Chapter 4 Linked Lists

4.1 Singly Linked lists Or Chains

The representation of simple data structure using an array and a sequential mapping has the property:

- **Successive nodes of the data object are stored at fixed distance apart.**
- **This makes it easy to access an arbitrary node in O(1).**

Disadvantage of sequential mapping:

It makes insertion and deletion of arbitrary elements expensive.

For example:

Insert GAT into or delete LAT from

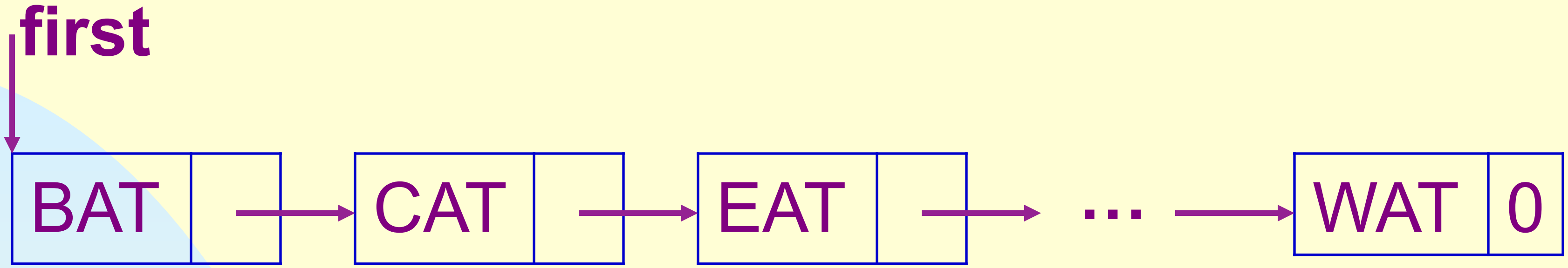
(BAT, CAT, EAT, FAT, HAT, JAT, LAT, MAT, OAT, PAT, RAT, SAT, TAT,VAT,WAT)

need data movement.

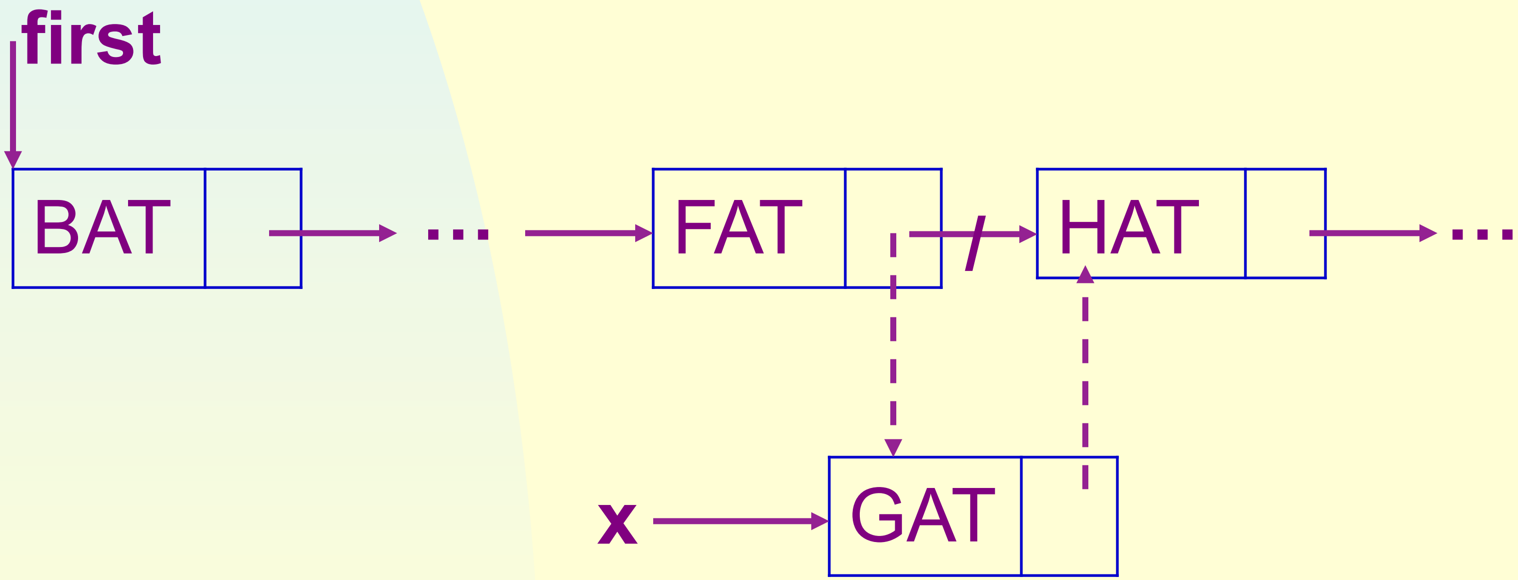
Solution---linked representation:

items of a list may be placed anywhere in the memory.

Associated with each item is a point (link) to the next item.



In linked list, insertion (deletion) of arbitrary elements is much easier:



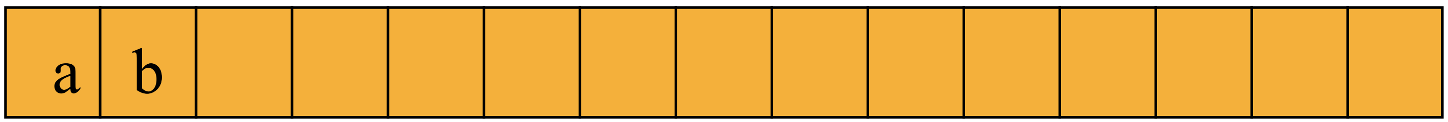
The above structures are called singly linked lists or chains in which each node has exactly one pointer field.

li elemen are ored, in memor , in an arbi rar order

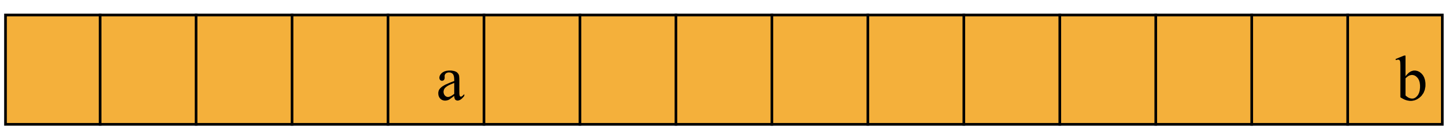
e plici informa ion (called a link) i ed o go from one elemen o he ne

Memory Layout

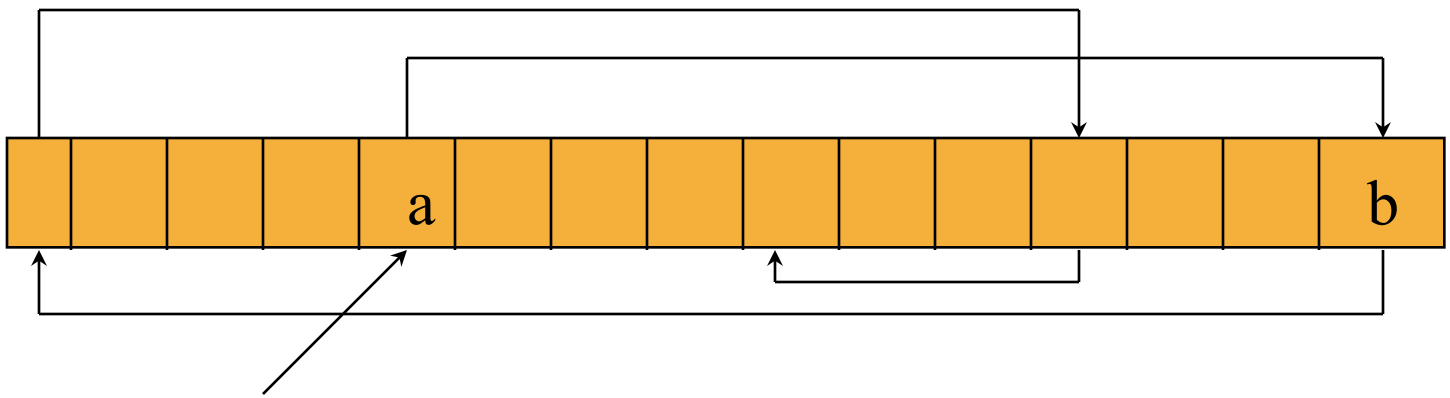
Layout of L = (a,b,c,d,e) using an array representation.



A a a a b a a .



