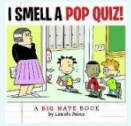
ECE 20010 Data Structures

Chapter 4

- *singly linked list*
- *linked stacks and queues*
- *polynomials (and sparse matrices)*
- *doubly linked list*



Chapter 4 – Linked lists

4.1 Singly linked lists and chains

Let's suppose that we have a list of words to store in an array in alphabetical order. The available storage is an array of strings and an array of integers only. The words are given in a random order.

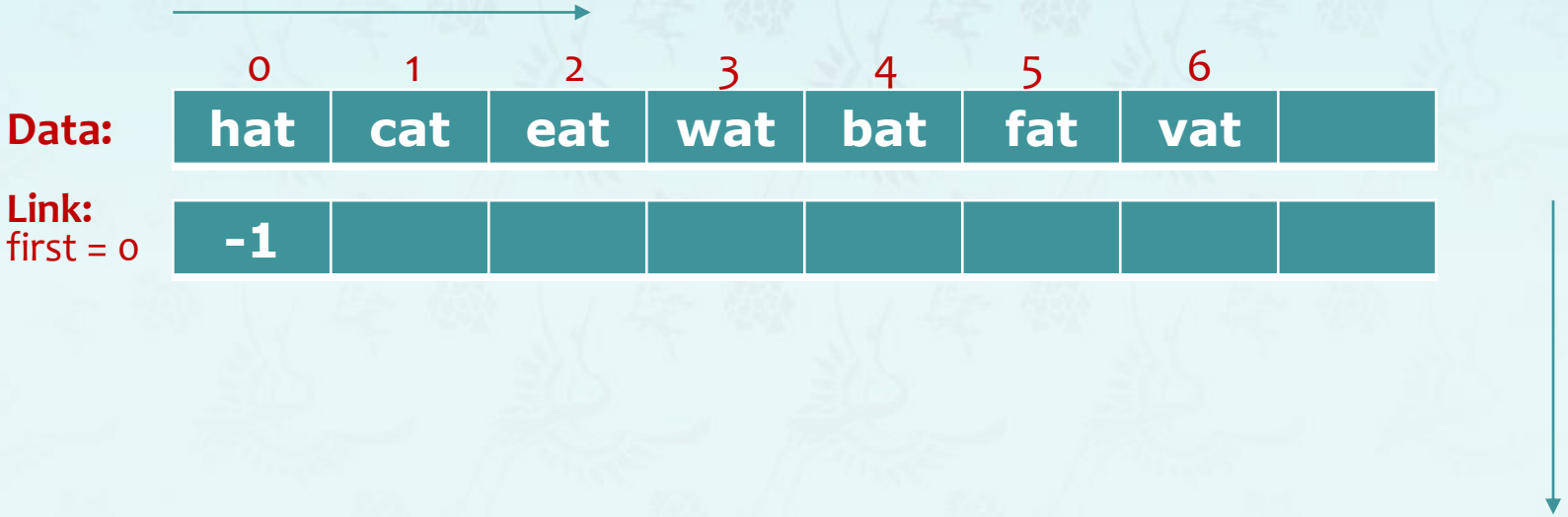
Example. HAT, CAT, EAT, WAT, BAT, FAT, VAT

Chapter 4 – Linked lists

4.1 Singly linked lists and chains

Let's suppose that we have a list of words to store in an array in alphabetical order. The available storage is an array of strings and an array of integers only. The words are given in a random order.

Example. HAT, CAT, EAT, WAT, BAT, FAT, VAT



Chapter 4 – Linked lists

4.1 Singly linked lists and chains

Let's suppose that we have a list of words to store in an array in alphabetical order. The available storage is an array of strings and an array of integers only. The words are given in a random order.

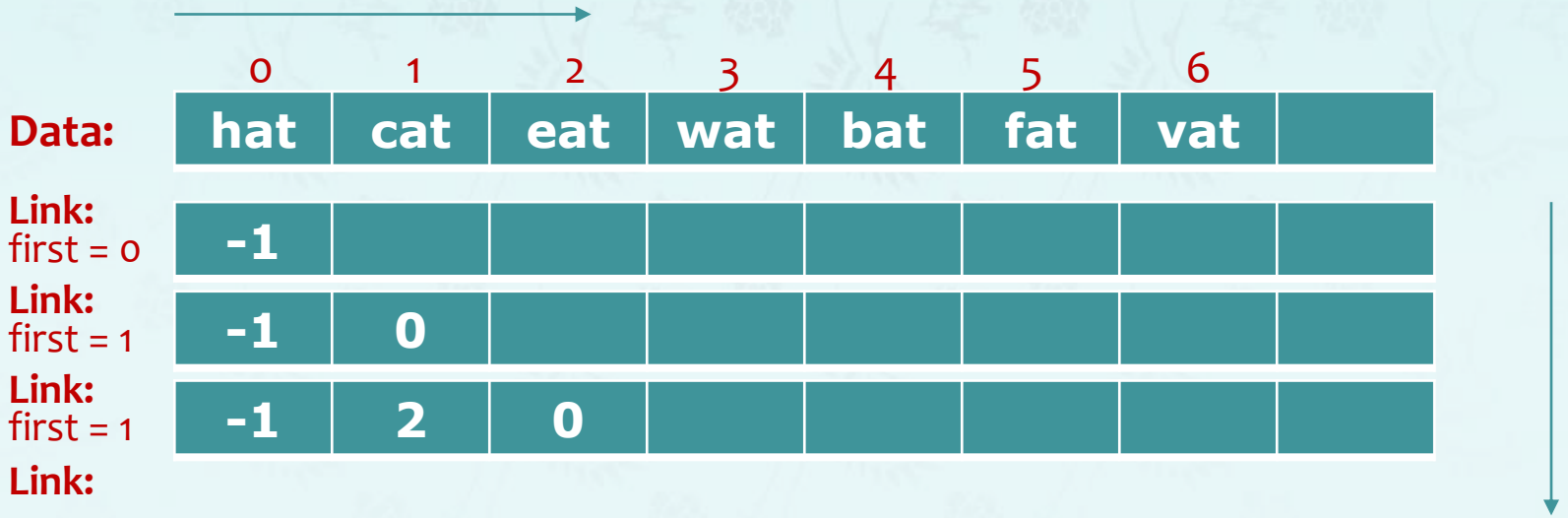
Example. HAT, CAT, EAT, WAT, BAT, FAT, VAT

Chapter 4 – Linked lists

4.1 Singly linked lists and chains

Let's suppose that we have a list of words to store in an array in alphabetical order. The available storage is an array of strings and an array of integers only. The words are given in a random order.

Example. HAT, CAT, EAT, WAT, BAT, FAT, VAT

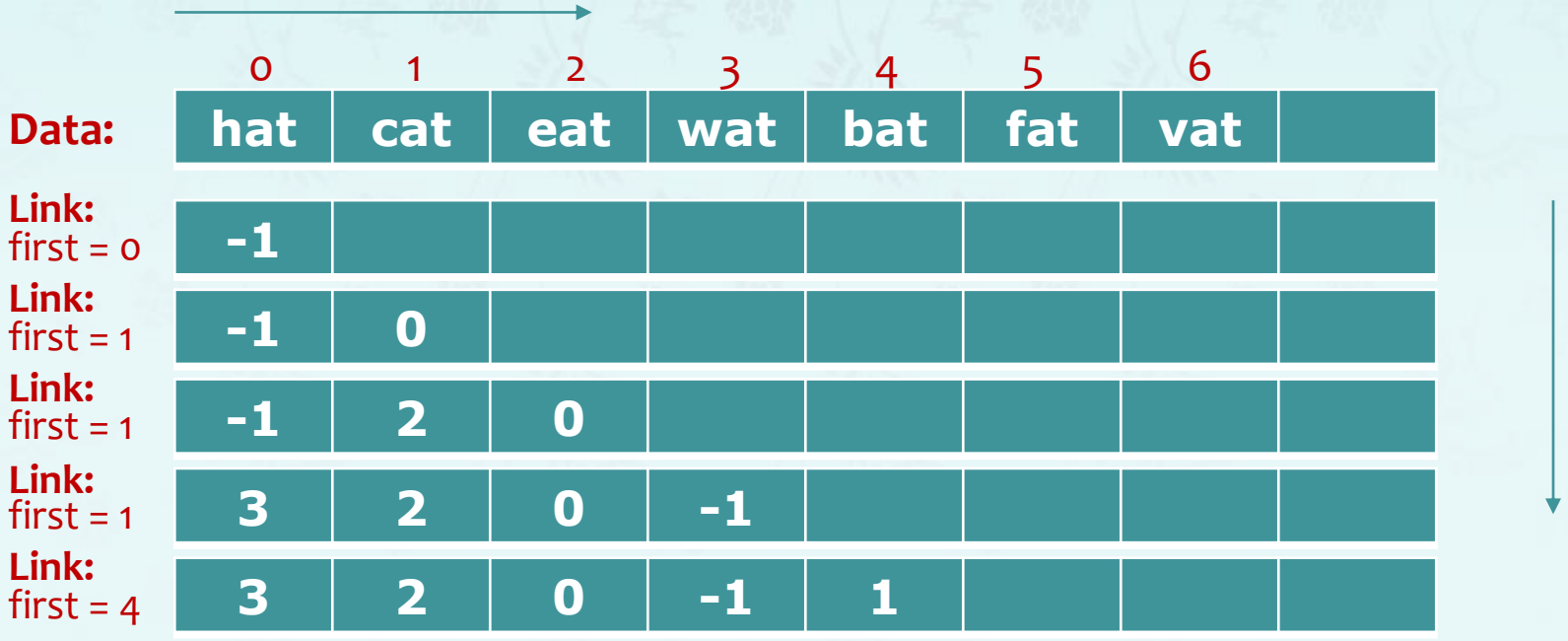


Chapter 4 – Linked lists

4.1 Singly linked lists and chains

Let's suppose that we have a list of words to store in an array in alphabetical order. The available storage is an array of strings and an array of integers only. The words are given in a random order.

Example. HAT, CAT, EAT, WAT, BAT, FAT, VAT



Chapter 4 – Linked lists

4.1 Singly linked lists and chains

Let's suppose that we have a list of words to store in an array in alphabetical order. The available storage is an array of strings and an array of integers only. The words are given in a random order.

Example. HAT, CAT, EAT, WAT, BAT, FAT, VAT

Chapter 4 – Linked lists

4.1 Singly linked lists and chains

Let's suppose that we have a list of words to store in an array in alphabetical order. The available storage is an array of strings and an array of integers only. The words are given in a random order.

Example. HAT, CAT, EAT, WAT, BAT, FAT, VAT

Chapter 4 – Linked lists

4.1 Singly linked lists and chains

Let's suppose that we have a list of words to store in an array in alphabetical order. The available storage is an array of strings and an array of integers only. The words are given in a random order.

