Fakultas Informatika
School of Computing
Telkom University

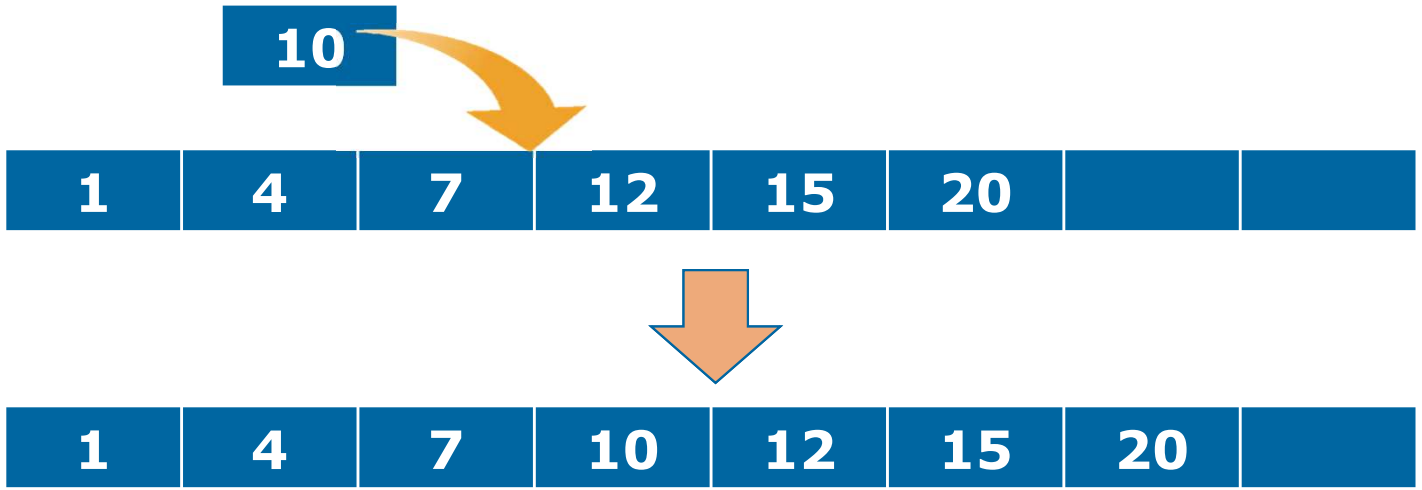
CDK2AAB4
STRUKTUR DATA

Singly Linked List

Introduction

Exercise

- Create an algorithm to insert a number into an ordered array of integer so that the array result remain ordered



Insert into a sorted Array

Algorithm

```
while i < n and  tab[i] < x do
    i ← i + 1
endwhile
temp1 ← tab[i]
tab[i] ← x
for j ← i+1 to n do
    temp2 ← tab[j]
    tab[j] ← temp1
    temp1 ← temp2
endfor
```

inisialisasi?

Insert into a sorted Array

Algorithm

```
i ← 0
while i < n and  tab[i] < x do
    i ← i + 1
endwhile
temp1 ← tab[i]
tab[i] ← x
for j ← i+1 to n do
    temp2 ← tab[j]
    tab[j] ← temp1
    temp1 ← temp2
endfor
```

inisialisasi?

size update?

Insert into a sorted Array

Algorithm

```
i ← 0
while i < n and  tab[i] < x do
    i ← i + 1
endwhile
temp1 ← tab[i]
tab[i] ← x
for j ← i+1 to n do
    temp2 ← tab[j]
    tab[j] ← temp1
    temp1 ← temp2
endfor
n ← n + 1     { update array size }
```

inisialisasi?

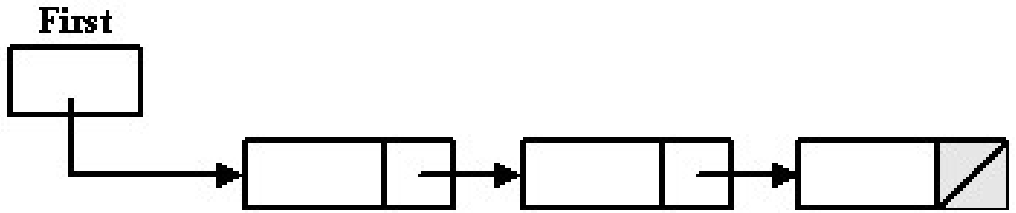
size update?