Linked Representation



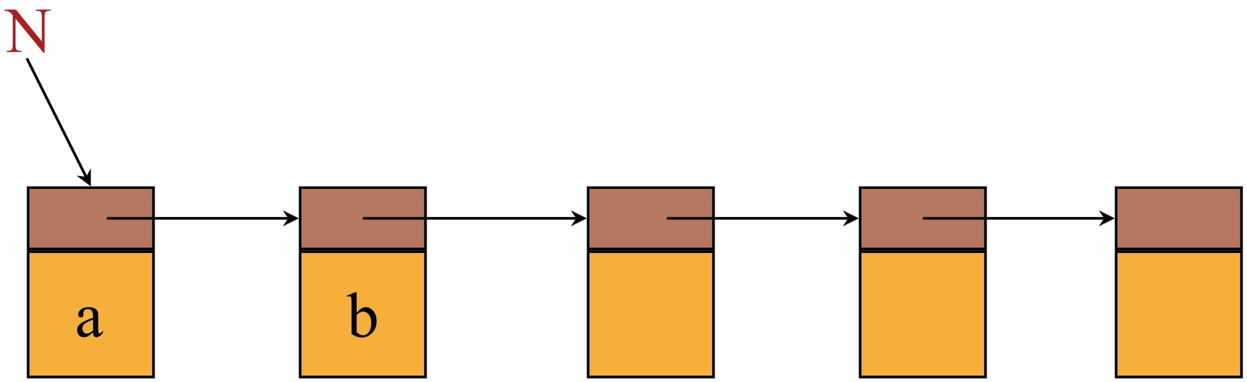
N

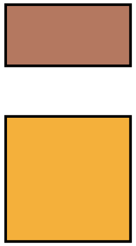
pointer (or link) in e is null

a a ab N

a

Normal Way To Draw A Linked List

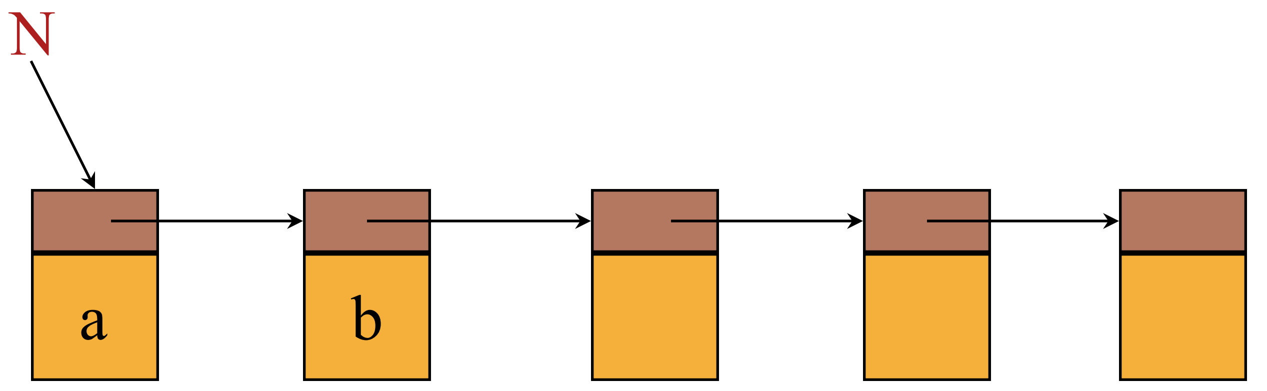




link or pointer field of node

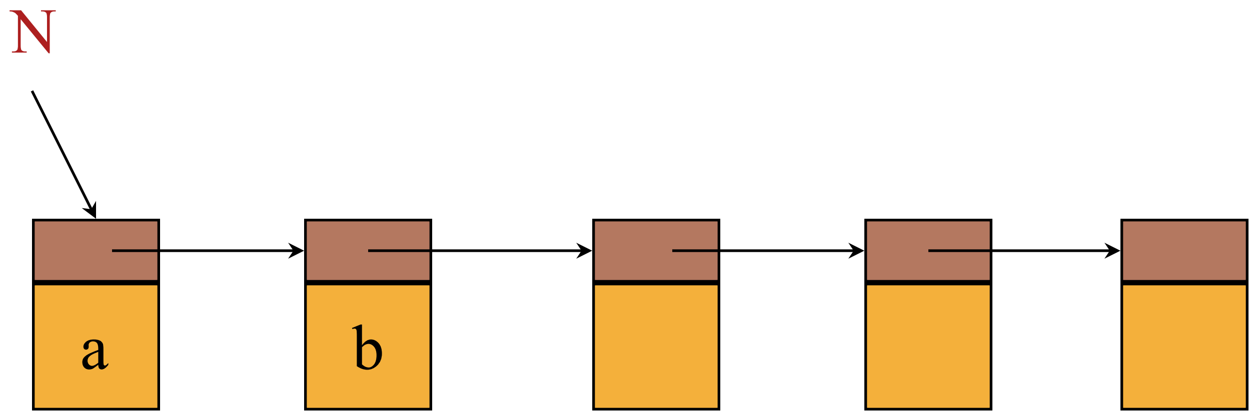
a a

Chain



- A chain is a linked list in which each node represents one element.
- There is a link or pointer from one element to the next.
- The last node has a null pointer.

get(0)

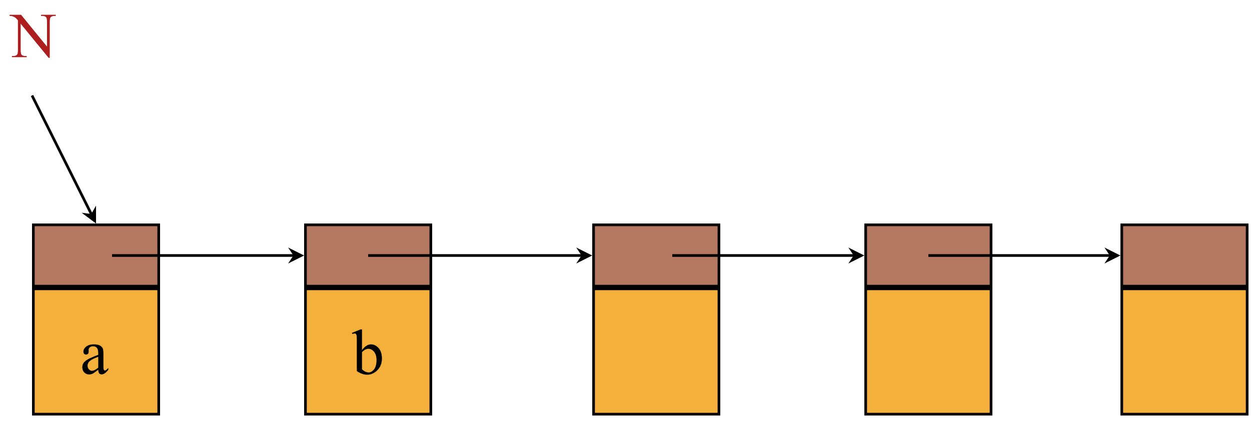


checkIndex(0);

desiredNode = firstNode; // gets you to first node

return desiredNode→element;

get(1)

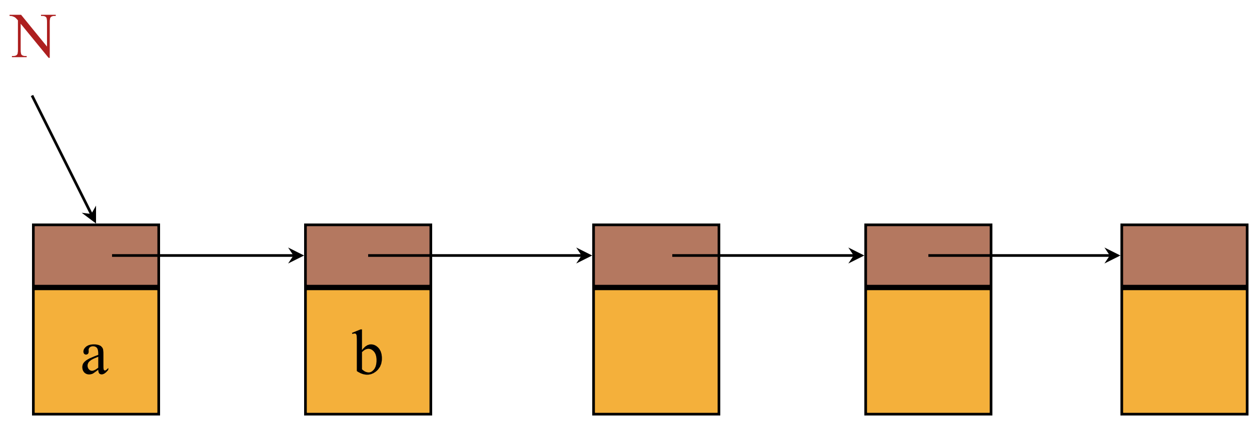


checkIndex(1);

desiredNode = firstNode → next; // gets you to second node

return desiredNode → element;

get(2)

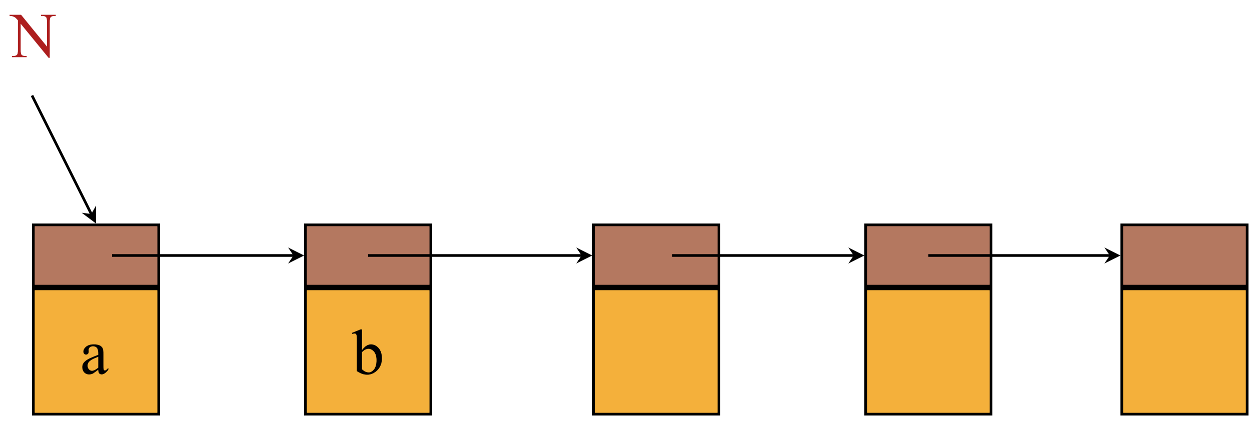


checkIndex(2);

desiredNode = firstNode → next → next; // gets you to third node

return desiredNode → element;

get(5)