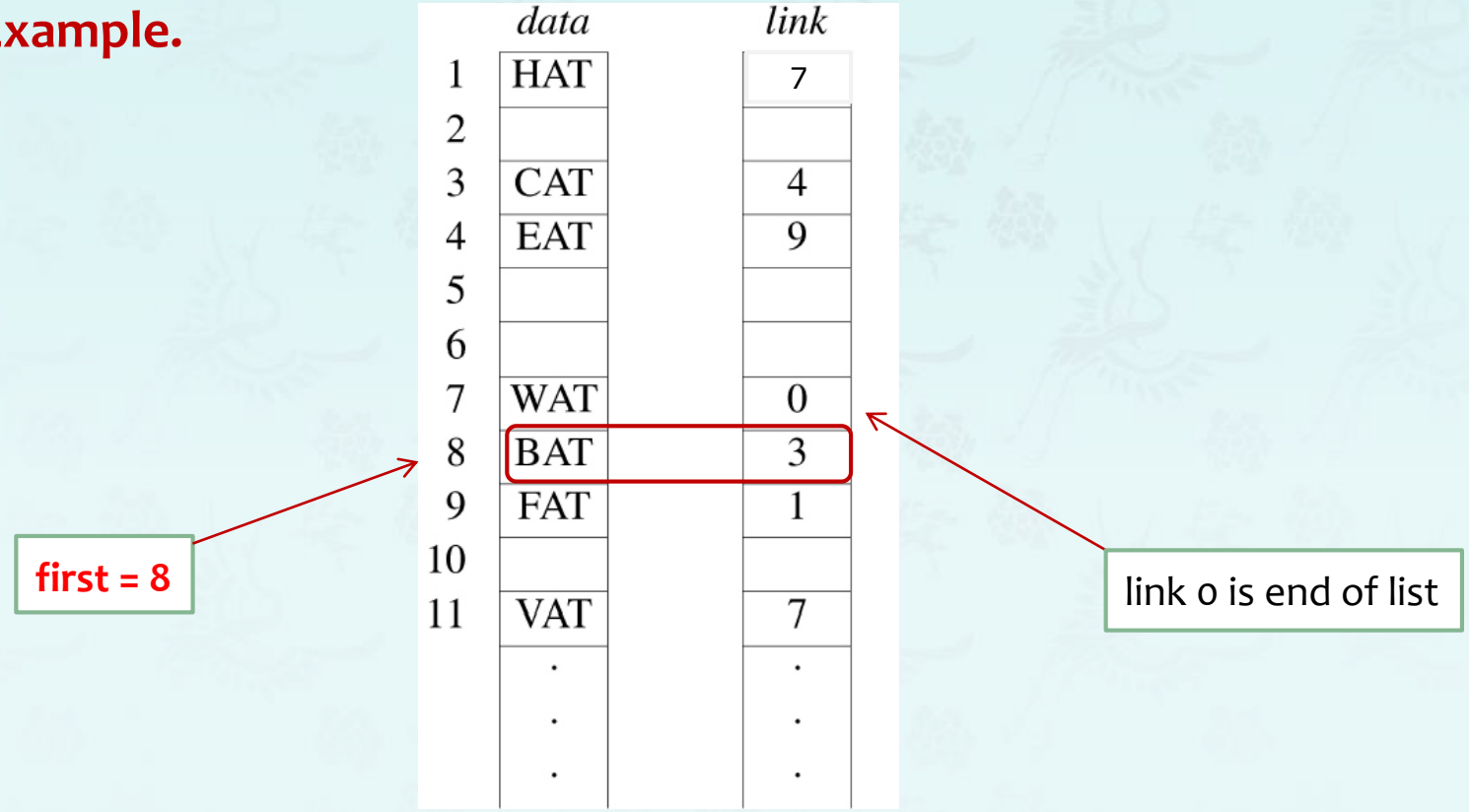
Example. HAT, CAT, EAT, WAT, BAT, FAT, VAT

How about adding act ?

4.1 Singly linked lists and chains

Two types of data representations: sequential vs linked

Example.



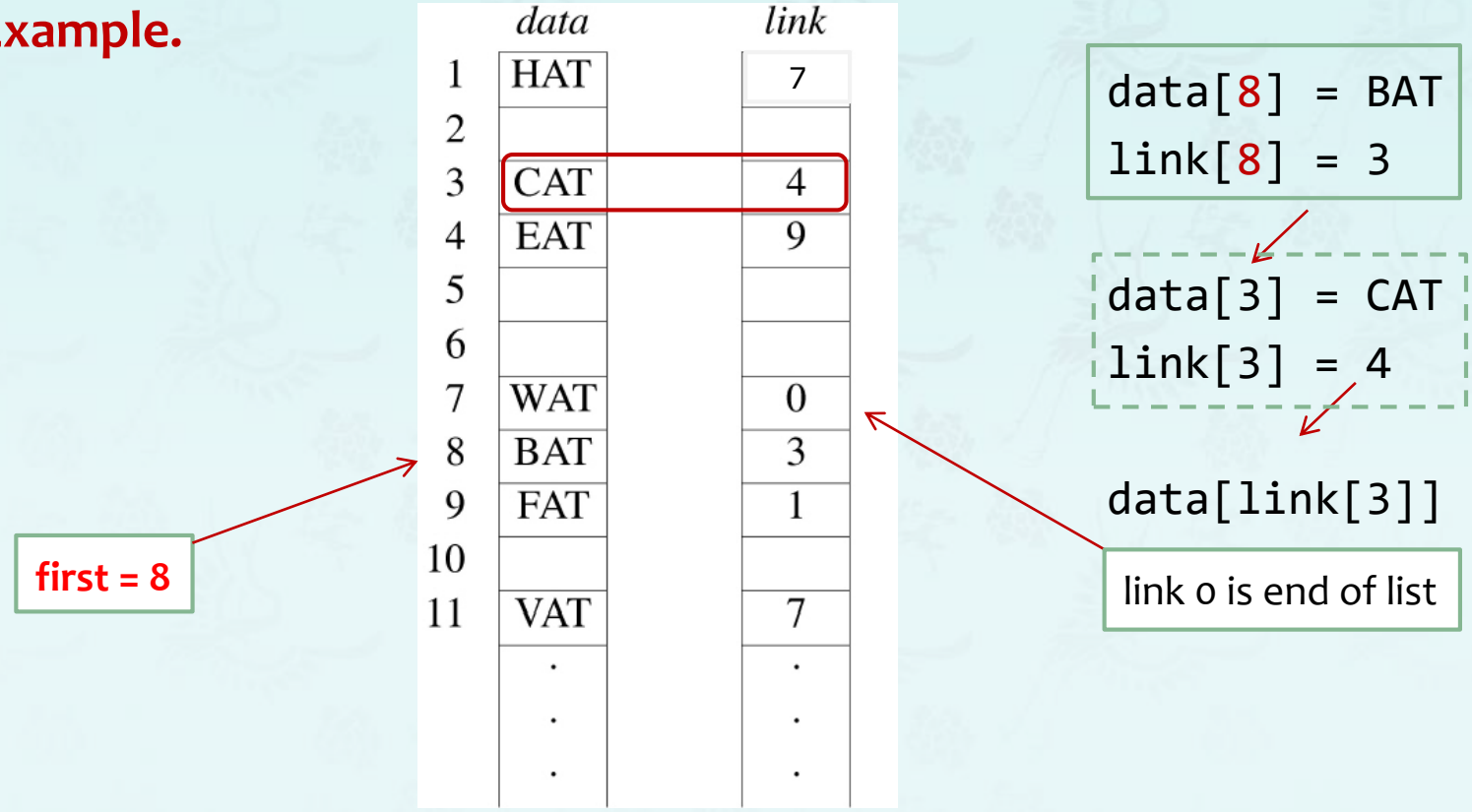
Q: How is the words listed above? (BAT, CAT, ….

BAT, CAT, EAT, FAT, HAT, WAT

4.1 Singly linked lists and chains

Two types of data representations: sequential vs linked

Example.



Q: How is the words listed above? (BAT, CAT, ….

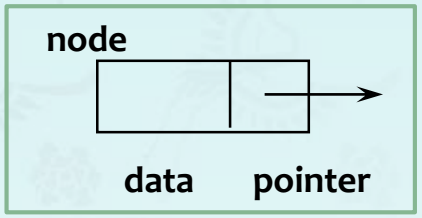
BAT, CAT, EAT, FAT, HAT, WAT

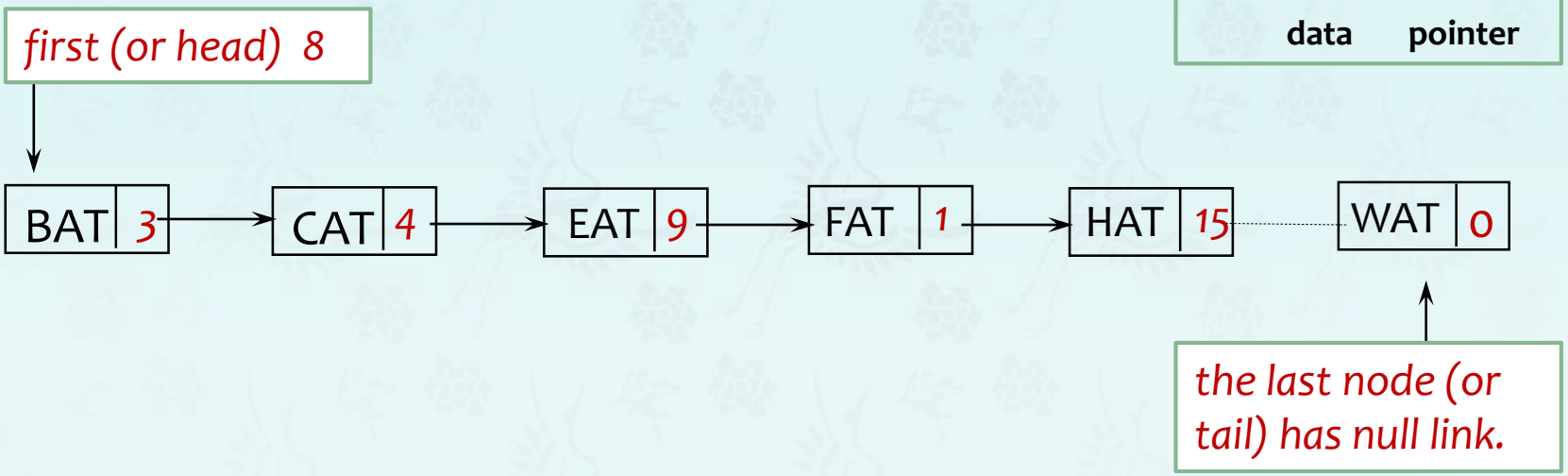
Chapter 4 – Linked lists

4.1 Singly linked lists and chains

Two types of data representations: sequential vs linked

Example. a common way to draw a linked list

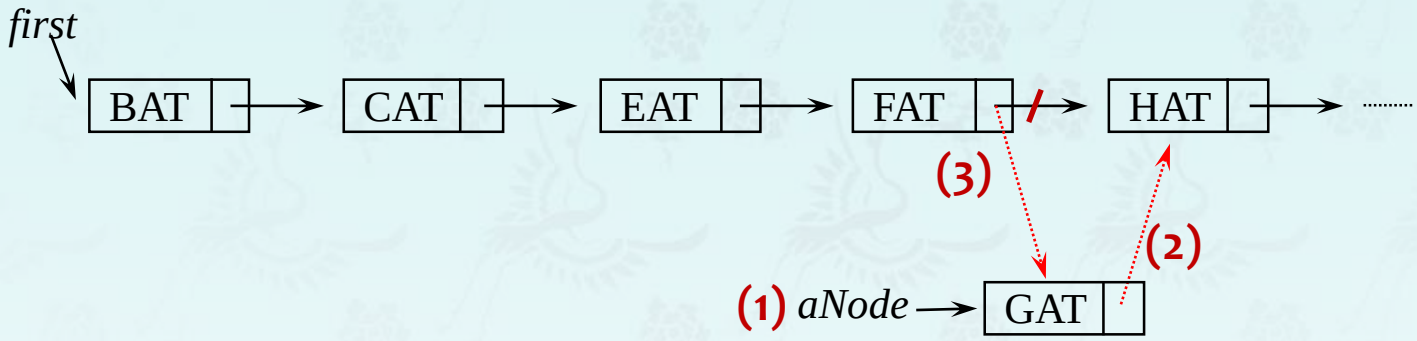




Q: How do you **insert** or **delete** a node in a middle of the ordered list above?
Exercise: How about deleting CAT?

