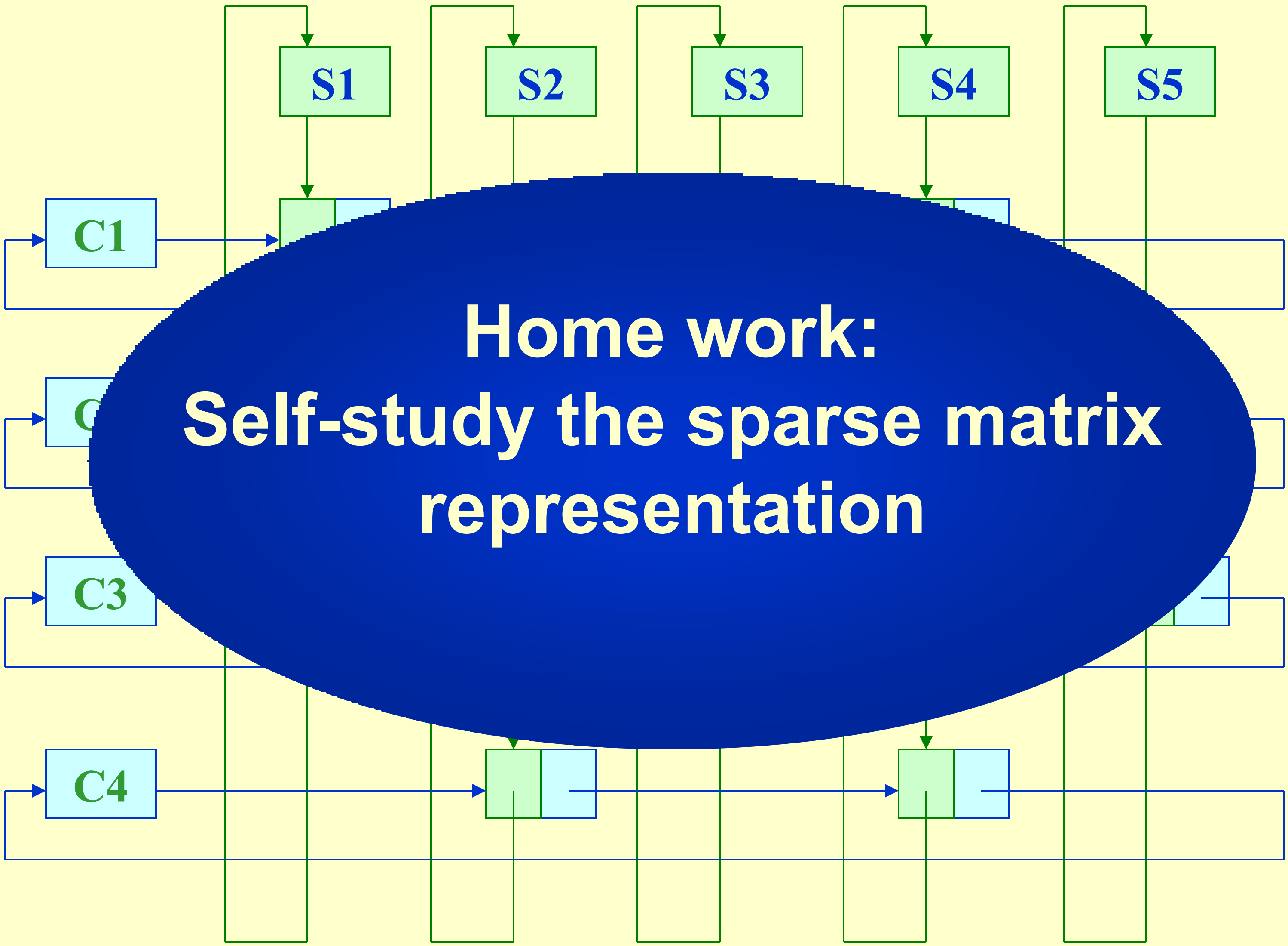
【Representation 1】

```
int  Array[40000][2500];
```

$$\text{Array}[i][j] = \begin{cases} 1 & \text{if student } i \text{ is registered for course } j \\ 0 & \text{otherwise} \end{cases}$$