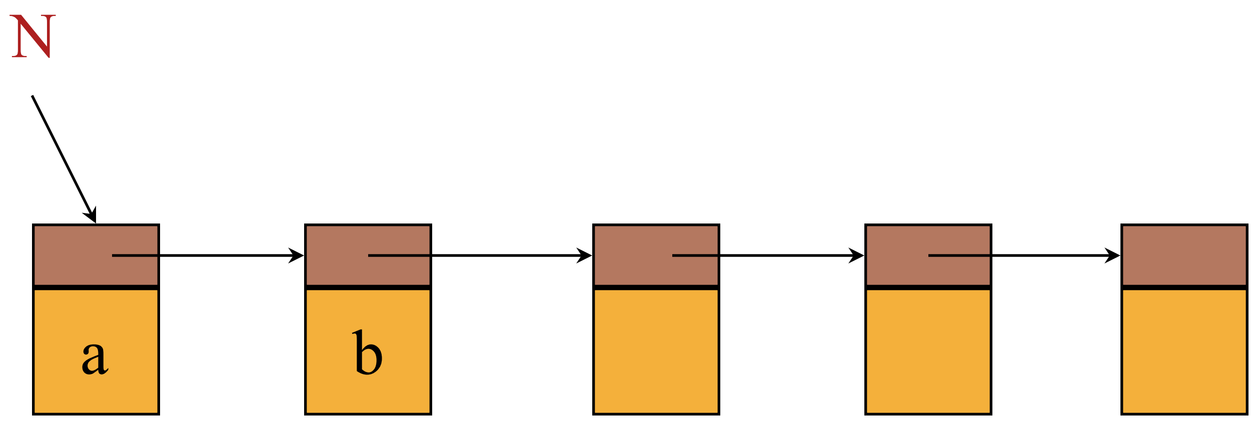
checkIndex(5); // throws exception

desiredNode =firstNodenextnext→next→next→next;

// desiredNode = null

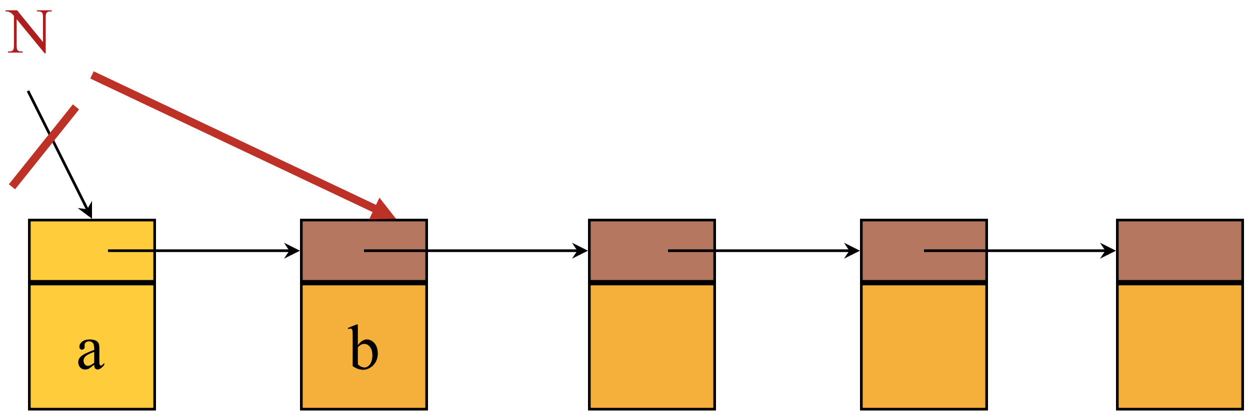
return desiredNode→element; // null→element

NullPointerException




```
desiredNode =
    firstNode→next→next→next→next→next;
      // gets the computer mad
      // you get a NullPointerException
```

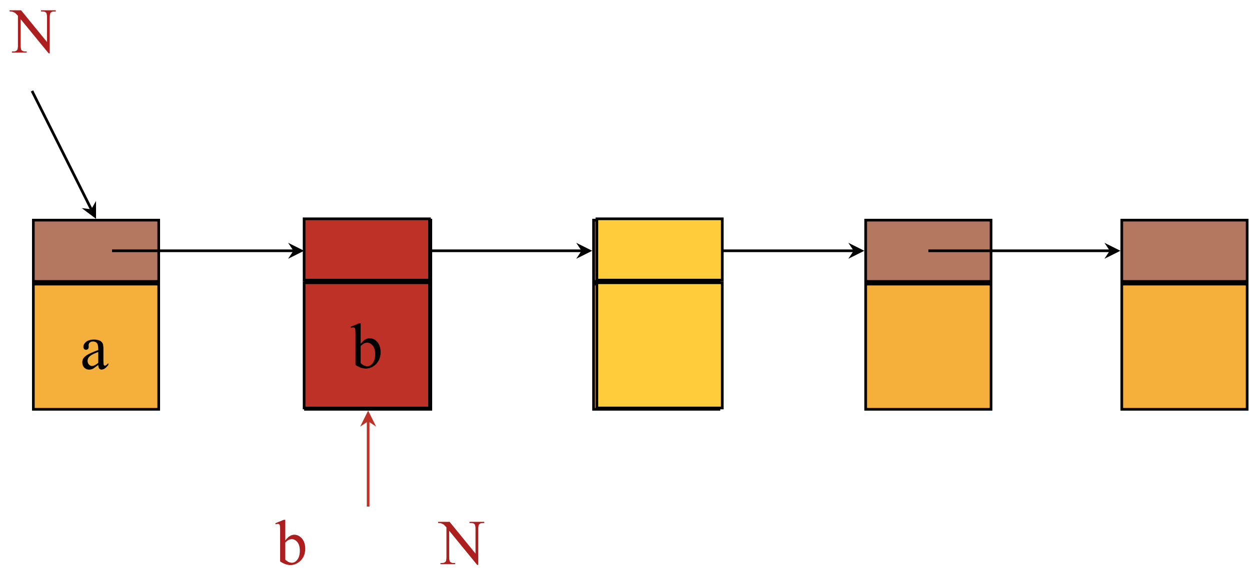
Remove An Element



remove(0)

N = N → ;

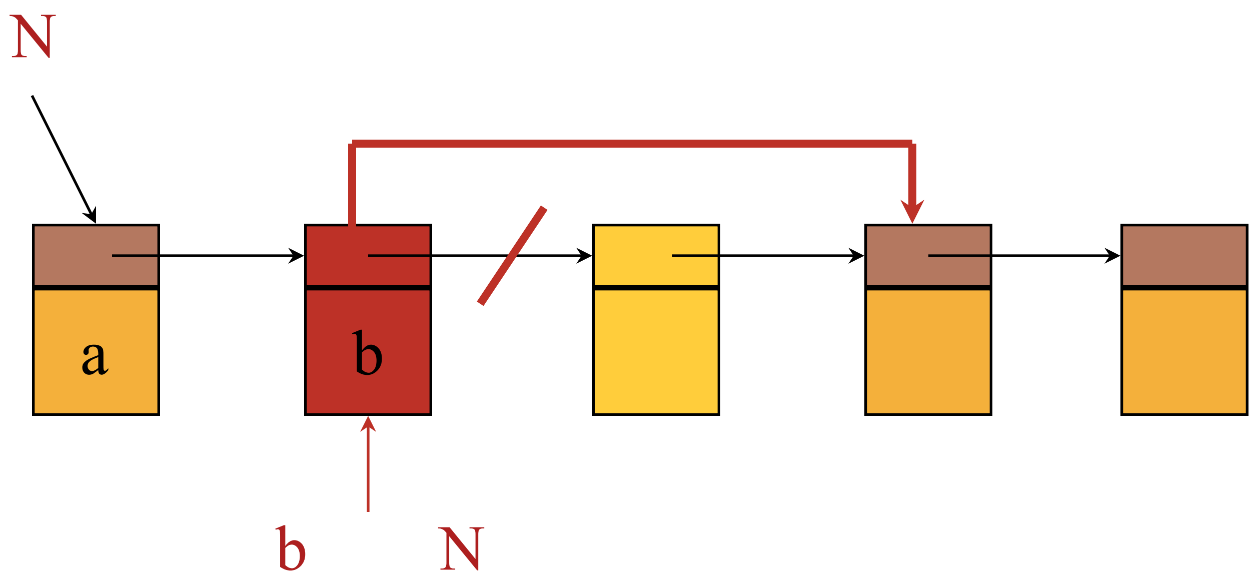
remove(2)



first get to node just before node to be removed

b N = N → ;

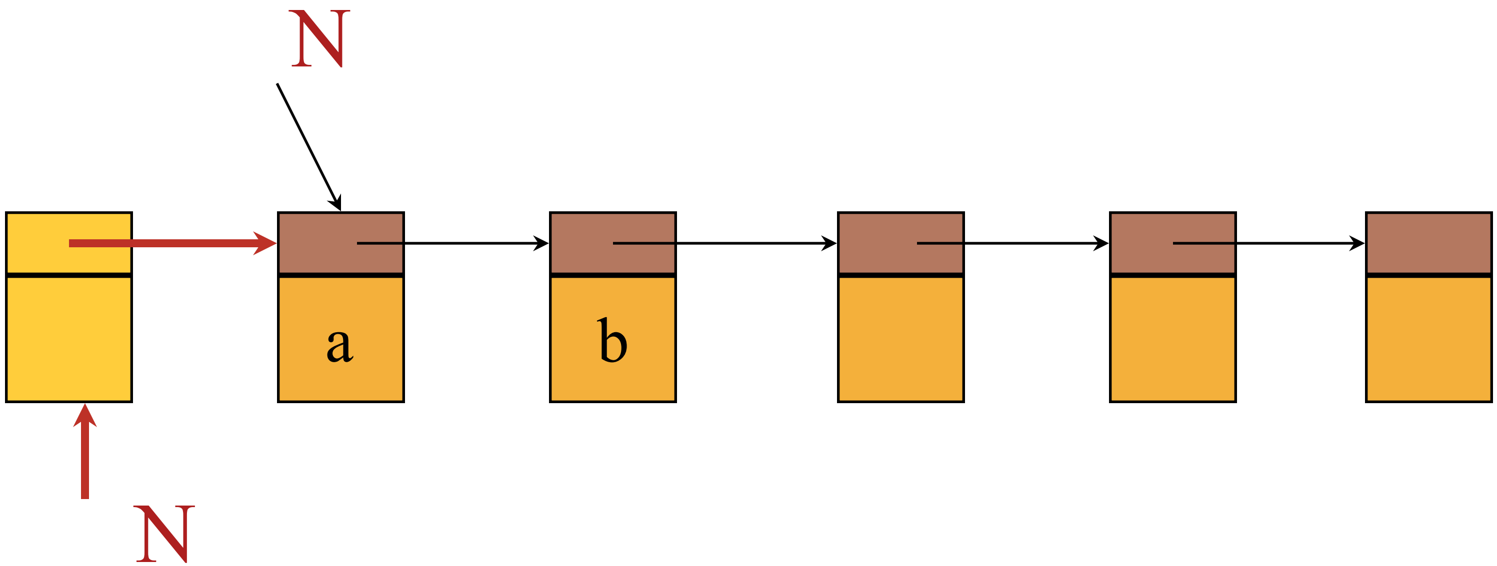
remove(2)



now change pointer in beforeNode

b N . = b N . . ;

add(0,’f’)