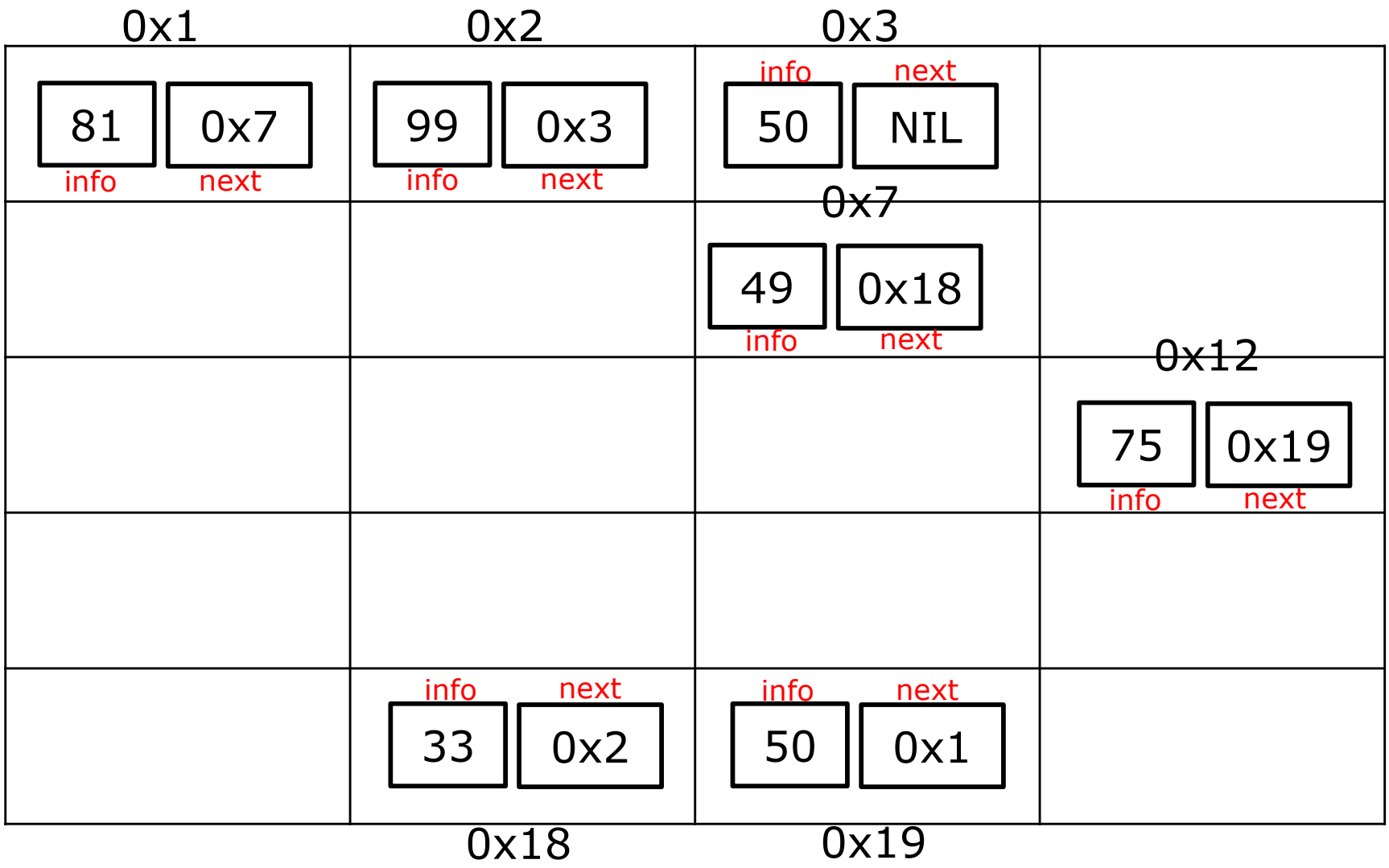
Array

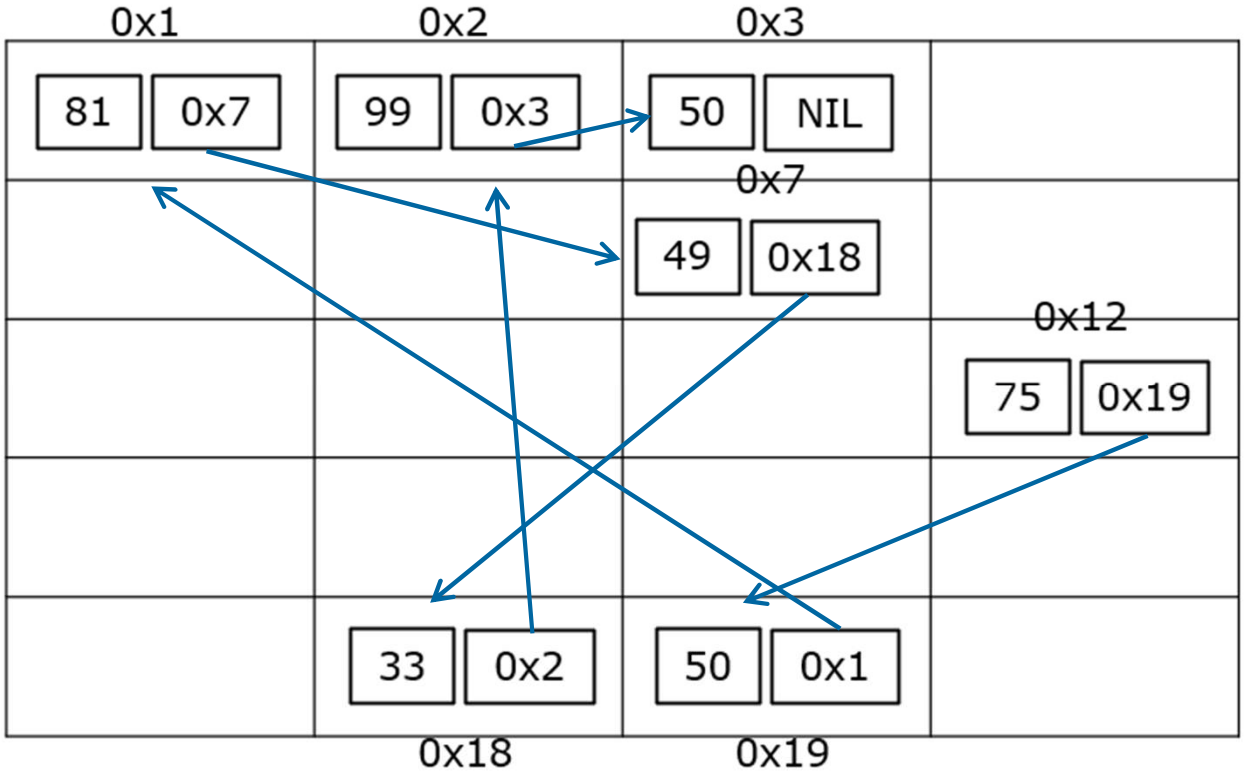
- Very useful data structure
- Limitation
  - its size should be known at compilation time
  - the data in the array are separated in computer memory by the same distance, which means that inserting an item inside the array requires shifting other data in this array.