4.1 Singly linked lists and chains

Q: How do you **insert** a node **GAT** after FAT and before HAT in the list?



(1) **Create** a node *aNode* and set its data field to ***GAT***

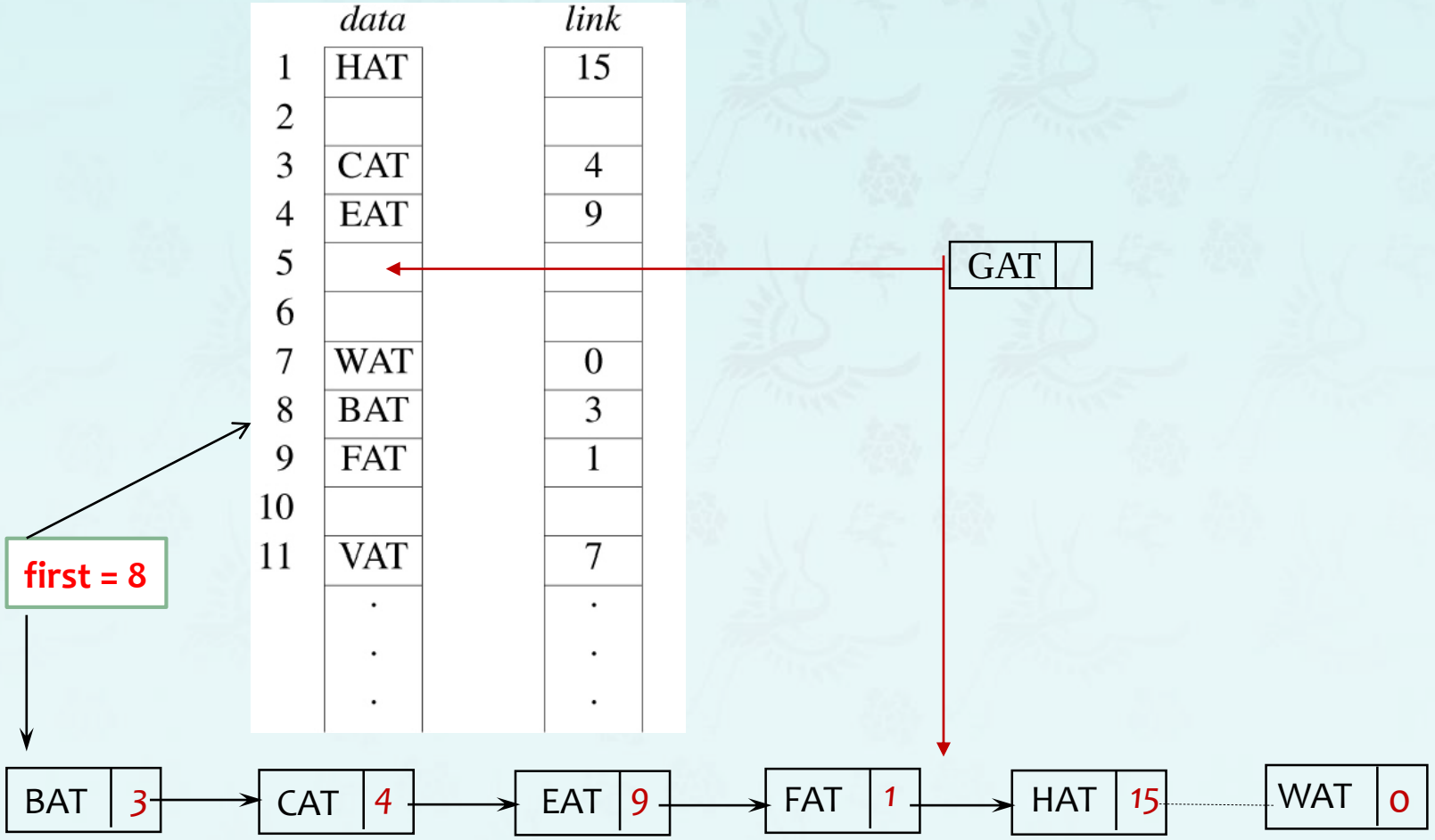
(2) Set the node **GAT**'s link field to point the node ***HAT***.

(3) Set the node **FAT**'s link field to point the node **GAT**.

Chapter 4 – Linked lists

4.1 Singly linked lists and chains

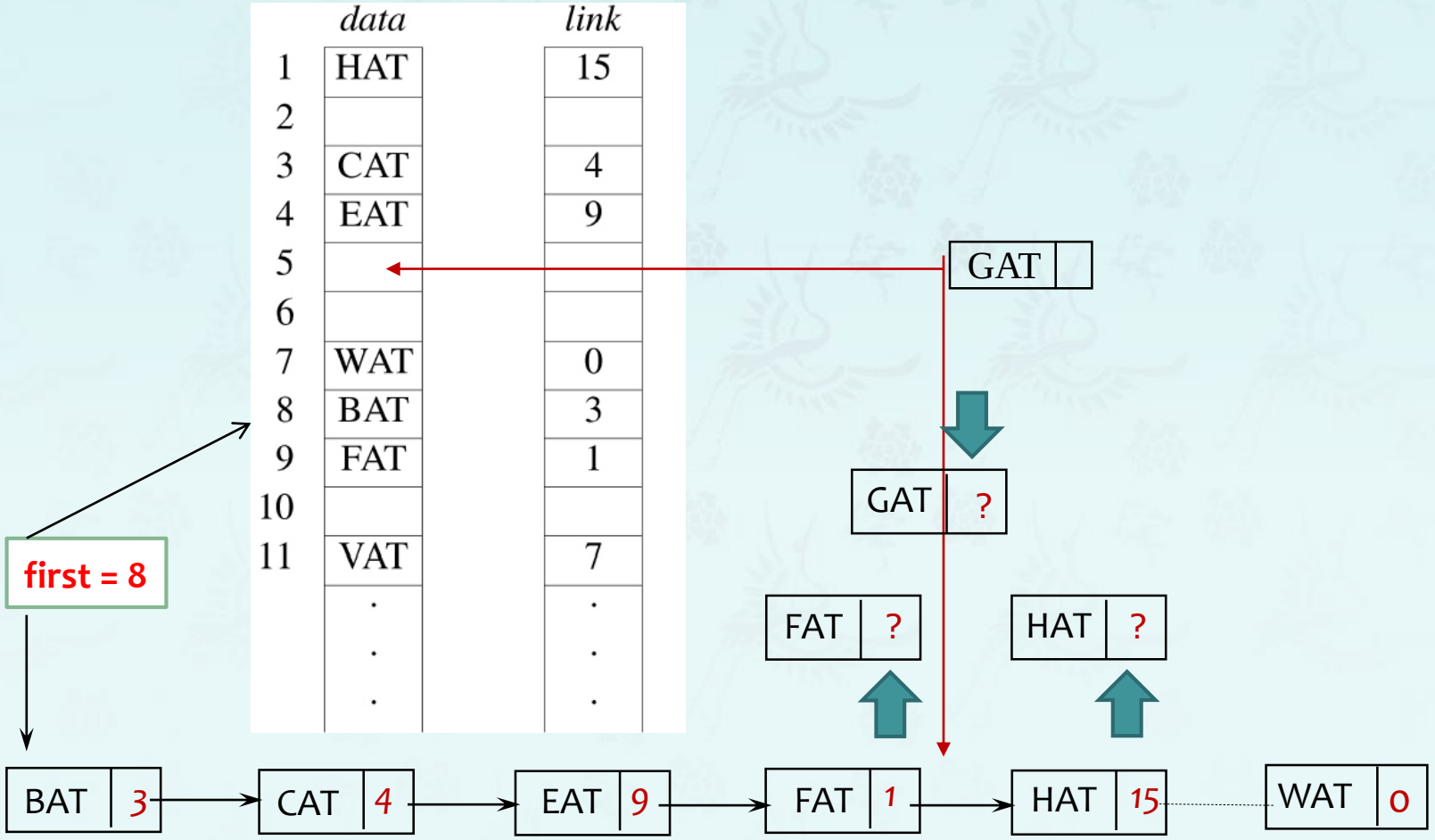
Q: How do you **insert** a node **GAT** after **FAT** and before **HAT** in the list?



Chapter 4 – Linked lists

4.1 Singly linked lists and chains

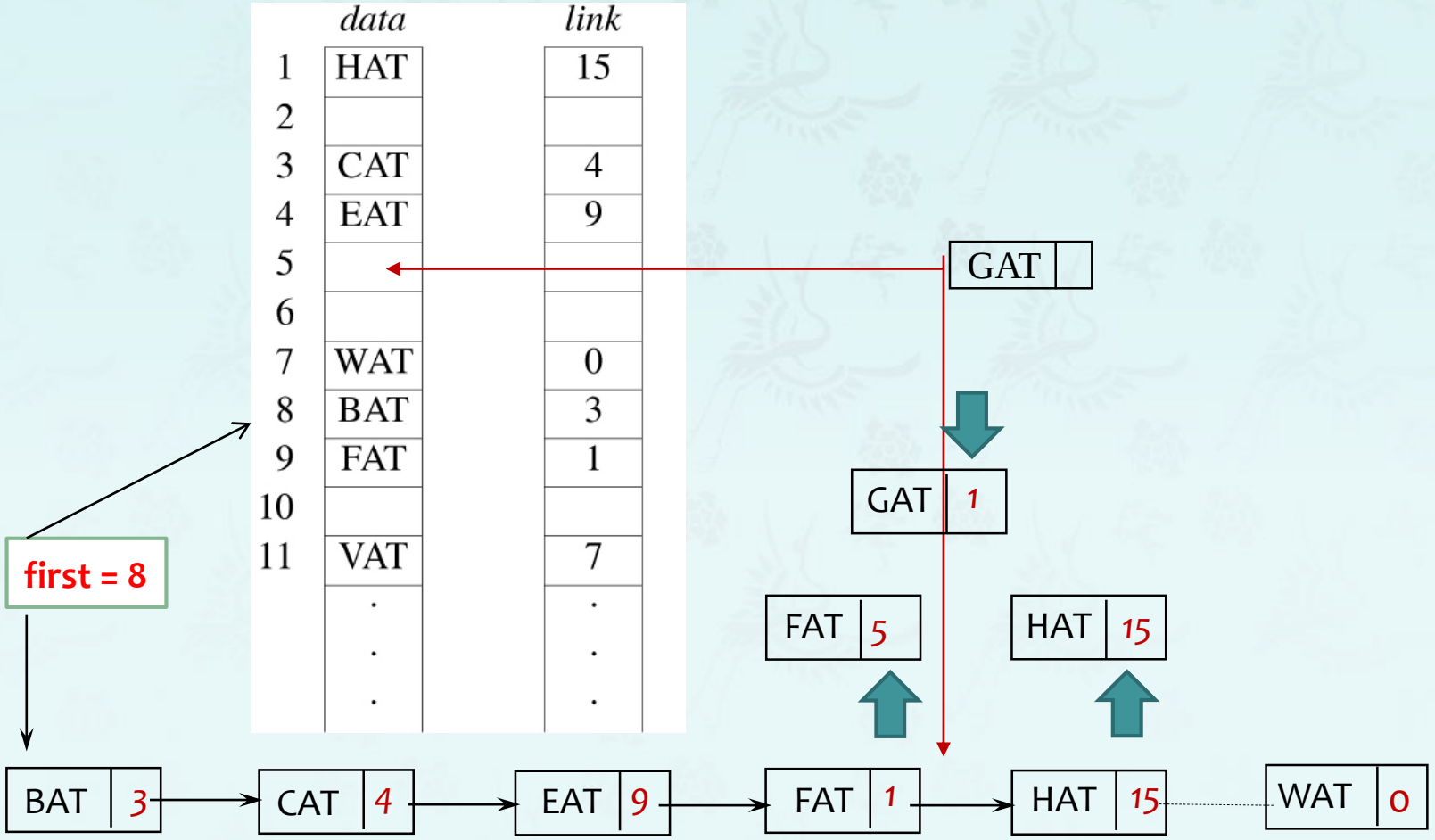
Q: How do you **insert** a node **GAT** after **FAT** and before **HAT** in the list?