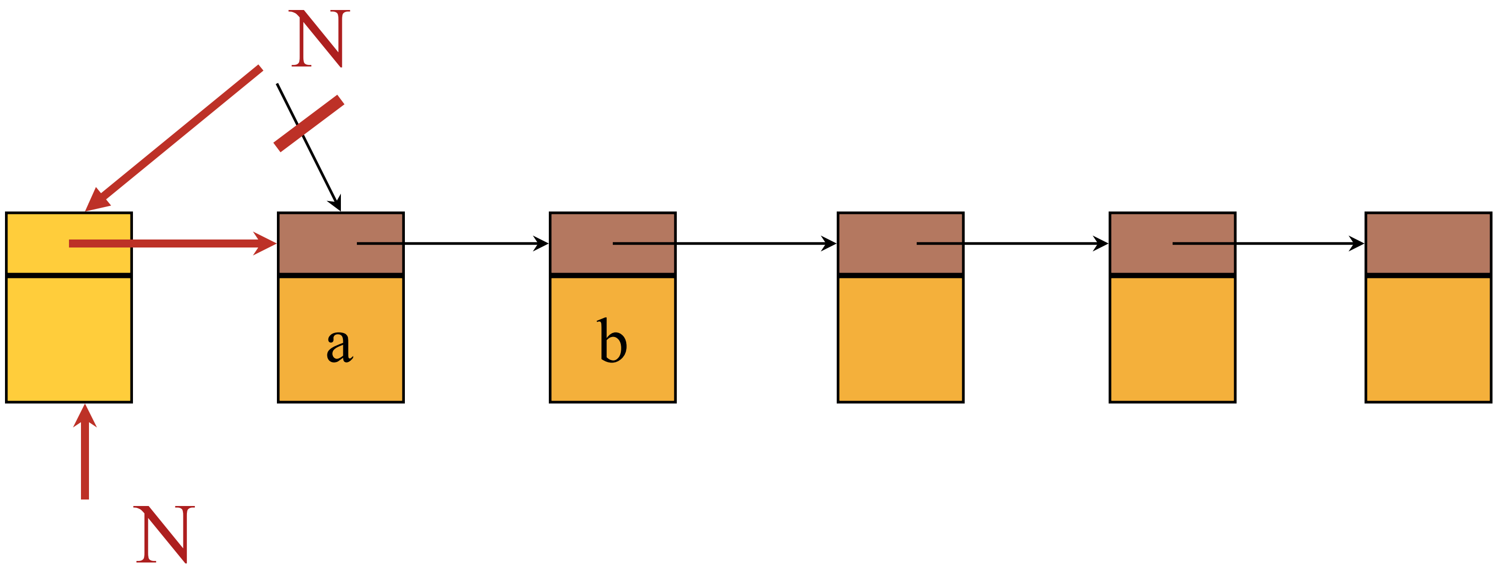


S 1: a , a a a

C a N N =

C a N (C a a ( ), N);

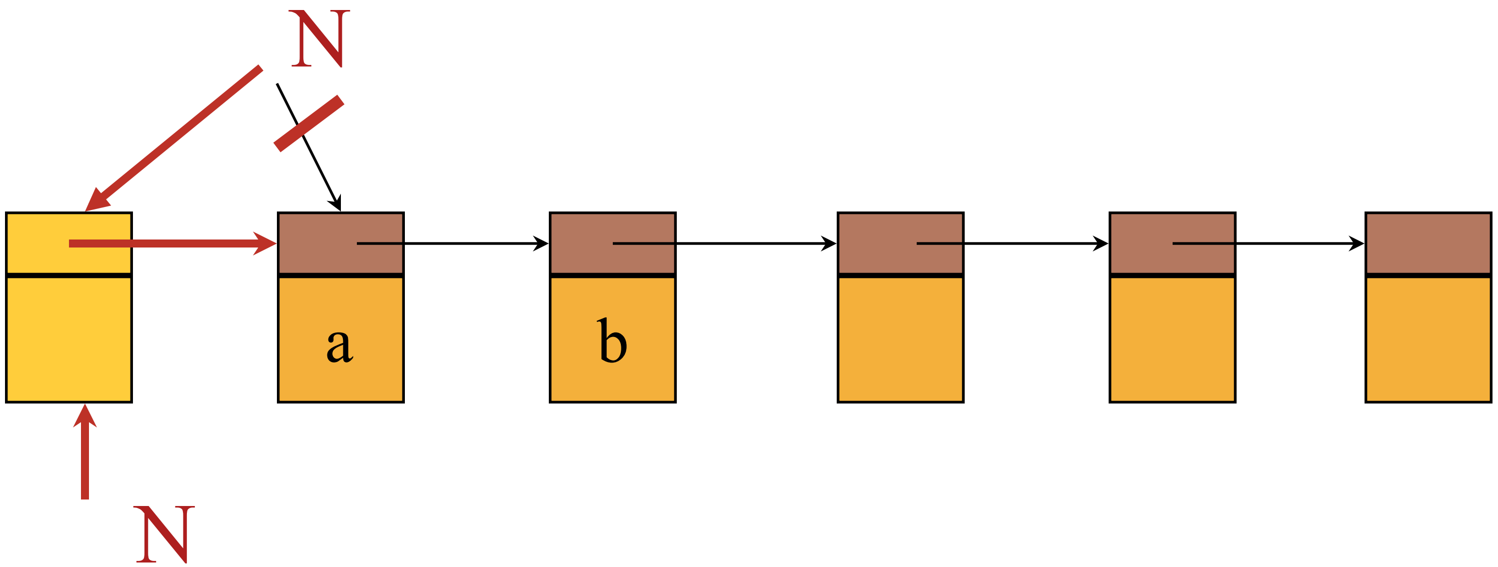
add(0,’f’)



S 2: a N

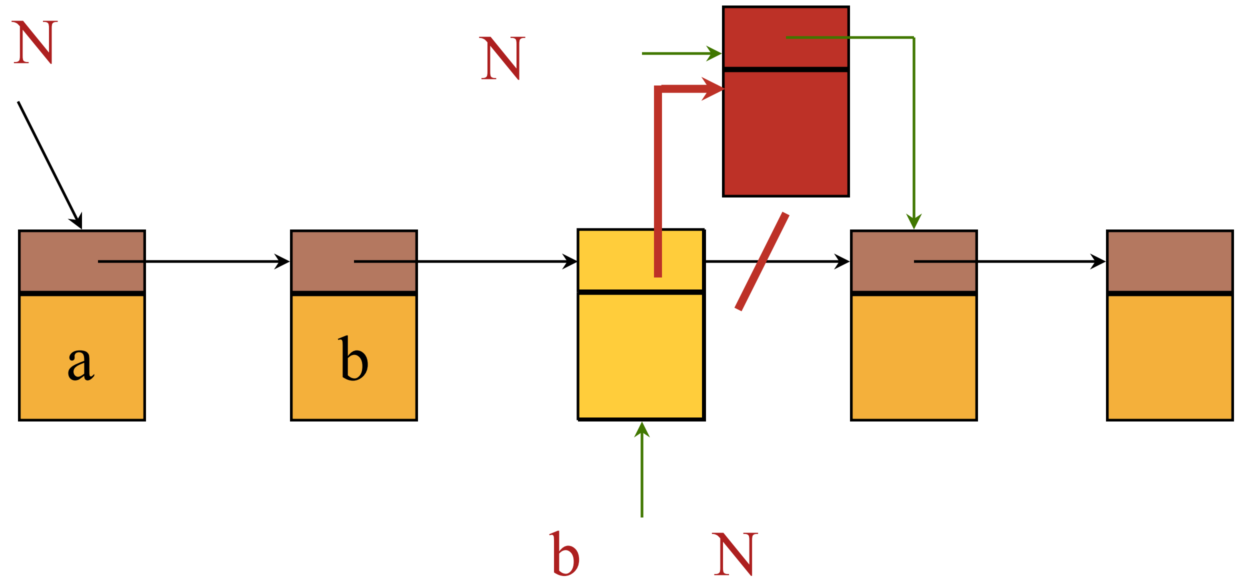
N = N ;

One-Step add(0,'f')



firstNode = new ChainNode(
new Character('f'), firstNode);

add(3,’f’)



- first find node whose index is 2

a a a a a a

C a N N = C a N (C a a ( ),

b N →);

a b N N

b N . = N ;

4.2 Representing Chains in C++

Assume a chain node is defined as:

```
class C a  N     {
private:
   int  a a;
   C a  N     *     ;
};
```

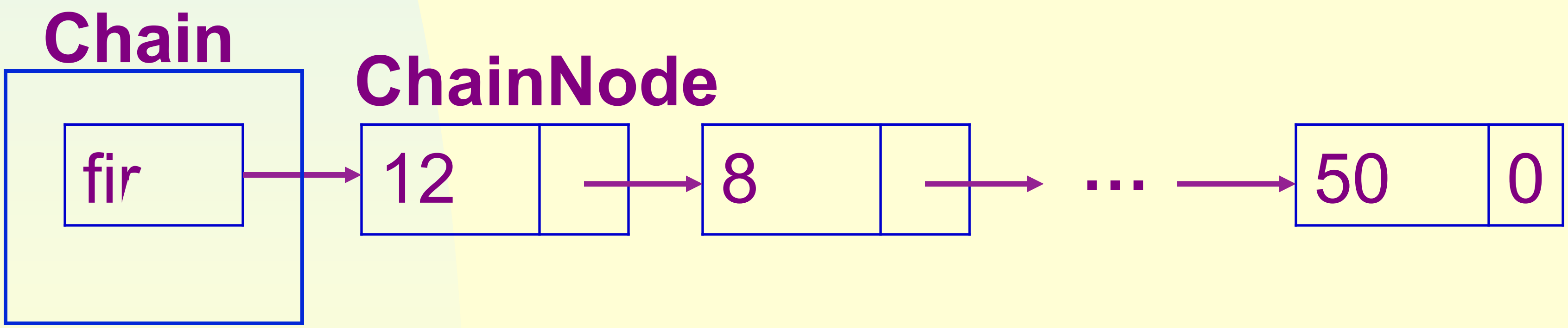
```
C a  N     * ;

 → a a
```

will cause a compiler error because a private data member cannot be accessed from outside of the object.