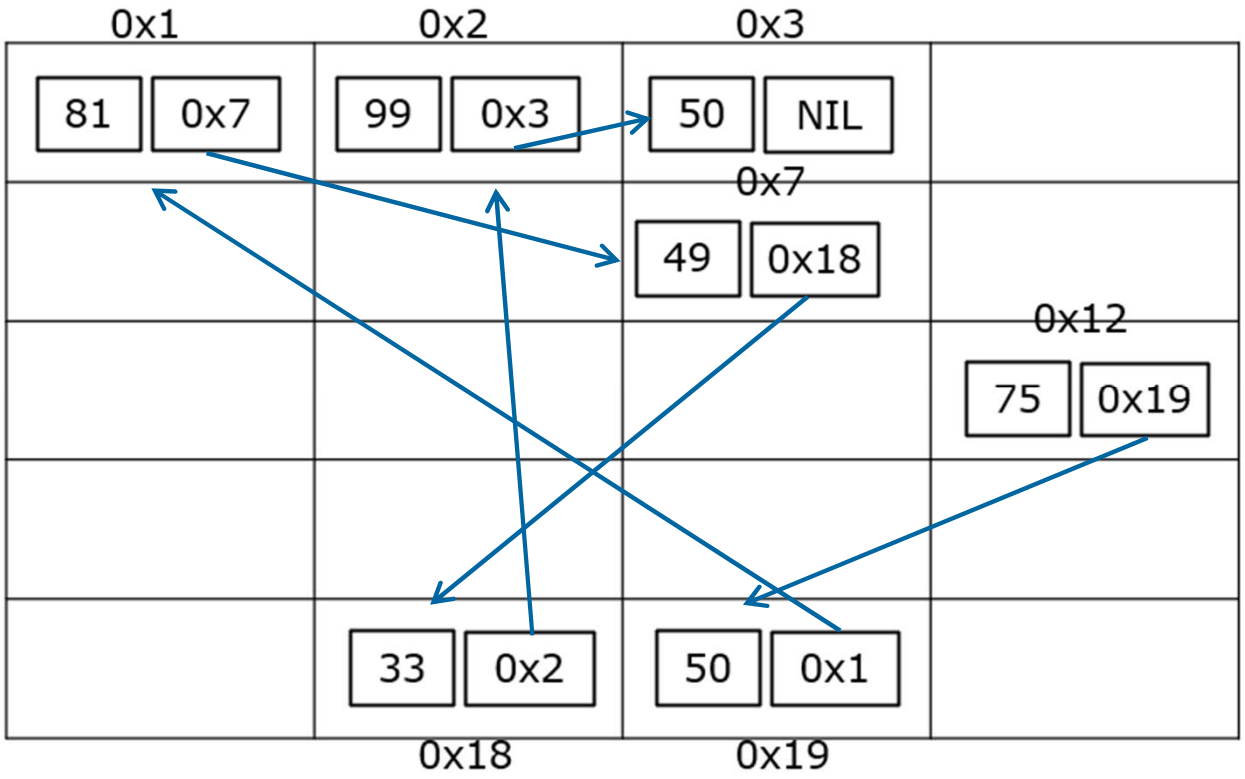
Troublesome, isn't it?

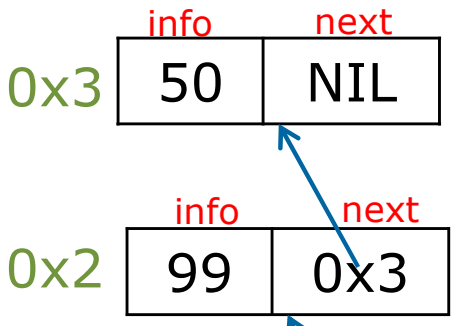
- Those limitations can be overcome by using *linked structures.*
- A linked structure is a collection of nodes storing data and links to other nodes.
  - the most flexible implementation is by using pointers.