Chapter 4 – Linked lists

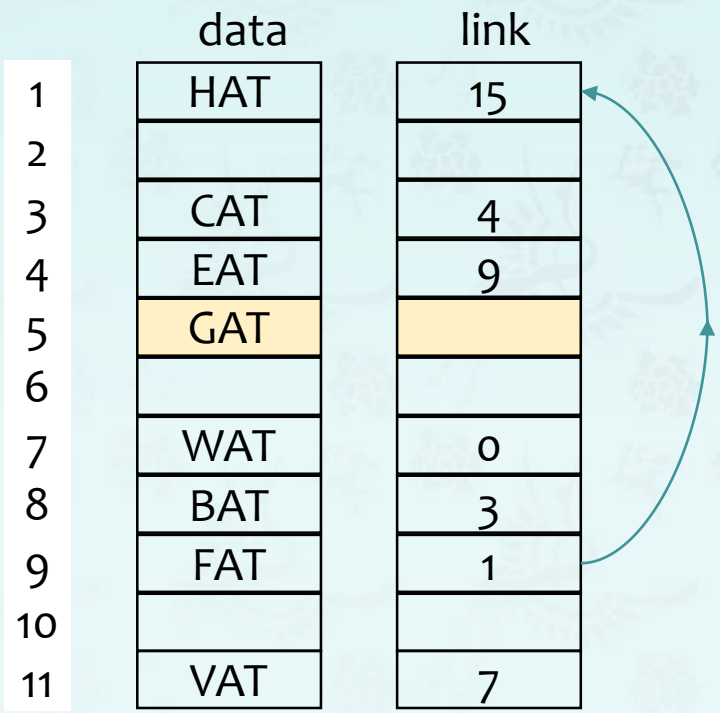
4.1 Singly linked lists and chains

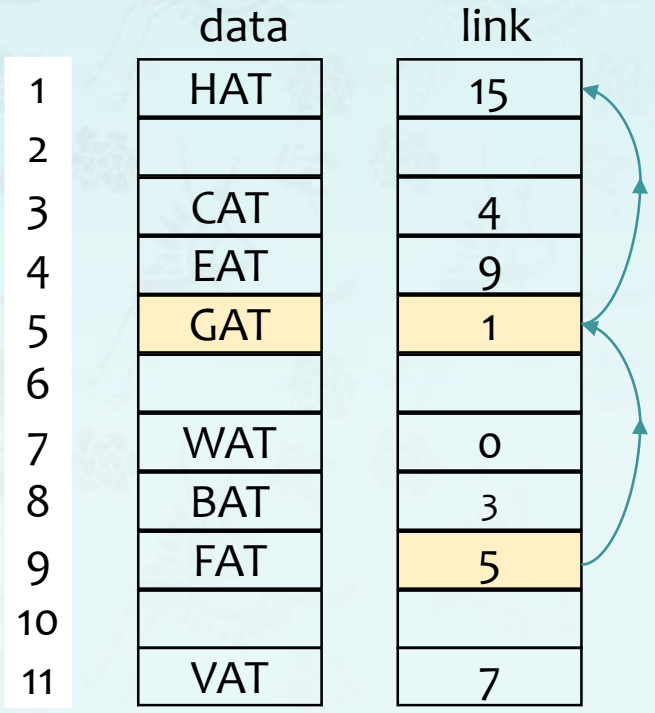
Q: How do you **insert** a node **GAT** after **FAT** and before **HAT** in the list?



4.1 Singly linked lists and chains

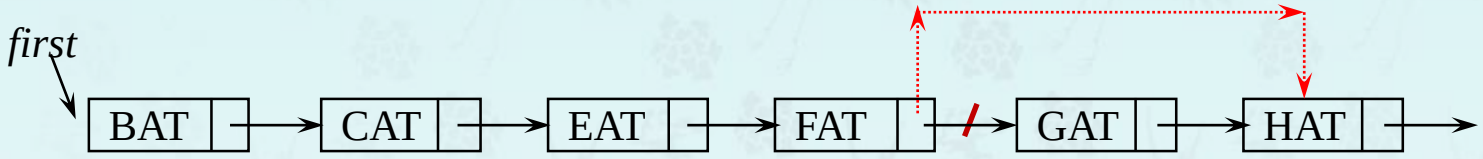
Q: How do you **insert** a node **GAT** after **FAT** or before **HAT** in the ordered list?





4.1 Singly linked lists and chains

Q: How do you **delete** a node **GAT** in the linked list below?



(1) Find the node **FAT** that is in front of **GAT**.

(2) Set the node **FAT** link field to point the node **HAT**.

(3) Free the node **GAT**.

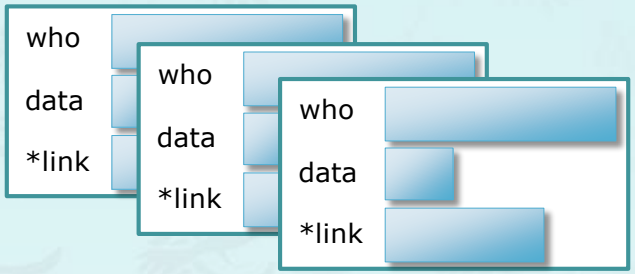
2.3 Self-referenced structures

Exercise: Link a, b and c nodes;

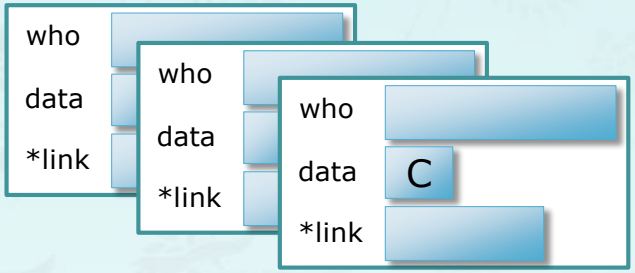
```
typedef struct list {
      student who;
      char  data;
      list    *link;      // pointer to list type
}list;

list a, b, c;
a.data = 'A';
b.data = 'B';
c.data = 'C';
a.link = b.link = c.link = NULL;
```

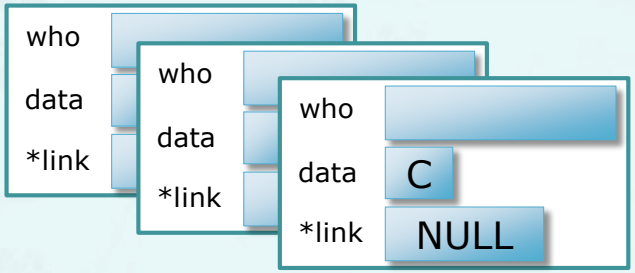
list a, b, c;

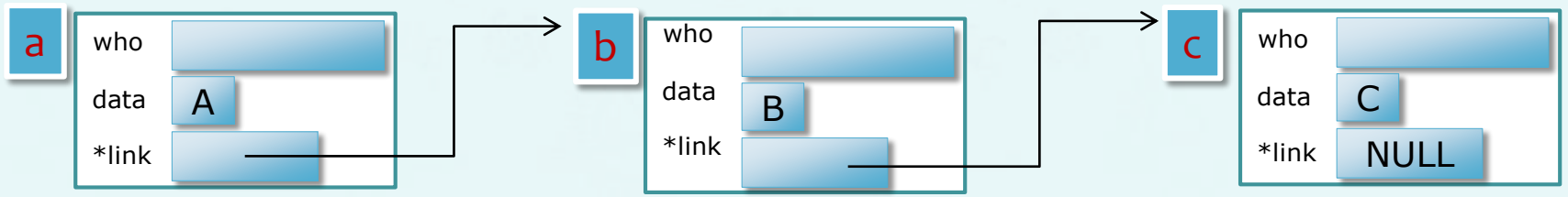


a.data = 'A';
b.data = 'B';
c.data = 'C';



a.link = b.link = c.link = NULL;





Chapter 2 – Arrays and structures

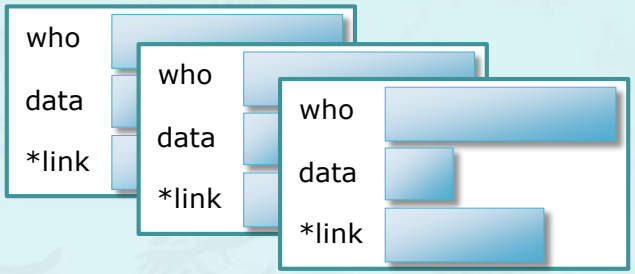
2.3 Self-referenced structures

Exercise: Link a, b and c nodes;

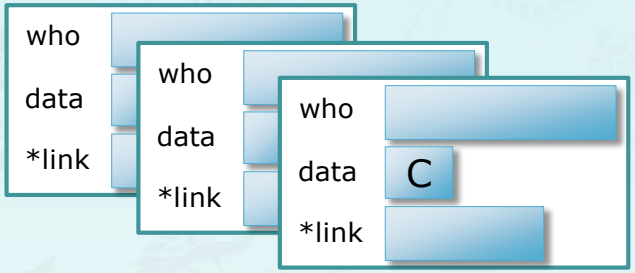
```
typedef struct list {
      student who;
      char  data;
      list    *link;      // pointer to list type
}list;

list a, b, c;
list *p, *q, *r;
```

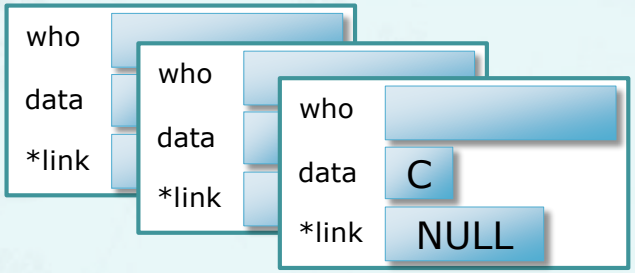
list a, b, c;



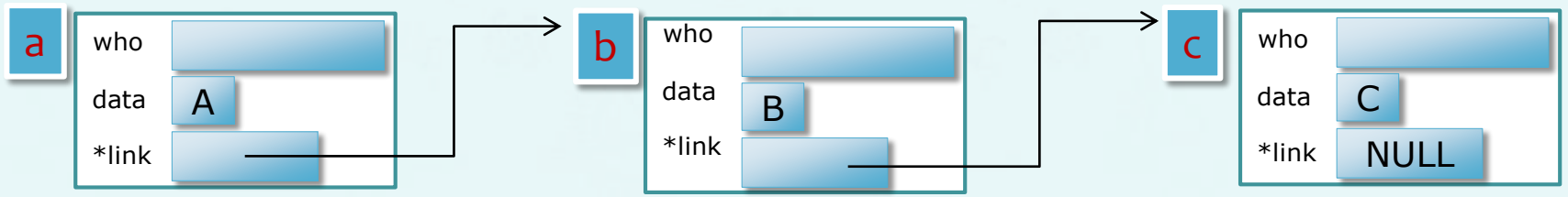

```
a.data = 'A';
b.data = 'B';
c.data = 'C';
```



a.link = b.link = c.link = NULL;