Definition: a data object of Type A HAS-A data object of Type B if A conceptually contains B or B is a part of A.

A composite of two classes: ChainNode and Chain.

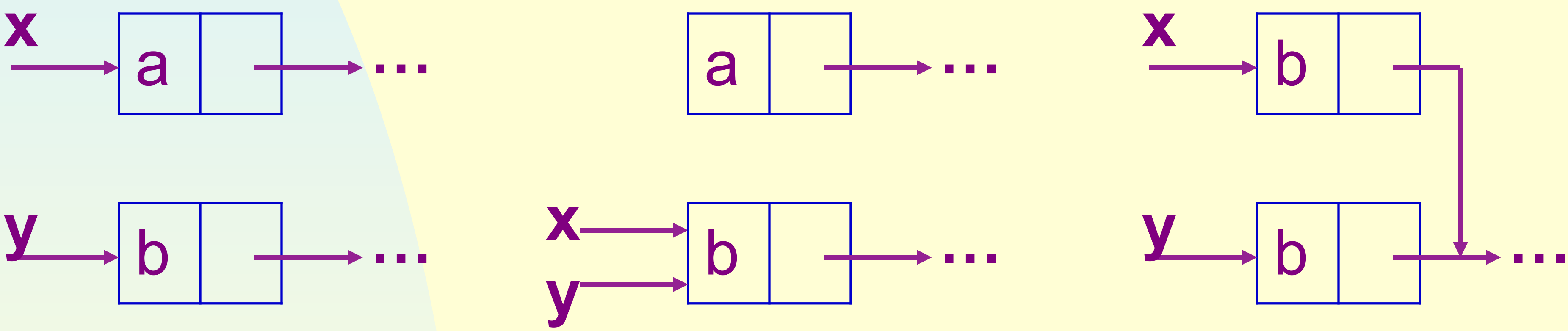
Chain HAS-A ChainNode.



```
class C  a   ;  //        a           a a
class C  a   N       {
friend class C  a   ; //        a                      C  a    b   ab
                    // a            a      a a        b         C  a   N
Public:
   C  a   N     (int              = 0, C  a   N     *        = 0)
         a a =            ;        =       ;
private:
   int   a a;
   C  a   N      *      ;
};
class C  a    {
public:
   // C  a      a       a           a

private:
   C  a   N      *      ;
};
```

Null pointer constant 0 is used to indicate no node.

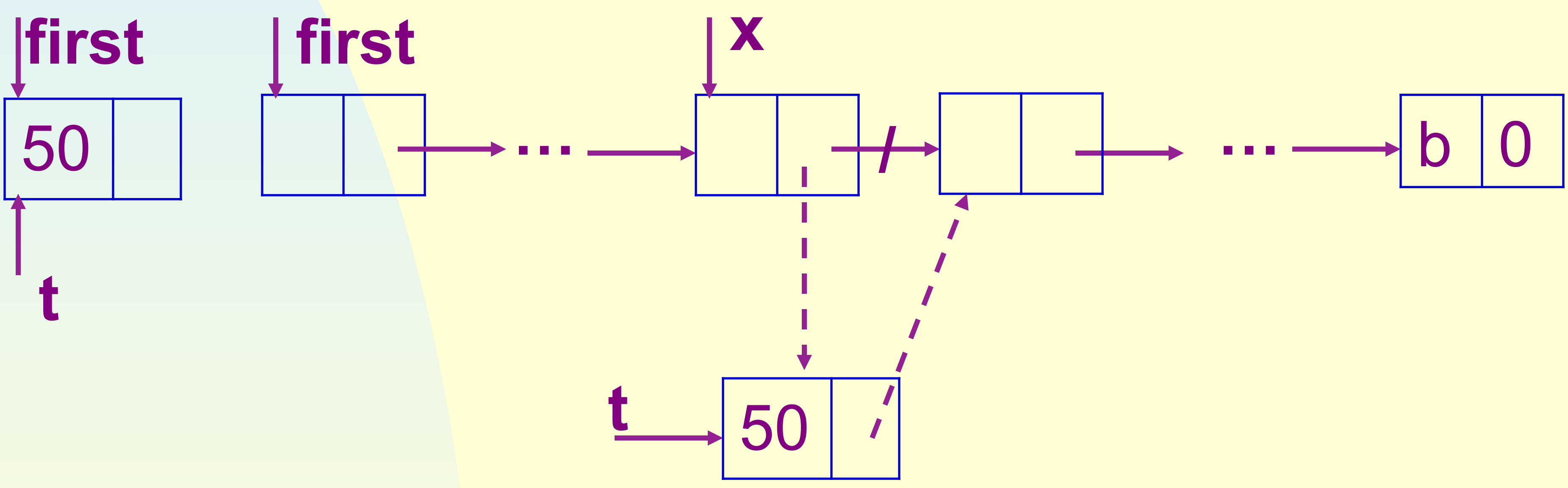
Pointer manipulation in C++:



(a) (b) x=y (c) *x=*y

Chain manipulation:

Example 4.3 insert a node with data field 50 following the node x.



(a) first=0

(b) First !=0

```
void C  a   ::I        50 (C  a   N       *  )
{
   if (       )
      //           a
       →       = new C  a   N     (50,  →       );
   else
      //                                a
              = new C  a   N     (50);
}
```

Exercises: P183-1,2

4.3 The Template Class Chain

We shall enhance the chain class of the previous section to make it more reusable.

4.3.1 Implementing Chains with Templates

```
template <class T>
class C  a    {
public:
   C  a   () {       =0;}; //                              a                    0
   // C  a       a       a             a

private:
   C  a   N      <T> *      ;
};
```

A empty chain of integers intchain would be defined as:

```
   C  a   <int>        a    ;
```

4.3.2 Chain Iterators

A **container** class is a class that represents a data structure that contains or stores a number of data objects.

An **iterator** is an object that is used to access the elements of a container class one by one.

Why we need an iterator?

Consider the following operations that might be performed on a container class C, all of whose elements are integers:

(1) Output all integers in C.

(2) Obtain the sum, maximum, minimum, mean, median of all integers in C.

(3) Obtain the integer x from C such that f(x) is maximum.

These operations have to be implemented as **member functions** of C to access its **private** data members.

Consider the container class Chain<T>, there are, however, some drawbacks to this:

(1) All operations of Chain<T> should preferably be independent of the type of object to which T is initialized. However, operations that make sense for one instantiation of T may not for another instantiation.

(2) The number of operations of Chain<T> can become too large.

Consider the container class Chain<T>, there are, however, some drawbacks to this:

(3) Even if it is acceptable to add member functions, the user would have to learn how to sequence through the container class.

These suggest that container class be equipped with iterators that provide systematic access the elements of the object.

User can employ these iterators to implement their own functions depending upon the particular application.

Typically, an iterator is implemented as a nested class of the container class.

A forward Iterator for Chain

A forward Iterator class for Chain may be implemented as in the next slides, and it is required that ChainIterator be a public nested member class of Chain.

```
class C a I a {
public:
  //                    b C++

  //
  C a I a (C a N <T>* a N = 0)
    {      = a N ; }

  //                    a
  T&   a   *() const { return      → a a;}
  T*   a   →() const { return &      → a a;}
```

```
//
C a I a & a ++() //
   {
              =          →    ;
      return *this;
   }
C a I a & a ++(int) //
   {
      C a I a      = *this;
              =          →    ;
      return    ;
   }
```

```
//      a
bool       a     !=(const C  a   I    a            ) const
    { return            !=       .         ; }
bool       a     == (const C  a   I    a           ) const
    { return            ==       .         ; }

private:
    C  a   N     <T>*          ;
};
```

Additionally, we add the following public member functions to Chain:

```
ChainI era or begin() {return ChainI era or(fir   );}
ChainI era or end() {return ChainI era or(0);}
```

We may initialize an iterator object yi to the start of a chain of integers y using the statement:

```
Chain<int>::ChainI era or   i =   .begin();
```

And we may sum the elements in y using the statement:

```
    m = acc  m  la e(  .begin(),   .end(), 0);
// no e     m doe   no  req  ire acce      o pri  a e member
```

```
Chain ch;
ChainNode * p, *pre;
P = ch.fir   ;
Pre = 0;
While(p != 0)

          co   << p->da a;
          pre = p;
          p = p->ne   ;
```

```
Chain<in > ch;
//////// ini (ch);
Chain<in >::i era or<in > i ;
In      m = 0;
For(I  = ch.begin();i  != ch.end(); i  ++)

    S  m += *i ;
```

Exercises: P194-3, 4

4.3.3 Chain Operations

Operations provided in a reusable class should be enough but not too many.

Normally, include: constructor, destructor, operator=, operator==, operator>>, operator<<, etc.

A chain class should provide functions to insert and delete elements.

Another useful function is reverse that does an in-place reversal of the elements in a chain.

To be efficient, we add a private member last to Chain<T>, which points to the last node in the chain.