【Representation 2】

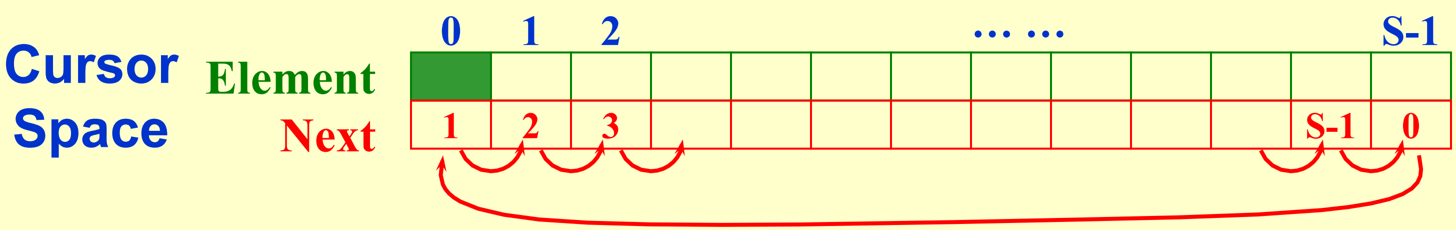


3. Cursor Implementation of Linked Lists (no pointer)

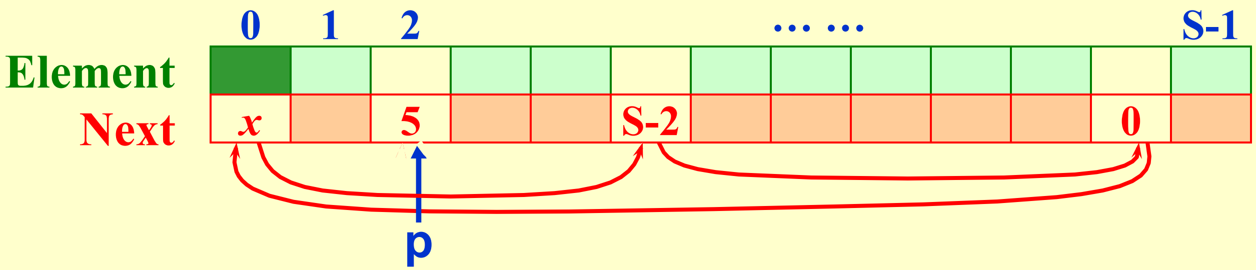
☞ Features that a linked list must have:

a) The data are stored in a collection of structures. Each structure contains data and a pointer to the next structure.

b) A new structure can be obtained from the system's global memory by a call to malloc and released by a call to free.

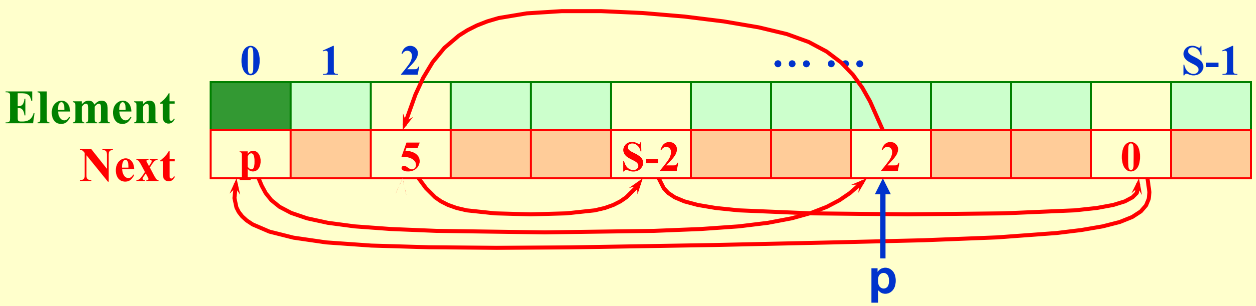


Note: The interface for the cursor implementation (given in Figure 3.27 on p. 52) is identical to the pointer implementation (given in Figure 3.6 on p. 40).



malloc: p = CursorSpace[0].Next ;

CursorSpace[0].Next = CursorSpace[p].Next ;



free(p): CursorSpace[p].Next = CursorSpace[0].Next ;

CursorSpace[0].Next = p ;

Read operation implementations given in Figures 3.31-3.35

Note: The cursor implementation is usually significantly **faster** because of the lack of memory management routines.