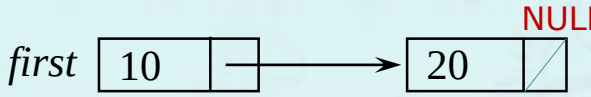


(1) Let each p, q, and r points to a, b, and c;
(2) Store each 'a', 'b', and 'c' in data, and NULL in link, respectively.
(3) Connect them using p, q and r as shown below:



4.2 Representing chains in C

Q: Write a function that creates linked list with two nodes & returns the first node:




```
typedef struct node *pNode
```

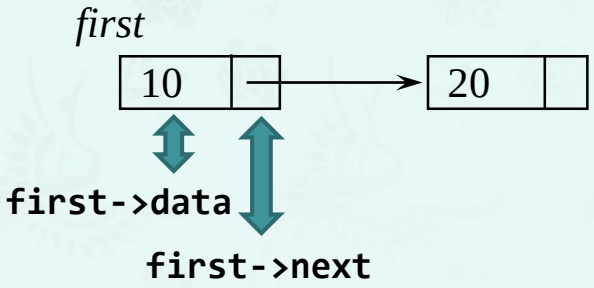
```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

```
pNode newNode(int item) {
  pNode node = (pNode)malloc(sizeof(node));
  node->data = item;
  node->next = NULL;
}
```

A:

```
pNode newList2(int a, int b)
  first  = newNode(a);
  second = newNode(b);

}
```



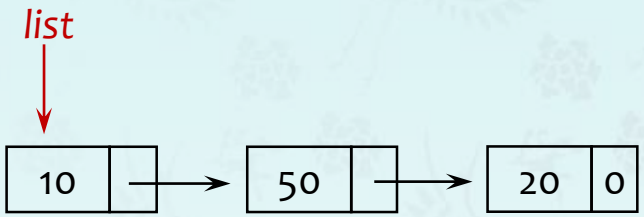
Chapter 4 – Linked lists

4.2 Representing chains in C

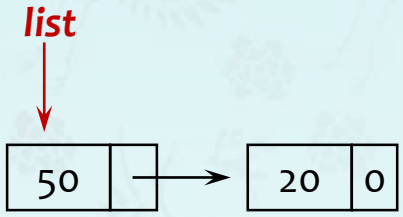
```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

deleteFirst(): Deleting the first node in the **list.** The second node becomes "first".



(a) before deletion



(b) after deletion

```
void deleteFirst(pList list)
```

```
list = list->next;
```

What's wrong? Then how do we deallocate the first node(10)?

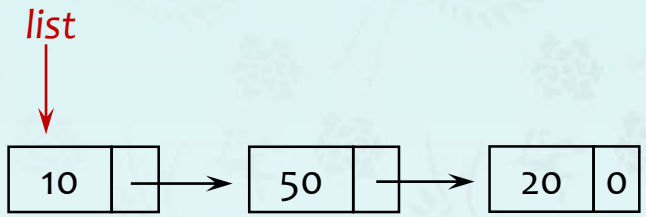
Chapter 4 – Linked lists

4.2 Representing chains in C

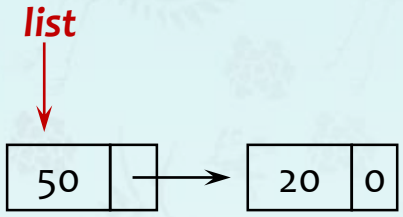
```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

deleteFirst(): Deleting the first node in the **list.** The second node becomes "first".



(a) before deletion



(b) after deletion

```
pNode deleteFirst(pList list)
```

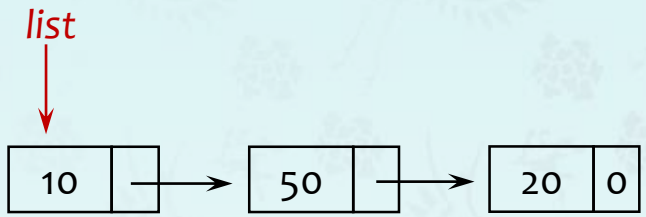
```
pNode x = list;
list = list->next;
free(x);
```

4.2 Representing chains in C

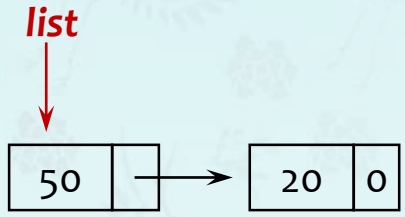
```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

deleteFirst(): Deleting the first node in the **list.** The second node becomes "first".



(a) before deletion



(b) after deletion

```
pNode deleteFirst(pList list)
```

```
pNode x = list;
list = list->next;
free(x);
return list
```

Chapter 4 – Linked lists

4.2 Representing chains in C

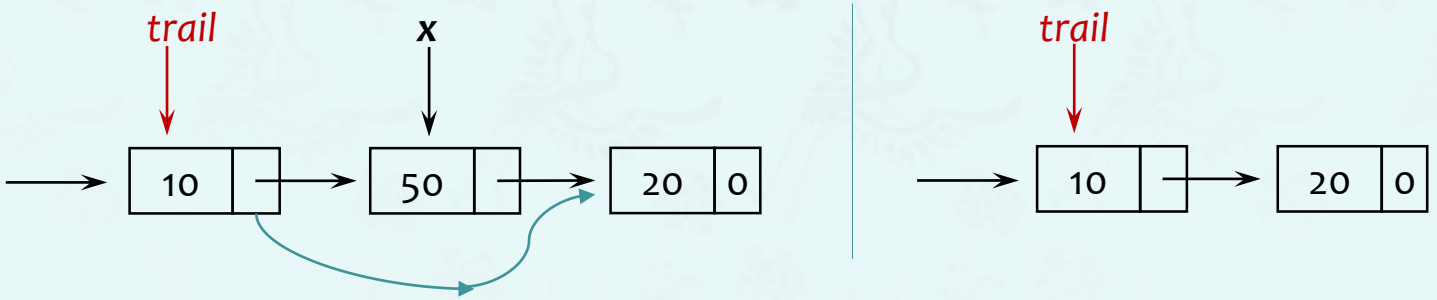
```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Deletion: Deleting a node x which is located in the middle of the list.

```
Given: trail, x
```

```
trail->next = x->next;
free(x);
```



Chapter 4 – Linked lists

4.2 Representing chains in C

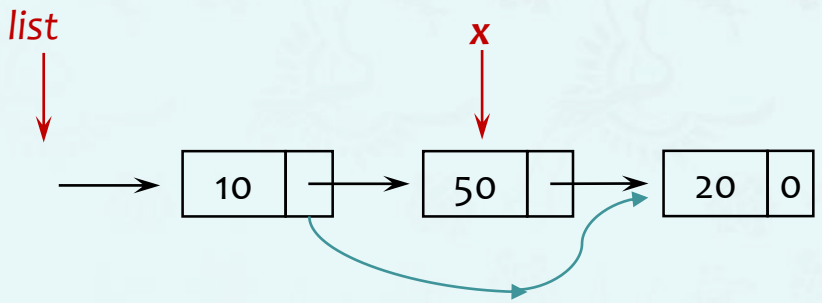
```
typedef struct node *pNode
```

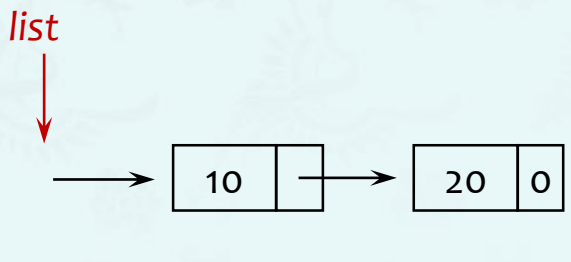
```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Deletion: Deleting a node x which is located in the middle of the list.

```
Given: list, x
```

Q: How about deleting x **without trail node given?**





Chapter 4 – Linked lists

4.2 Representing chains in C

```
typedef struct node *pNode
```

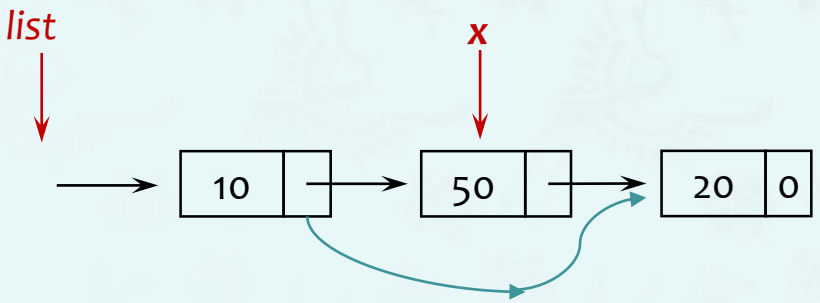
```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

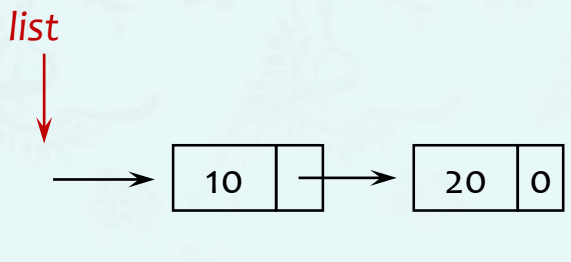
Deletion: Deleting a node x which is located in the middle of the list.

```
Given: list, x
```

Q: How about deleting x **without trail node given?**

A: While searching for x, **get the trail node as well.**





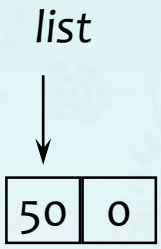
Chapter 4 – Linked lists

4.2 Representing chains in C

```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **at last of the list** .
If given **list** is null (or empty list), the node becomes **list.** (program4.2)



(a)

```
pNode newNode(int data) {
  pNode node = (pNode)malloc(sizeof(node));
  node->data = data;
  return node;
}node;
```

```
pNode addNodeLast(pNode list, int item)

if (list == NULL) {
  list = newNode(item);
}
```