InsertBack

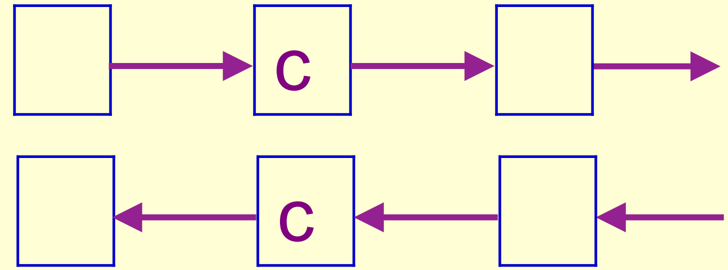
```
template <class T>
void C  a   <T>::I        Ba    (const T&   )
{
   if (      ) { //                        a
      a  →        = new C  a   N      <T>(  );
      a    = a  →       ;
   }
   else        =  a   = new C  a   N      <T>(  );
}
```

The complexity: O(1).

Concatenate

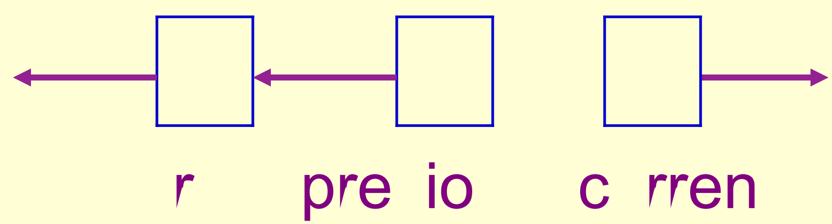
```
template <class T>
void C  a   <T>::C      a      a   (C  a   <T>& b)
{ // b            a                              *this
    if (      )
         {  a   →        = b.       ;  a    = b. a   ;}
    else
         {        = b.       ;  a    = b. a   ; );}
    b.        = b. a    = 0;
}
```

The complexity: O(1).



Reverse

```
template <class T>
void C  a   <T>::R          ()
{ //    a    (a1,.., a ) b            (a  ,.., a1).
   C  a   N      <T> *          =       , *             = 0;
   while (          ) {
      C  a   N      <T> *  =              ; //     a
                 =          ;
               =        →     ;
                →     =  ;
   }
         =            ;
}
```



For a chain with m ≥ 1 nodes, the computing time of Reverse is O(m).

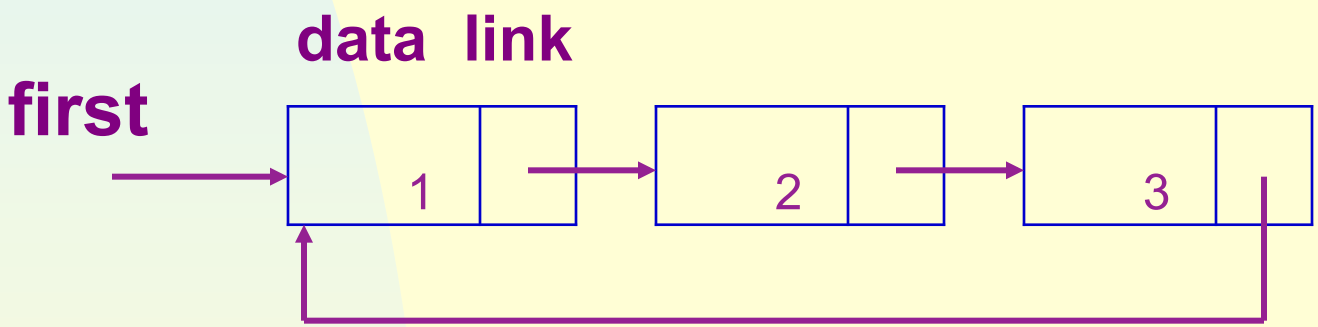
Write an algorithm to construct a Chain from an Array.

Write an algorithm to print all data of a Chain.

Exercises: P184-6

4.4 Circular Lists

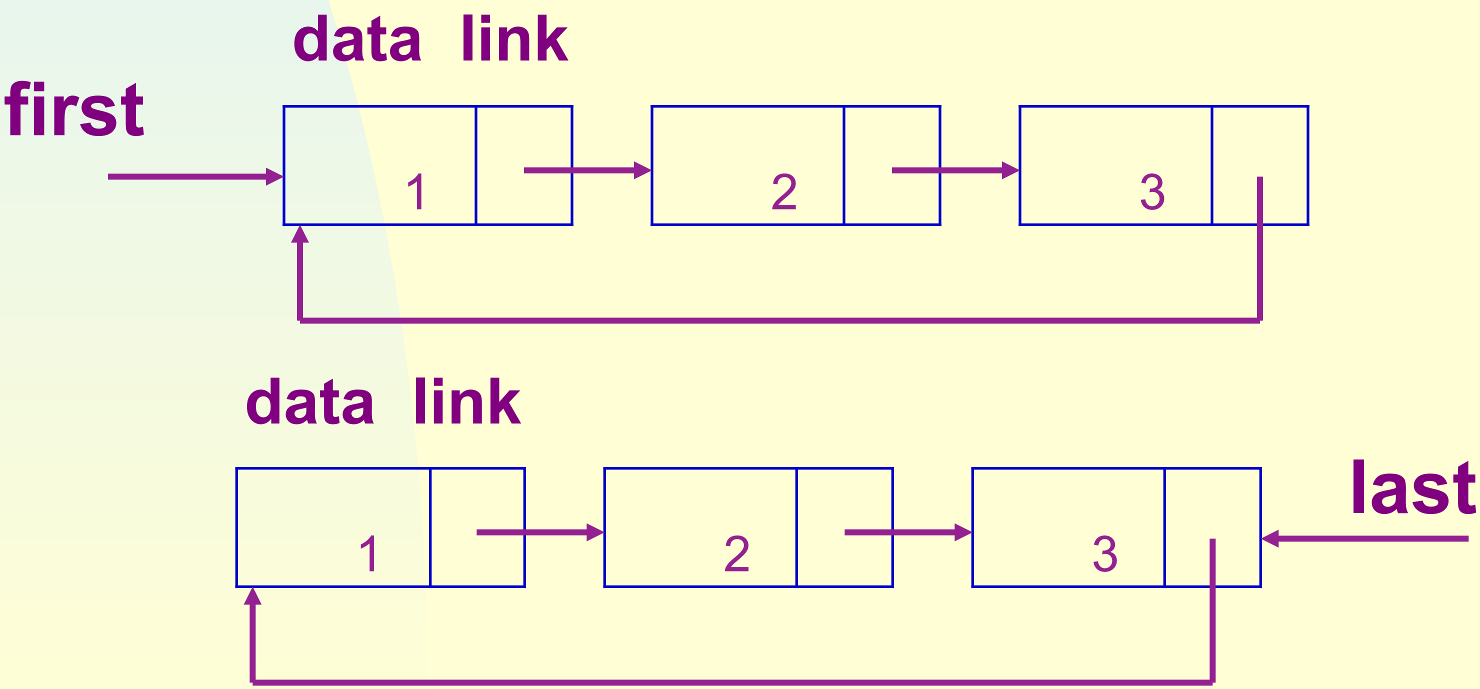
A circular list can be obtained by making the link field point to the first node of a chain.



Consider inserting a new node at the front

We need to change the link field of the node containing x_3.

It is more convenient if the access pointer points to the last rather than the first.



Now we can insert at the front in O(1):

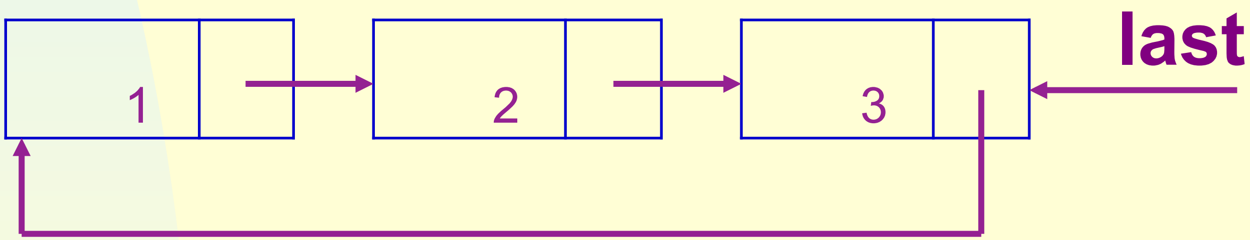
```
template <class T>
void C     a L   <T>::I      F      (const T&   )
{ //                        a                           a      *this,
 //            a                  a                    .
  C  a  N     <T>*      N      = new C  a  N     <T>(  );
  if ( a   ) { //
         N     →       =  a  →      ;
      a  →      =      N     ;
   }
  else {  a   =      N     ;     N     →      =      N     ;}
}
```

To insert at the **back**,

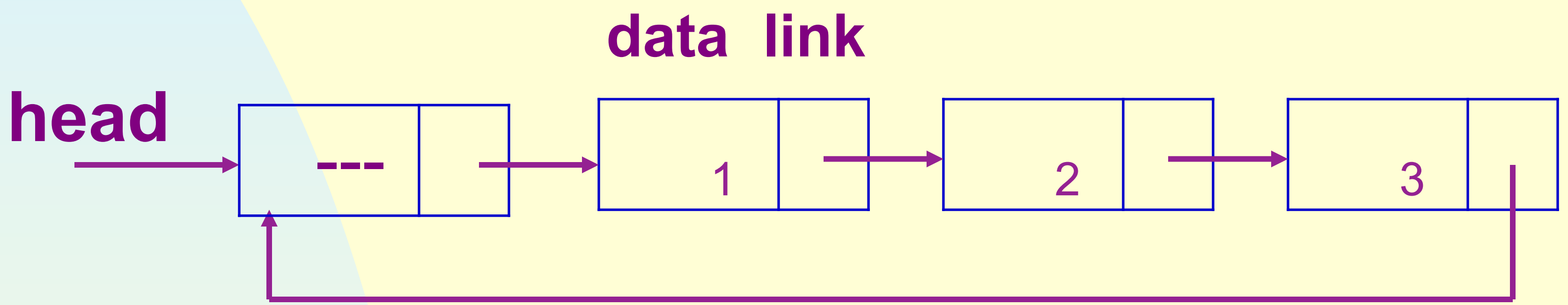
we only need to add the statement

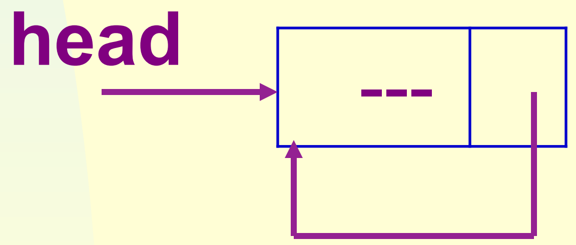
last = newNode;

to the if clause of **InsertFront**, the complexity is still O(1).