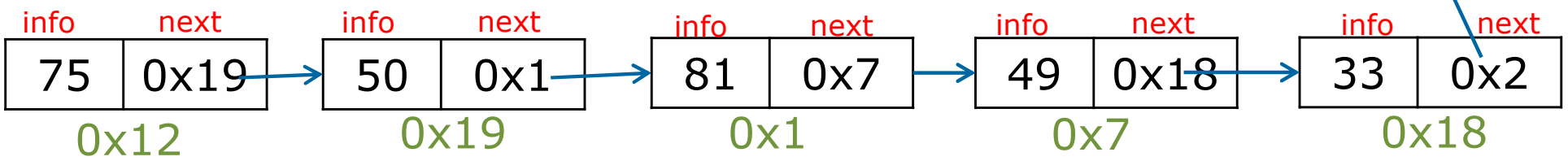


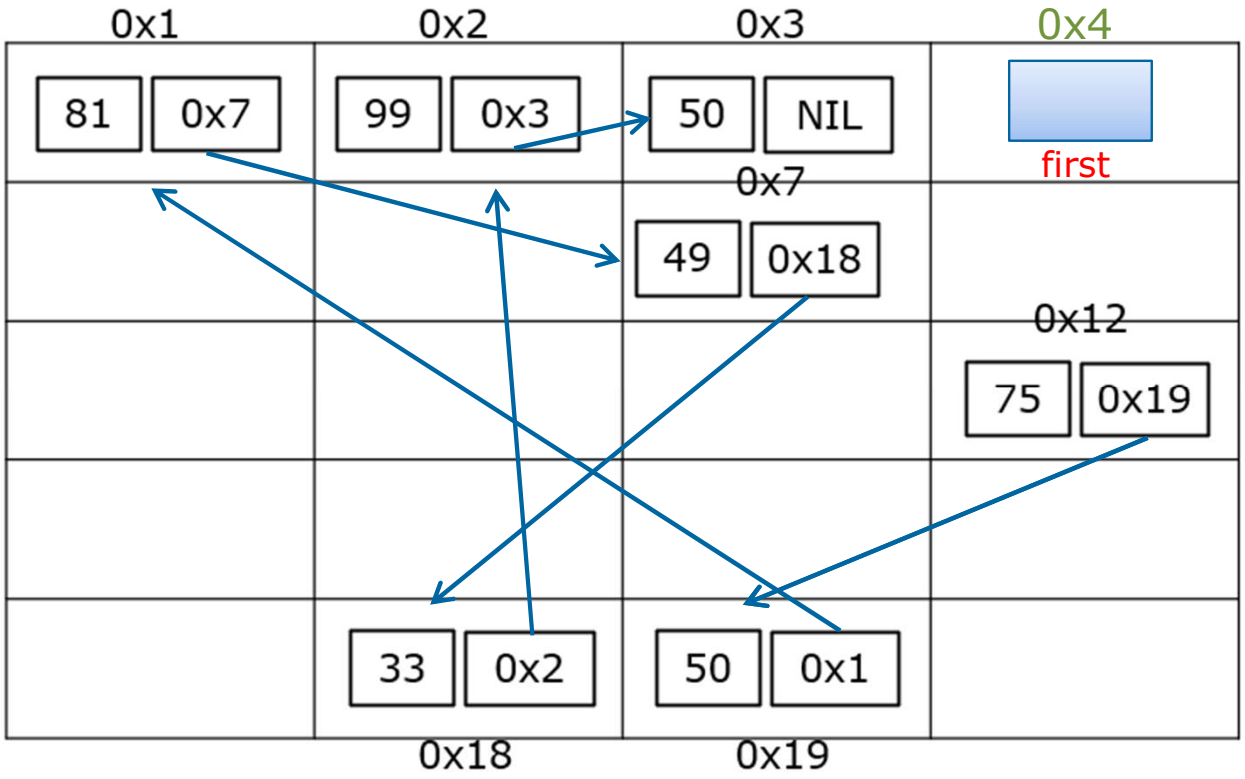

0x1
0x2
0x3
81
0x7
info
next
99
0x3
info
next
info
next
50
NIL
0x7
49
0x18
info
next
0x12
75
0x19
info
next
info
next
33
0x2
info
next
50
0x1
0x18
0x19


0x1
0x2
0x3
81
0x7
99
0x3
50
NIL
0x7
49
0x18
0x12
75
0x19
33
0x2
50
0x1
0x18
0x19


0x1
0x2
0x3
81
0x7
99
0x3
50
NIL
0x7
49
0x18
0x12
75
0x19
33
0x2
50
0x1
0x18
0x19