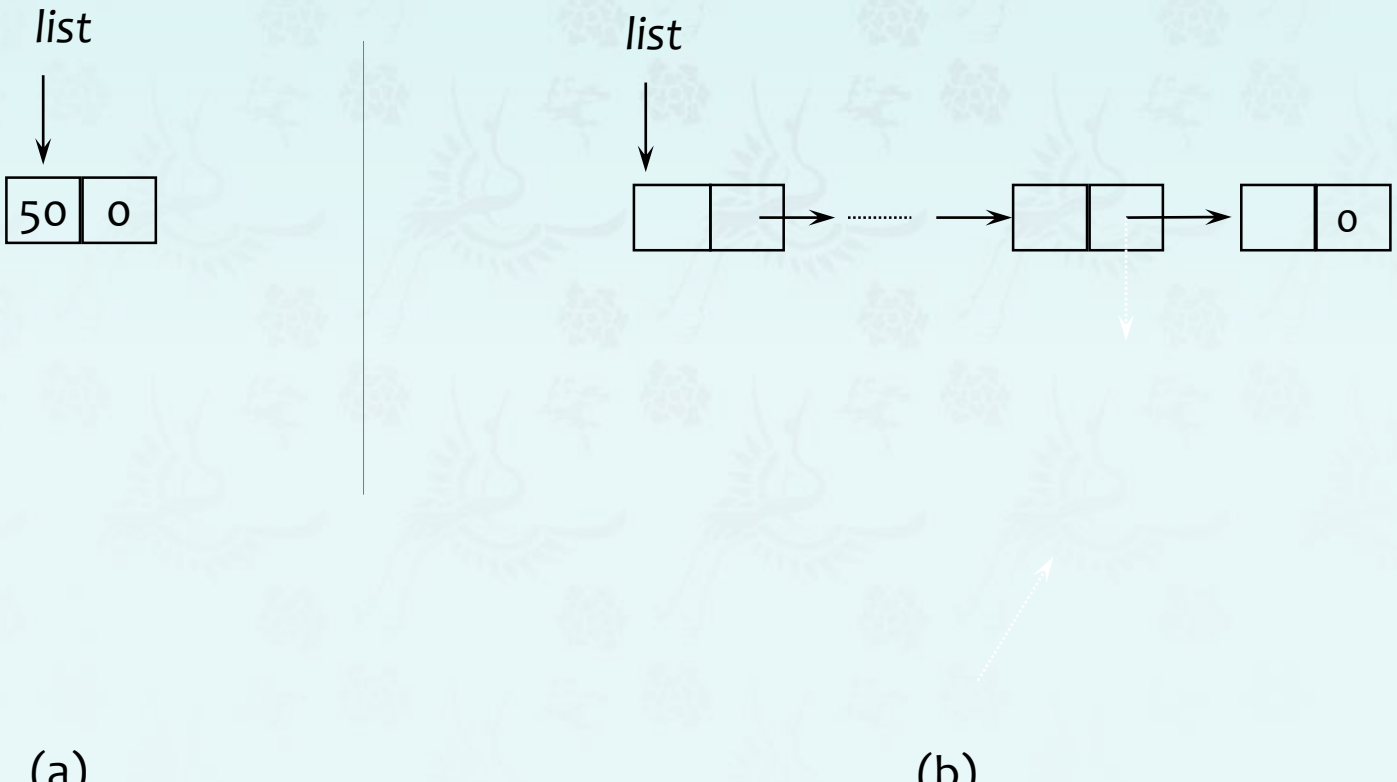
Chapter 4 – Linked lists

4.2 Representing chains in C

```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **at last of the list** .
If given **list** is null (or empty list), the node becomes **list.** (program4.2)

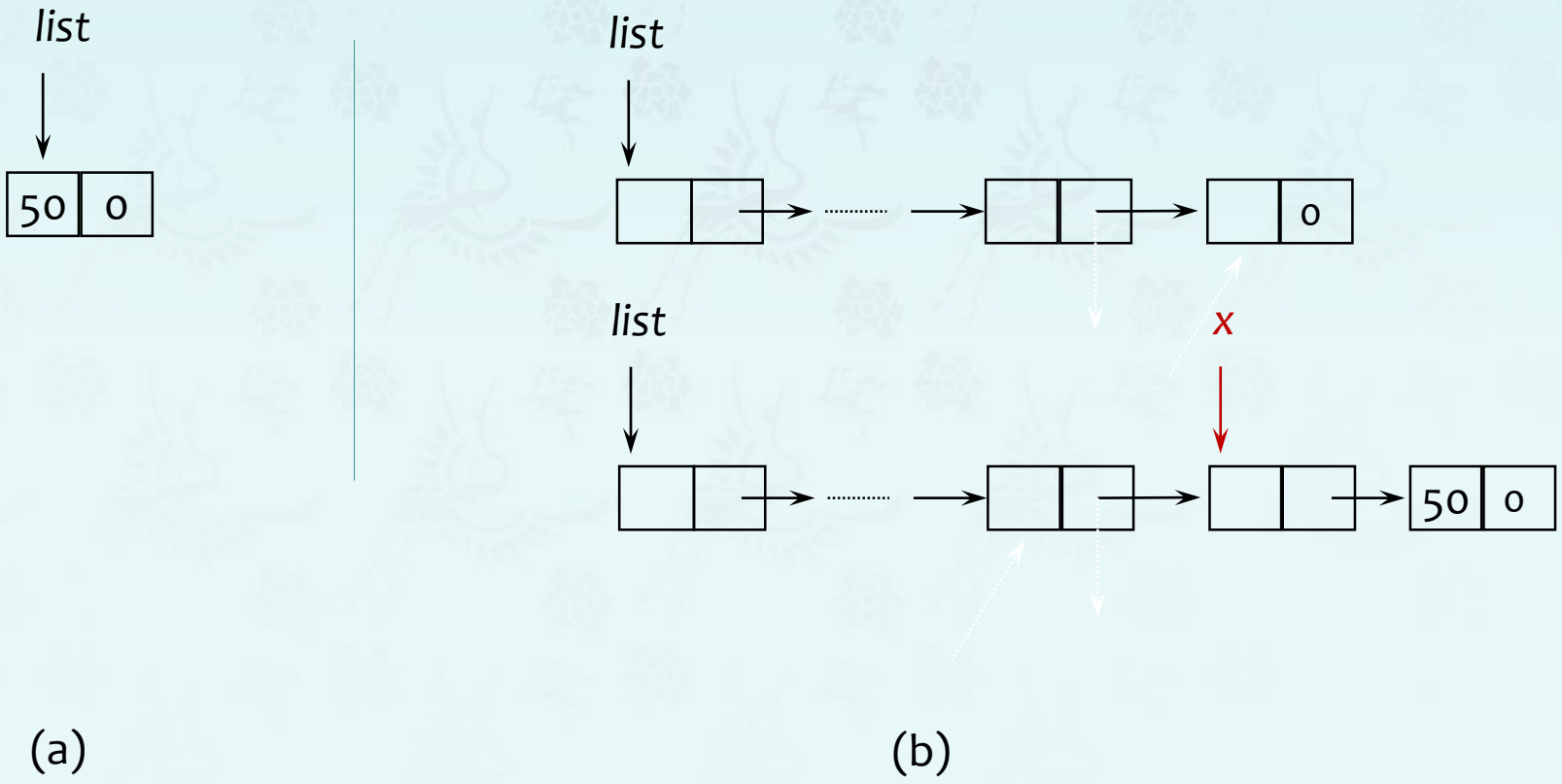


```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;     // data
  pNode next;     // link
}node;
```

4.2 Representing chains in C

Q: Make a node **item** = 50, Add it **at last of the list** .
If given **list** is null (or empty list), the node becomes **list.** (program4.2)



Chapter 4 – Linked lists

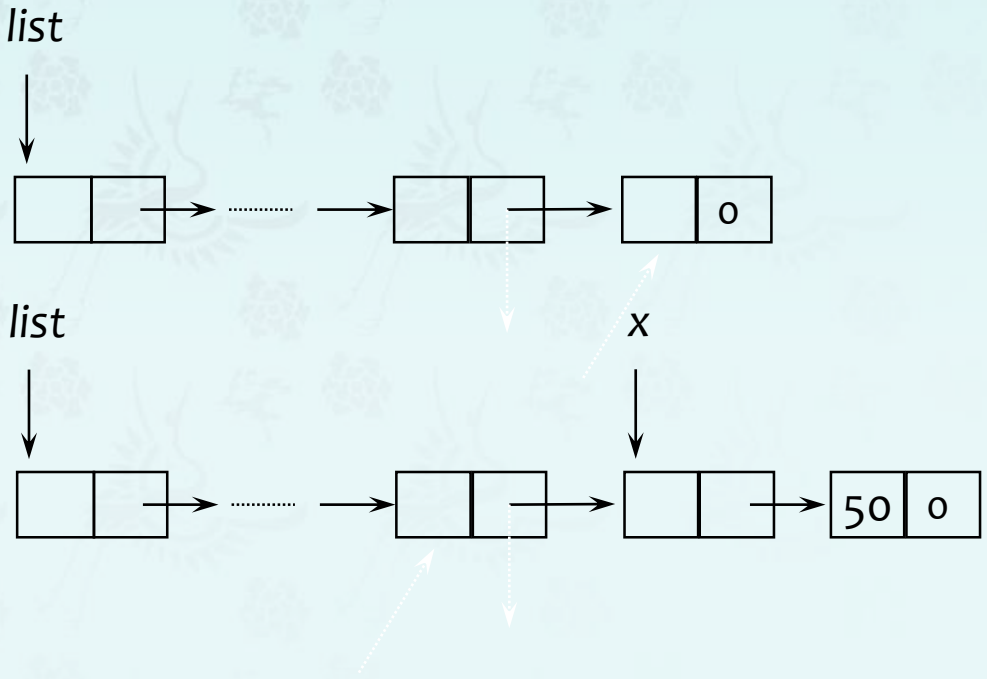
4.2 Representing chains in C

```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **at last of the list** .
If given **list** is null (or empty list), the node becomes **list.** (program4.2)

```
Given: list, item = 50
// find the last node x
pNode x = list
While (x != NULL) {
    
}
```



(b)

Chapter 4 – Linked lists

4.2 Representing chains in C

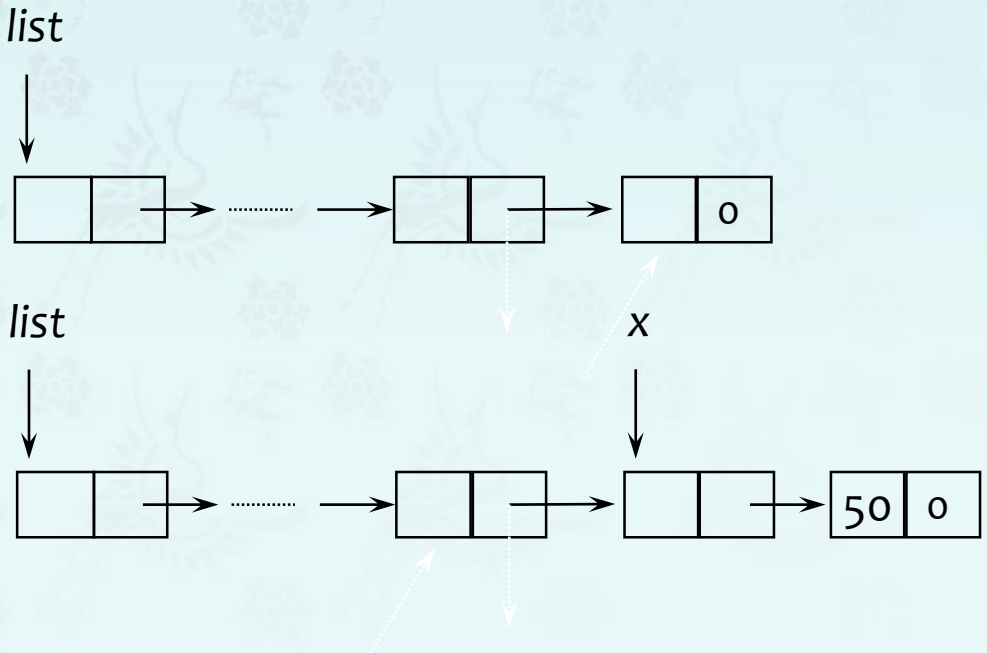
```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **at last of the list** .
If given **list** is null (or empty list), the node becomes **list.** (program4.2)