To avoid handling empty list as a special case introduce a dummy **head** node:.





empty circular list

4.5 Available Space lists

the time of destructors for chains and circular lists is **linear** in the length of the chain or list.

it may be reduced to O(1) if we maintain our own chain of free nodes.

the available space list is pointed by **av**.

av be a static class member of **CircularList<T>** of type **ChainNode<T>** *, initially, **av** = 0.

only when the **av** list is empty do we need use **new**.

We shall now use CircularList<T>::GetNode instead of using new:

```
template <class T>
C  a   N      <T>* C        a L   <T>::G   N      ( )
{ //              a
    C  a   N      <T> *   ;
    if (a  ) {    = a  ; a   = a →       ;}
    else    = new C  a   N      <T>;
    return   ;
}
```

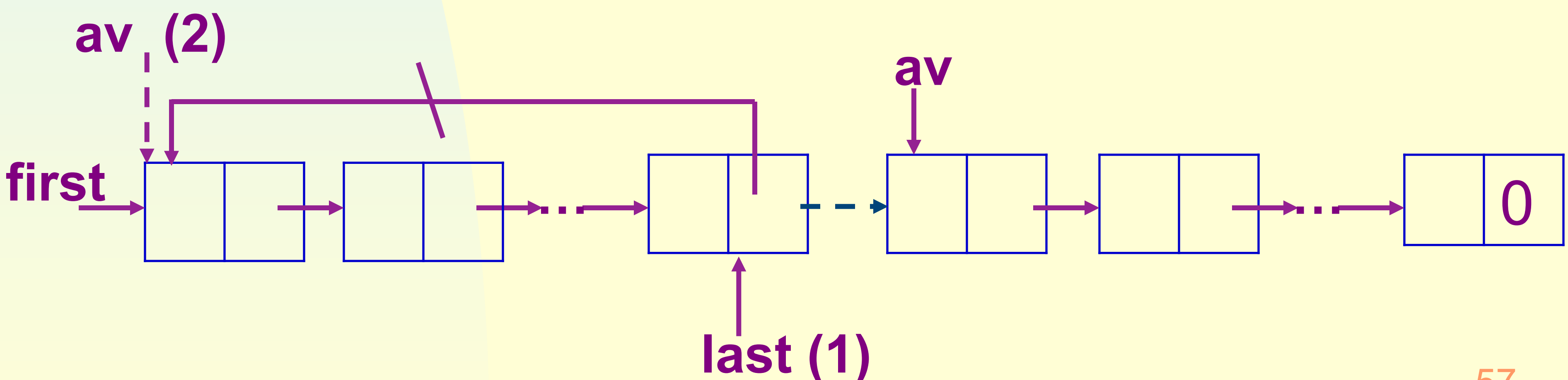
A circular list may be destructed in O(1):

```
template <class T>
void C      a L   <T>::  C      a L   ()
{ //                        a      .
    if ( a   ) {
        C  a   N      <T> *        =  a  →      ;
         a  →       = a  ;   // (1)
        a  =       ; // (2)
         a   = 0;
    }
}
```

As shown in the next slide:

A circular list may be deleted in O(1):

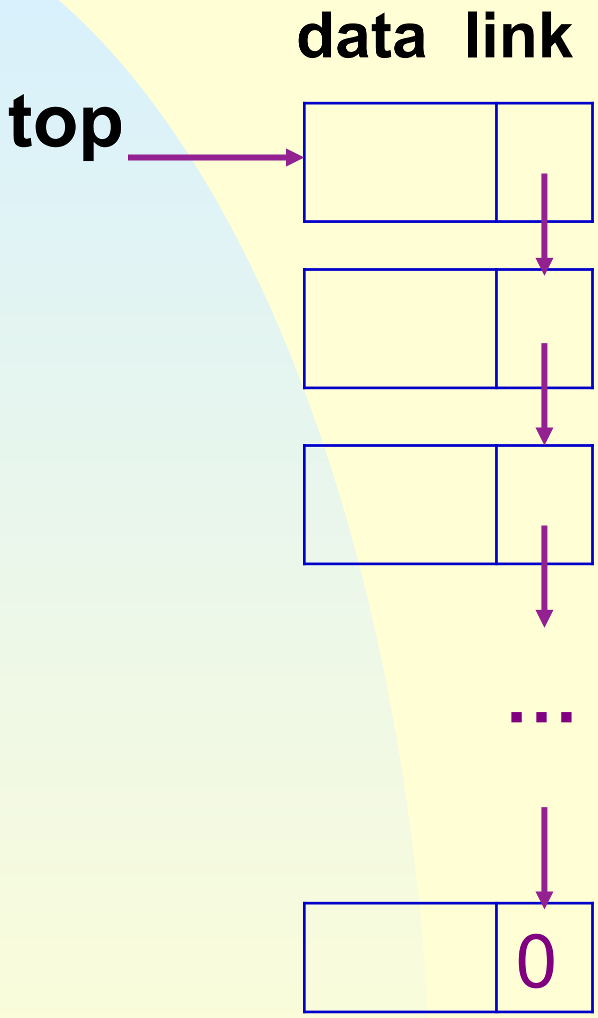
```
template <class T>
void C      a L   <T>:: C      a L   ()
{ //                    a       .
   if ( a  ) {  C  a  N     <T> *       =  a  →      ;
         a  →       = a  ;  // (1)
        a  =        ; // (2)
         a   = 0;
   }
}
```



A chain may be deleted in O(1) if we know its **first** and **last** nodes:

```
template <class T>
Chain<T>::  Chain()
{ // delete the chain
    if (first) {
        last→link = av;
        av = first;
        first = 0;
    }
}
```

4.6 Linked Stacks and Queues



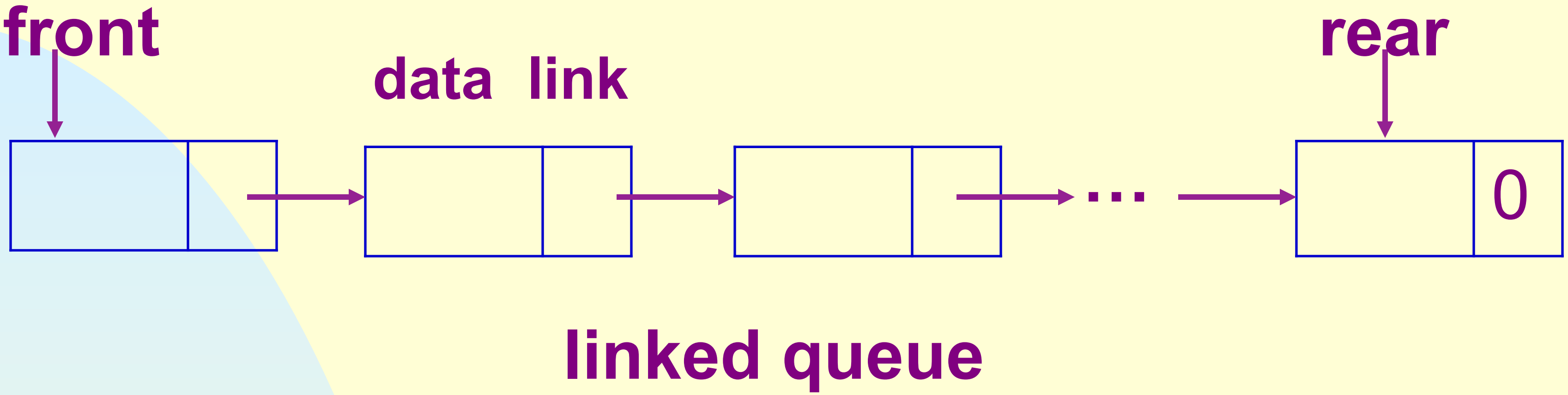
linked stack

Assume the LinkedStack class has been declared as friend of ChainNode<T>.

```
template <class T>
class LinkedStack {
public:
   LinkedStack() { top=0;}; // constructor initializing top to 0
   // LinkedStack manipulation operations

private:
   ChainNode<T> *top;
};
```

```
template <class T>
void L        S a   <T>::P     (const T&   ) {
        = new C  a   N      (  ,      );
}
template <class T>
void L        S a   <T>::P    ()
{ //                                   a    .
   if (I  E        ()) throw   S a              . Ca               .  ;
   C  a   N      <T> *      N      =      ;
        =      →       ;
   delete      N      ;
}
```

The functions IsEmpty and Top are easy to implement, and are omitted.



The functions of LinkedQueue are similar to those of LinkedStack, and are left as exercises.

Exercises: P201-2

4.7 Polynomials

4.7.1 Polynomial Representation

Since a polynomial is to be represented by a list, we say Polynomial is IS-IMPLEMENTED-IN-TERMS-OF List.

Definition: **a data object of Type A IS -IMPLEMENTED-IN-TERMS-OF a data object of Type B if the Type B object is central to the implementation of Type A object. ---Usually by declaring the Type B object as a data member of the Type A object.**

$A(x) = a_m x^{em} + a_{m-1} x^{em-1} + , \quad , + a_1 x^{e1}$

Where $a_i \neq 0, \quad e_m > e_{m-1} >, \quad , e_1 \geq 0$

Make the chain poly a data member of Polynomial.