```
Given: list, item = 50
```

```
// find the last node x
pNode x = list
While (x != NULL) {
   x = x->next;
}
```



(b)

Chapter 4 – Linked lists

4.2 Representing chains in C

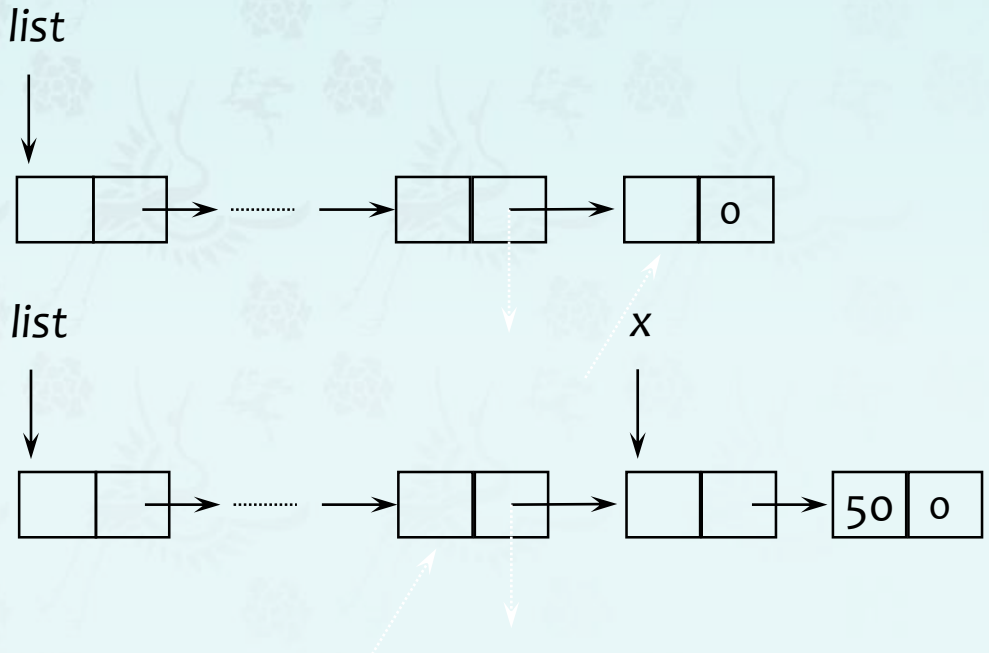
```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **at last of the list** .
If given **list** is null (or empty list), the node becomes **list.** (program4.2)

```
Given: list, item = 50
```

```
// find the last node x
pNode x = list
While (x != NULL) {
  if (x->next == NULL) {
    
  }
  x = x->next;
}
```



(b)

4.2 Representing chains in C

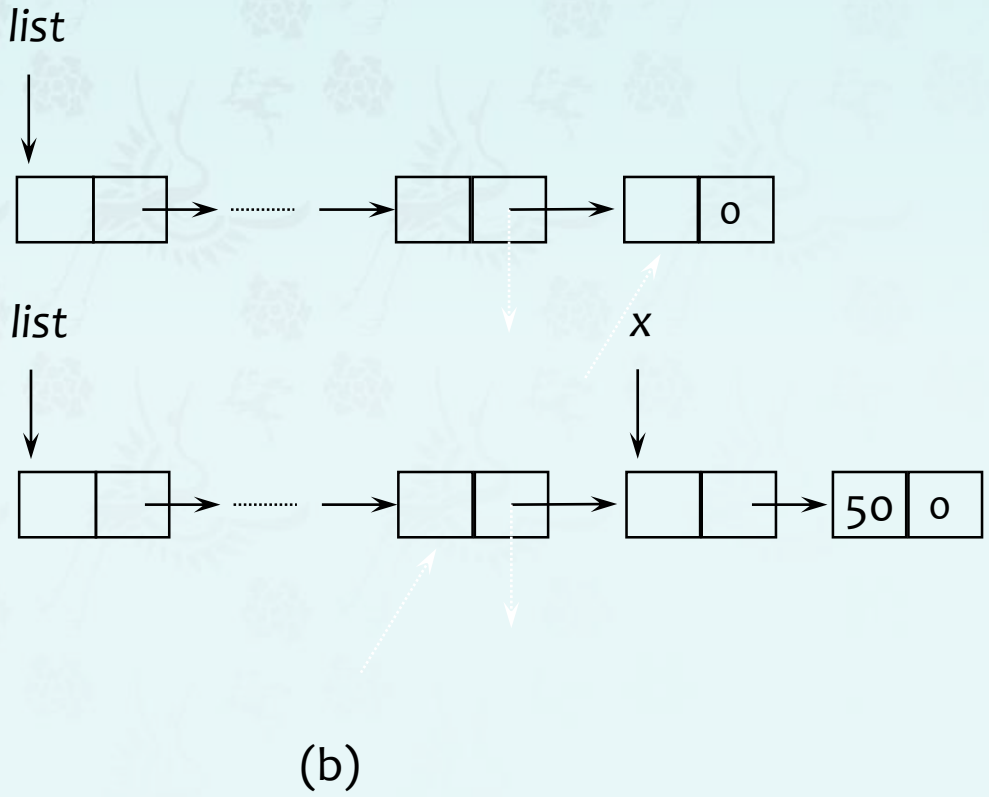
```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **at last of the list** .
If given **list** is null (or empty list), the node becomes **list.** (program4.2)

```
Given: list, item = 50
```

```
// find the last node x
pNode x = list
While (x != NULL) {
  if (x->next == NULL) {
     x->next = newNode(item);
     break;
  }
  x = x->next;
}
```



(b)

4.2 Representing chains in C

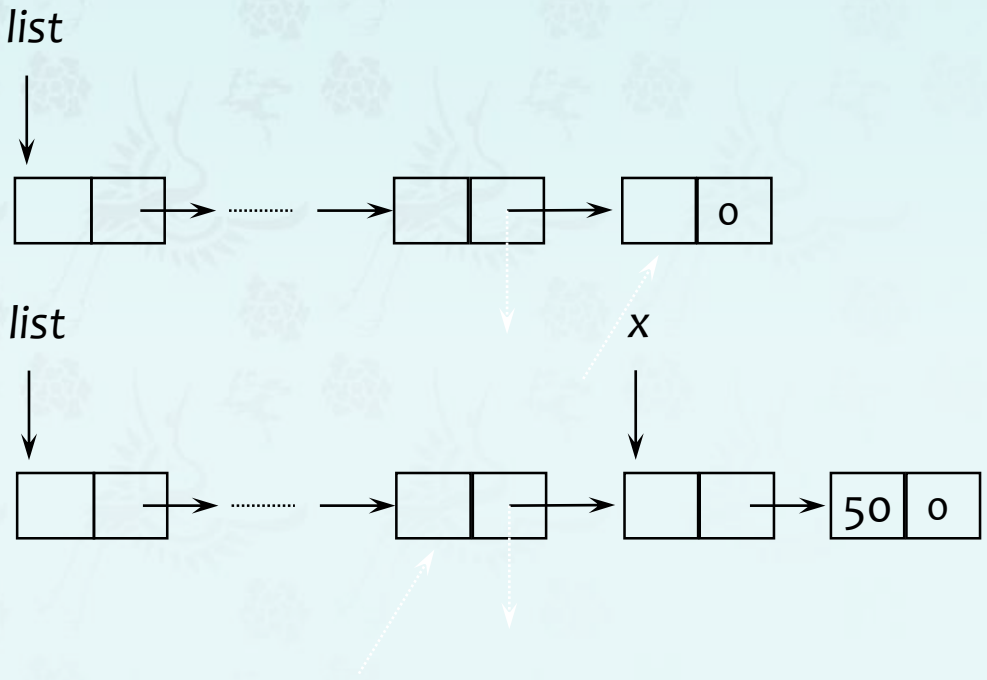
```
typedef struct node *pNode
```

```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **at last of the list** .
If given **list** is null (or empty list), the node becomes **list.** (program4.2)

```
pNode addNodeLast(pList list, int item)
```

```
if (list == NULL) {
  list = newNode(item);
}
else // find the last node x
  pNode x = list
  while (x != NULL) {
    if (x->next == NULL) {
      x->next = newNode(item);
      break;
    }
  x = x->next;
  }
  return list;
}
```



(b)

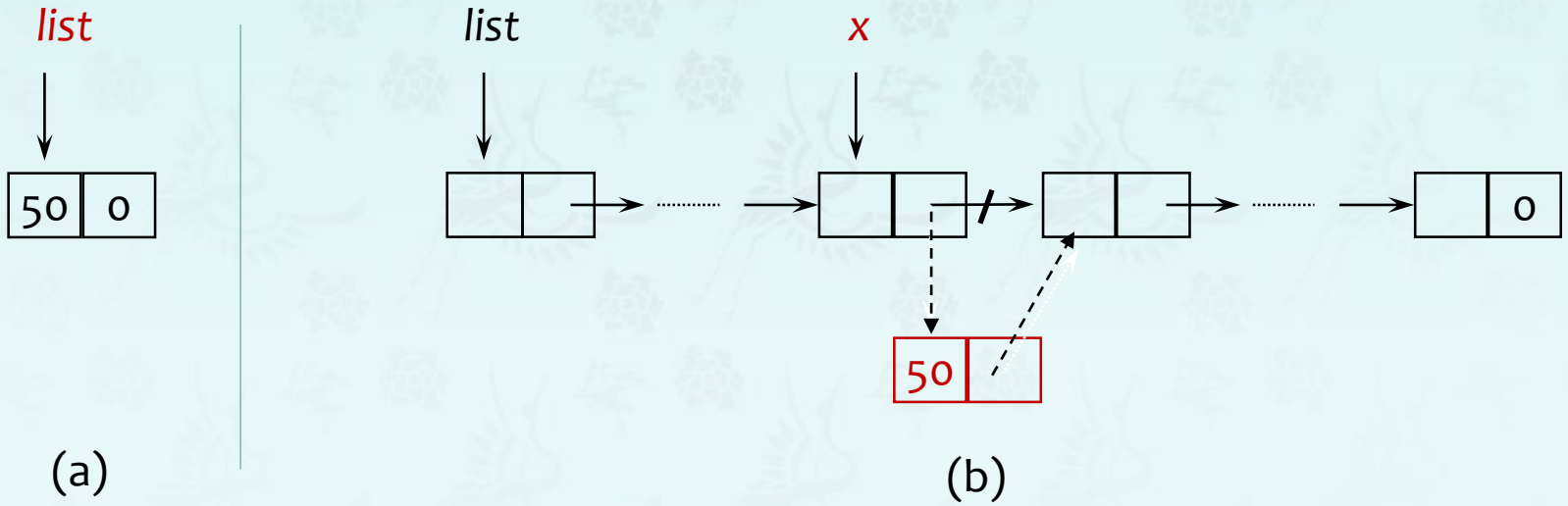
Chapter 4 – Linked lists

4.2 Representing chains in C

```
typedef struct node *pNode

typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **after** some arbitrary node **x**.
If given **list** is null (or empty list), the node becomes **list**. (program4.2)



Chapter 4 – Linked lists

4.2 Representing chains in C

```
typedef struct node *pNode
```

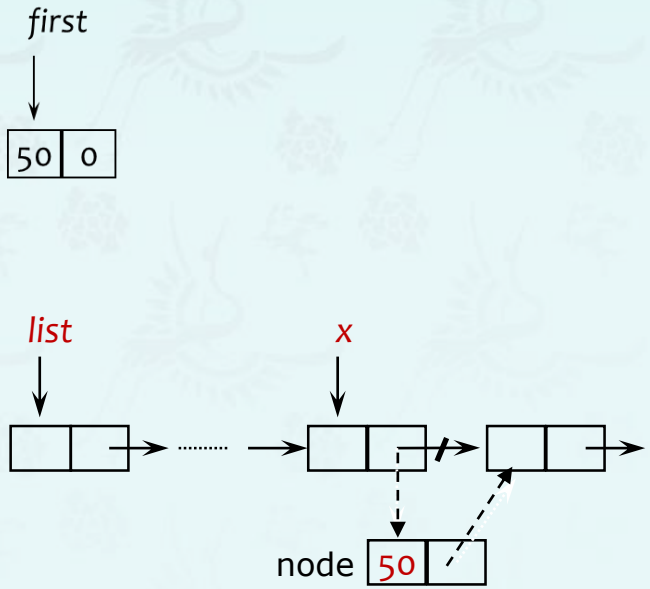
```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **after** some arbitrary node **x**.
If given **list** is null (or empty list), the node becomes **list**. (program4.2)

```
pNode newNode(int data) {
  pNode node = (pNode)malloc(sizeof(node));
  node->data = data;
  return node;
}node;
```

```
Given: list, x, item = 50

if (list == NULL) {
  list = newNode(item);
}
else {
  pNode node = newNode(item);
  
}
return list;
```



Chapter 4 – Linked lists

4.2 Representing chains in C

```
typedef struct node *pNode
```

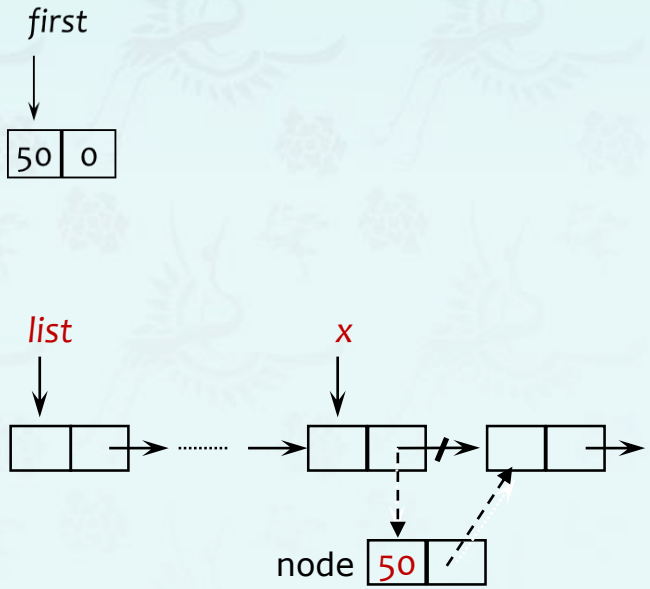
```
typedef struct node {
  int   data;    // data
  pNode next;    // link
}node;
```

Q: Make a node **item** = 50, Add it **after** some arbitrary node **x**.
If given **list** is null (or empty list), the node becomes **list**. (program4.2)

```
pNode newNode(int data) {
  pNode node = (pNode)malloc(sizeof(node));
  node->data = data;
  return node;
}node;
```

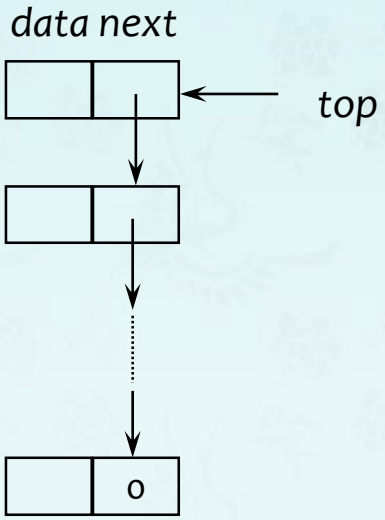
```
Given: list, x, item = 50

if (list == NULL) {
  list = newNode(item);
}
else {
  pNode node = newNode(item);
  node->next = x->next
  x->next = node;
}
return list;
```

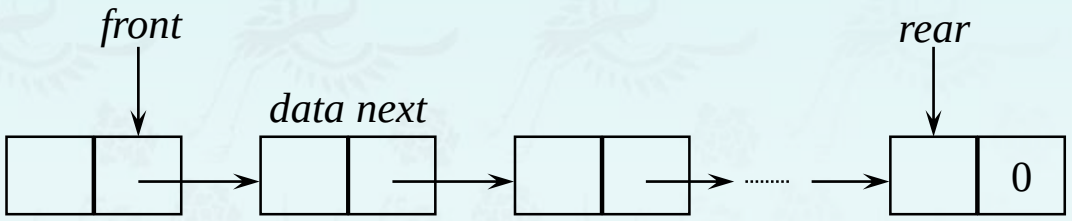


4.3 Linked list stacks and queues

Using linked lists, stacks and queues facilitate easy insertion and deletion of nodes.



(a) linked stack



(b) linked queue

4.3 Linked list stacks and queues

❖ resizing array vs. linked list

Tradeoffs. Can implement a stack with either resizing array or linked list; client can use interchangeably. Which one is better?

Linked-list implementation
- Every operation takes constant time in the worst case.
- Uses extra time and space to deal with the links.

Resizing-array implementation
- Every operation takes constant amortized time.
- Less waste space

4.4 Polynomials (다항식)

❖ Polynomials representation

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

a_i = nonzero coefficients

e_i = nonnegative integer exponents such as

$e_{m-1} > e_{m-2} > \ldots > a_0 \geq 0$

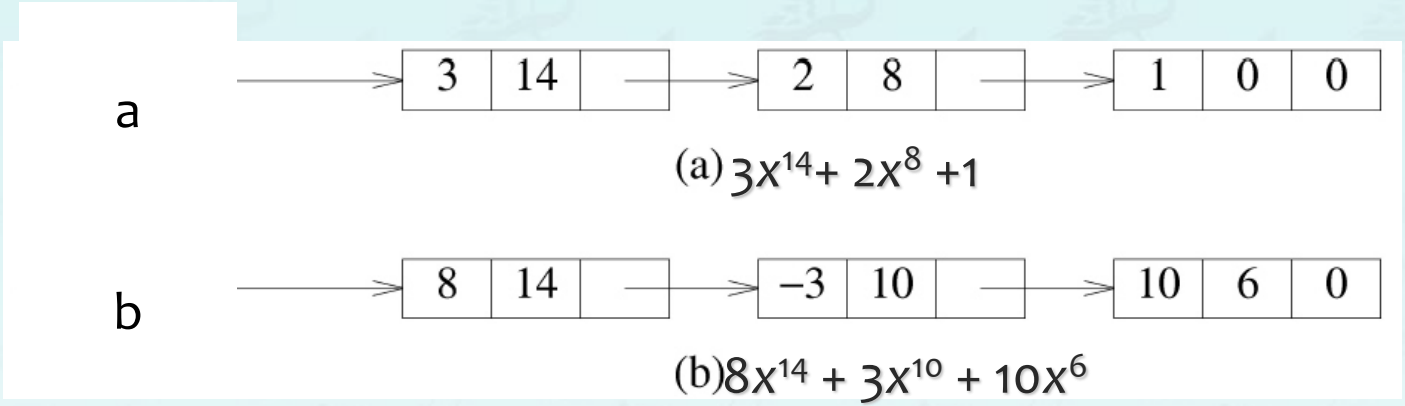
❖ We may draw a poly node as

❖ Type definition

```
typedef struct node *pPoly;
typedef struct poly {
    double      coef;
    double      expo;
    pPoly   next;
}poly;
```

4.4 Polynomials (다항식)

❖ **Example:**



Q: How to add two polynomials? c = a + b

Chapter 4 – Linked lists

4.4 Polynomials (다항식)

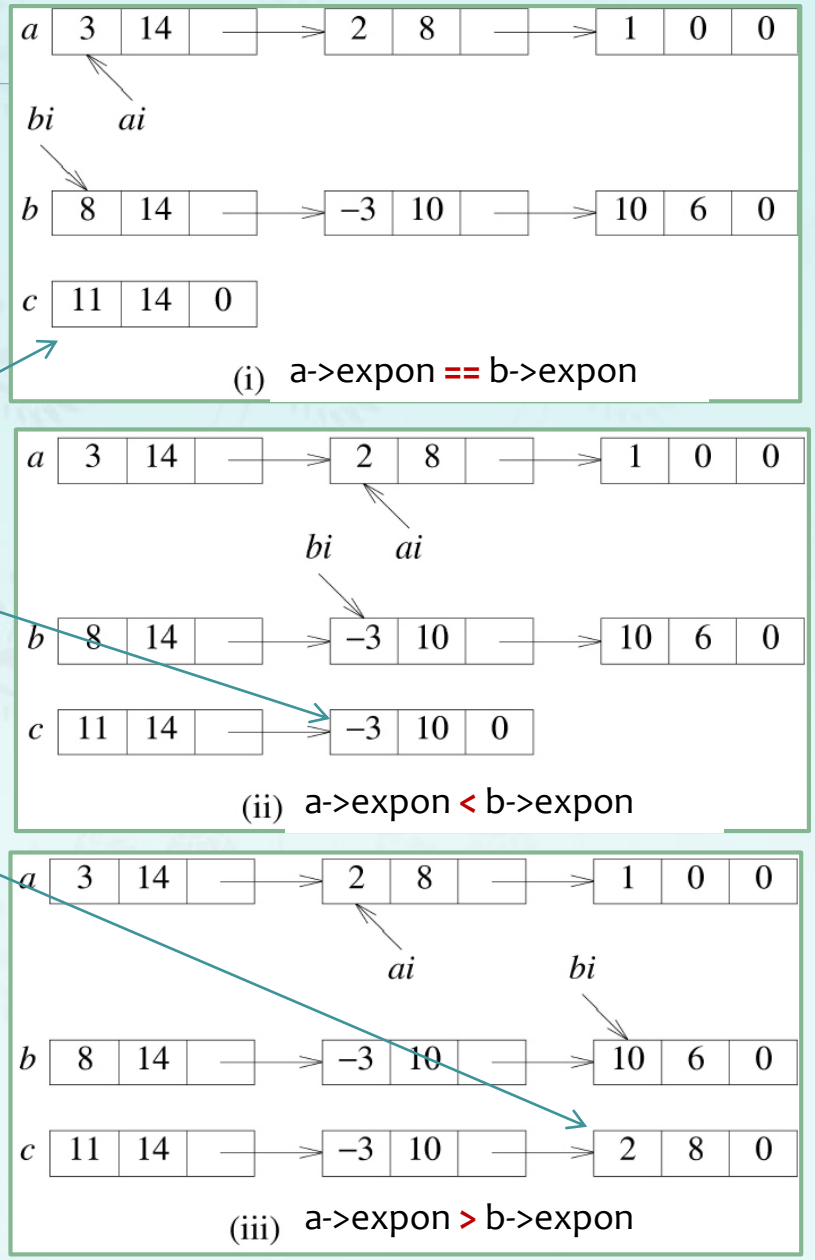
Q: How to add two polynomials?

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

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

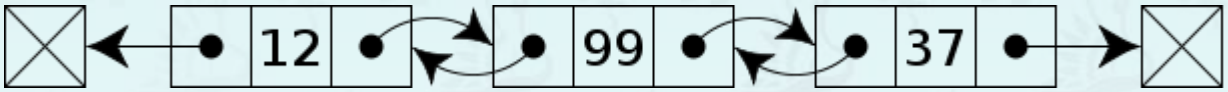
$c = a + b$

$= 11x^{14} - 3x^{10} + 2x^{8} + 10x^{6} + 1$



4.8 Doubly linked list

- **Doubly linked list:** each node contains, besides the **next**-node link, a second link field pointing to the ***previous*** node in the sequence. The two links may be called **forward** and **backward**, or **next** and **prev**(**ious**).



- **Type definition**

```
typedef struct node *pDNode
typedef struct node {
   int              item;
   pDNode   prev;
   pDNode   next;
} node;
```

Q. Array vs. Singly linked list vs. Doubly linked list, Why?

4.8 Doubly linked list

Q. Array vs. Singly linked list vs. Doubly linked list, Why?

Advantages of linked list:

- Dynamic structure (Memory Allocated at run-time)
- Have more than one data type.
- Re-arrange of linked list is easy (Insertion-Deletion).
- It doesn't waste memory.

Disadvantages of linked list:

- In linked list, if we want to access any node it is difficult.
- It is occupying more memory.

Advantages of doubly linked list:

- A doubly linked list can be **traversed in both directions** (forward and backward). A singly linked list can only be traversed in one direction.

ITP20001/ECE 20010 Data Structures

Data Structures

Chapter 4

- *singly linked list*
- *linked list stacks and queues*
- *polynomials (and sparse matrices)*
- *doubly linked list*