Each ChainNode will represent a term. The template T is instantiated to struct Term:

```
T
// a          b          T     a      b    b      a
          ;
         ;
  T     S  (       ,       )          = ;     = ;          *      ; ;
;
```

```
class P          a  {
public:
   //    b
private:
   C  a   <T     >        ;
};
```

4.7.2 Adding Polynomials

To add two polynomials a and b, use the chain iterators ai and bi to move along the terms of a and b.

```
1 P          a P          a ::operaor+ (const P          a & b) const
2 { // *this (a) a    b a   a        a
3   T            ;
4   C  a  <T     >::C  a  I   a    a  =      .b     (),
5                                  b  = b.     .b     ();
6   P           a   ;
```

```
while (a  !=         .       () && b  != b.         .       ()) { //
    if (a →         == b →       ) {
                int        = a →        + b →       ;
        if (       )  .       .I       Ba    (         .S   (       , b →       );
         a ++; b ++; //
    }
    else if (a →        < b →       ) {
            .       .I       Ba    (        .S   (b →        , b →       ));
          b ++;  //                      b
    }
    else {
            .       .I       Ba    (        .S   (a →        , a →       ));
          a ++;  //                      a
     }
}
```

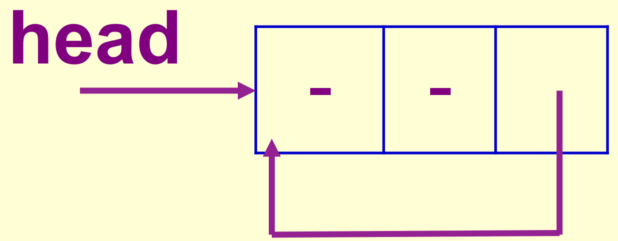
```
while (a  !=        .      ()) { //                    a
      .      .I       Ba    (       .S   (a →        , a →      ));
   a ++;
}
while (b  != b.       .      ()) { //                    b
      .      .I       Ba    (       .S   (b →        , b →      ));
   b ++;
 }
return   ;
}
```

Analysis:

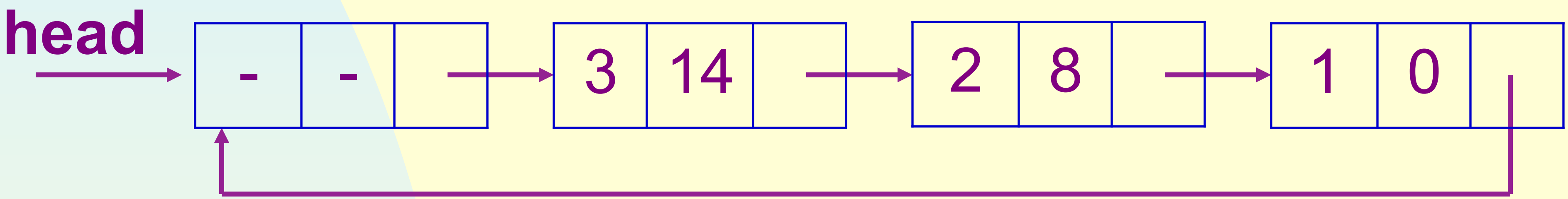
Assume a has m terms, b has n terms. The computing time is O(m+n).

4.7.3 Circular List Representation of Polynomials

Polynomials represented by circular lists with head node are as in the next slide:



(a) Zero polynomial



(b) $3x^{14} + 2x^{8} + 1$

Adding circularly represented polynomials

The exp of the head node is set to 1 to push the rest of a or b to the result.

Assume the begin() function for class CircularListWithHead return an iterator with its current points to the node head→link.

```
P        a  P        a ::operaor+(const P        a & b) const
{ // *this (a) a    b a   a        a
    T           ;
    C      a L   W   H  a  <T    >::I   a    a  =       .b      (),
                                                     b  = b.     .b      ();
    P           a    ; //a                                     a  →      = -1
    while (1) {
        if (a →      == b →     ) {
            if (a →      == -1) return   ;
            int        = a →       + b →      ;
            if (      )  .     .I      Ba   (       .S   (      , a →     );
            a ++; b ++; //
    }
```

```
        else if (a →       < b →      ) {
                .      .I      Ba   (      .S  (b →        , b →      ));
                b ++;  //                 b
        }
        else {
                .      .I      Ba   (      .S  (a →        , a →      ));
                a ++;  //                 a
        }
    }
}
```

Experiment: P209-5

4.10 Doubly Linked Lists

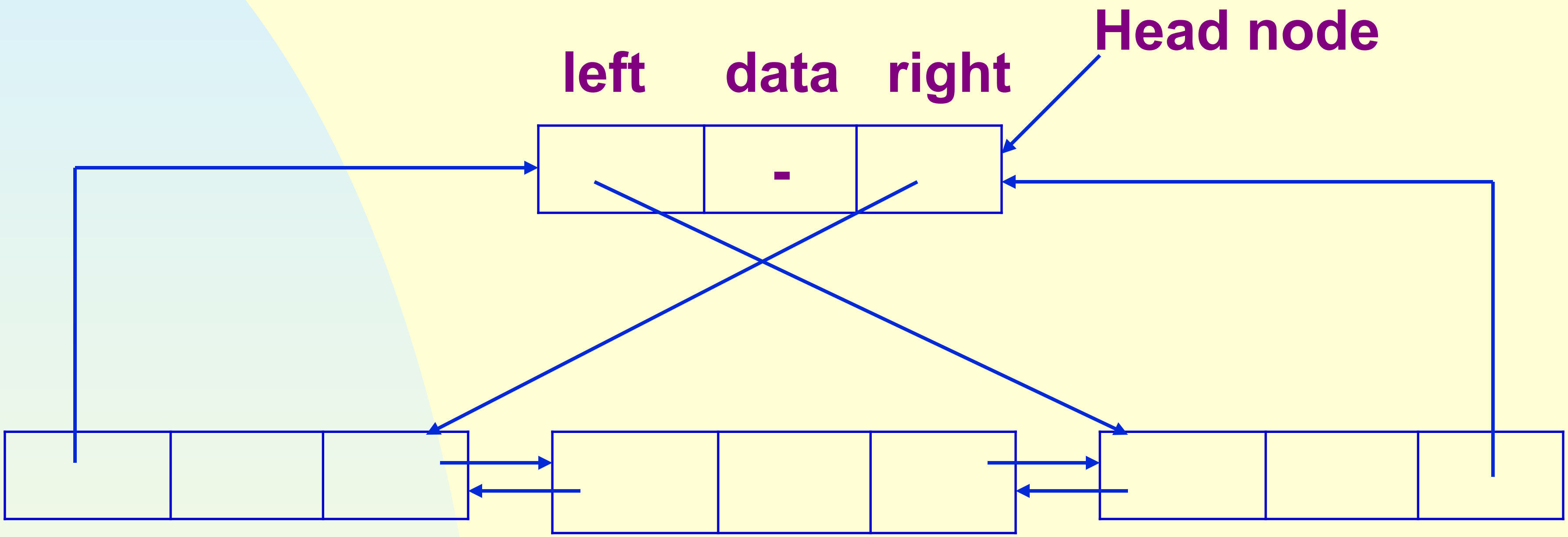
Difficulties with singly linked list:

can easily move only in one direction
not easy to delete an arbitrary node
requires knowing the preceding node

A node in doubly linked list has at least 3 field: data, left and right, this makes moving in both directions easy.

left	data	right

A doubly linked list may be circular. The following is a doubly linked circular list with head node:



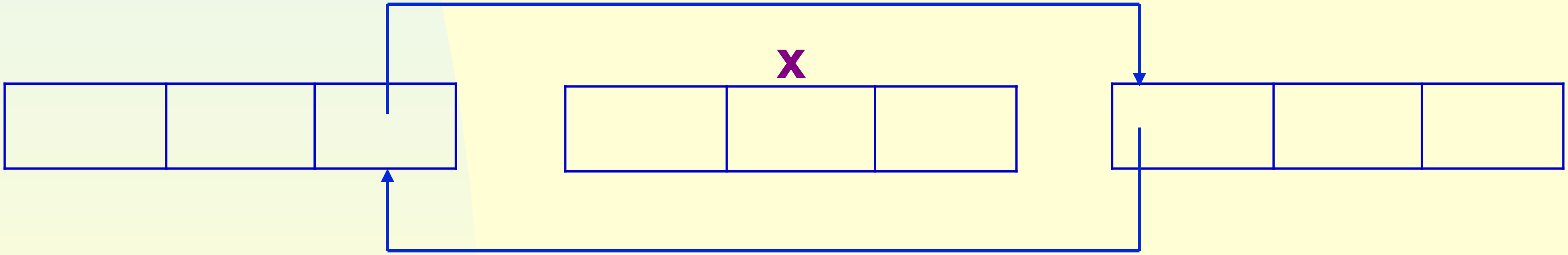
Suppose p points to any node, then

$p == p \rightarrow left \rightarrow right == p \rightarrow right \rightarrow left$

```
class Db L    ;

class Db L   N       {
friend class Db L   ;
private:
    int   a a;
    Db L   N       *      , *        ;
};

class Db L    {
public:
    // L        a        a               a

private:
    Db L   N       *      ; //                    a
};
```

Delete

```
void DbL    ::D       (DbL   N       *   )
{
   if(   ==       ) throw   D              a                    ;
   else  {
       →     →       =  →      ;
       →       →     =  →    ;
      delete   ;
   }
}
```



Insert

void Db L ::I (Db L N * , Db L N *)

{ //

→ = **;** // (1)

Exercises: P225-2

1. Write an algorithm to construct a Chain from an Array.

2.Given a sorted single linked list L = <a_1, …., a_n>, where a_i.data<= a_j.data (i < j).
Try to write an algorithm of inserting a new data element X to L, and analysis its complexities.

3.Given a linear list L = <a_1, …., a_n>, implemented by a single linked list.
Delete data a_i with Time Complexity O(1). We have a pointer to node(a_i).

```
Node * fir    = 0, *la    =0;
In  [n];
For(in  = 0; I < n; i++)


    In  da a = a[i];
    Node * p = ne    Node(da a);
    If(fir    == 0)
        Fir    = la    = p;
    El  e
        La   ->ne    = p;
        La    = p;
```

```
Node * c rren = fir , *pre = 0;

While ( c rren != 0 && c rren -> da a < X)

Pre = c rren ;
c rren = c rren ->ne ;
```