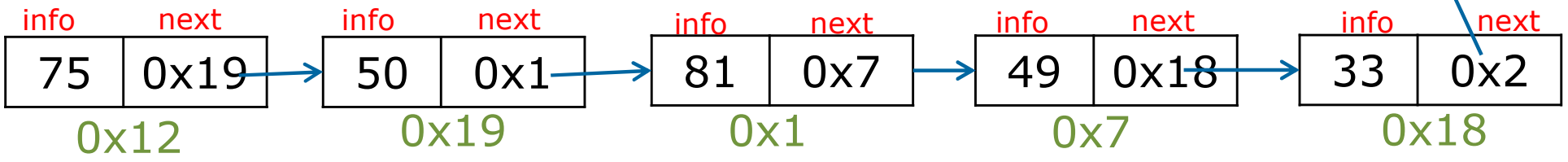

info
next
0x3
50
NIL
info
next
0x2
99
0x3

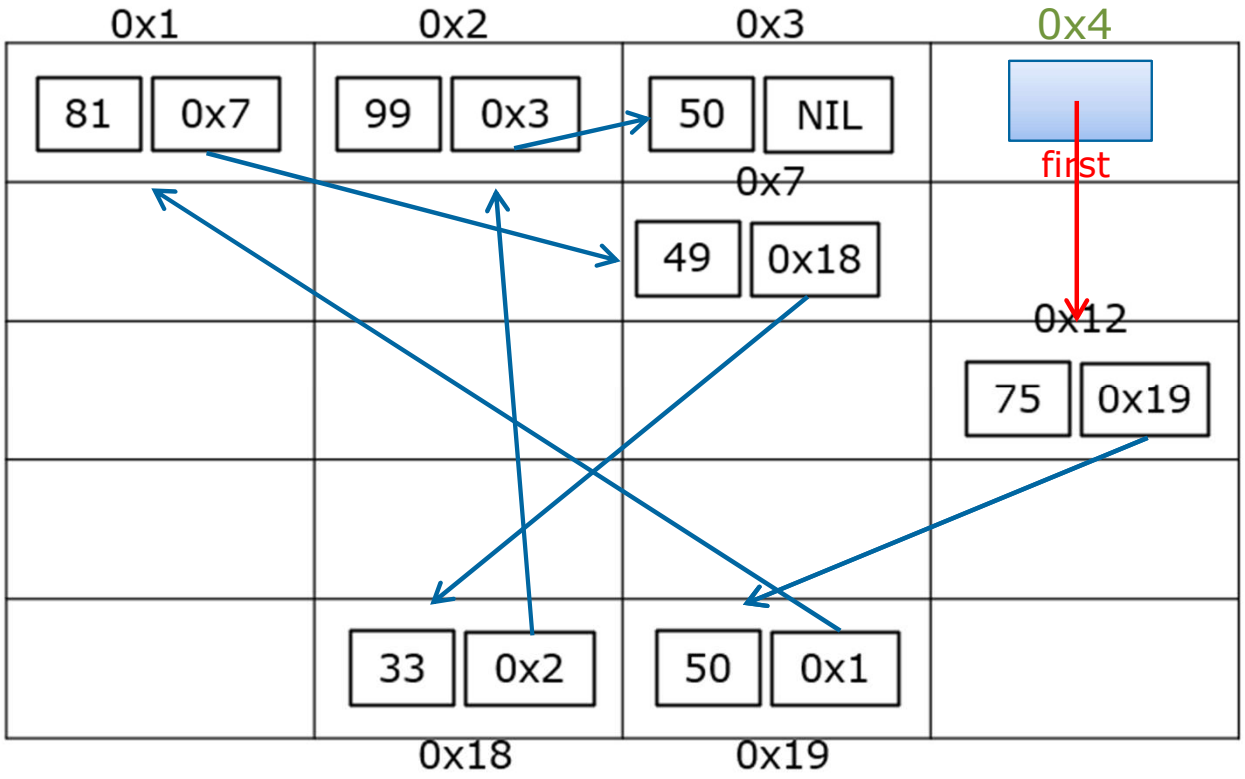

info
next
75
0x19
0x12
info
next
50
0x1
0x19
info
next
81
0x7
0x1
info
next
49
0x18
0x7
info
next
33
0x2
0x18


0x1
0x2
0x3
0x4
81
0x7
99
0x3
50
NIL
first
0x7
49
0x18
0x12
75
0x19
33
0x2
50
0x1
0x18
0x19


info
next
0x3
50
NIL

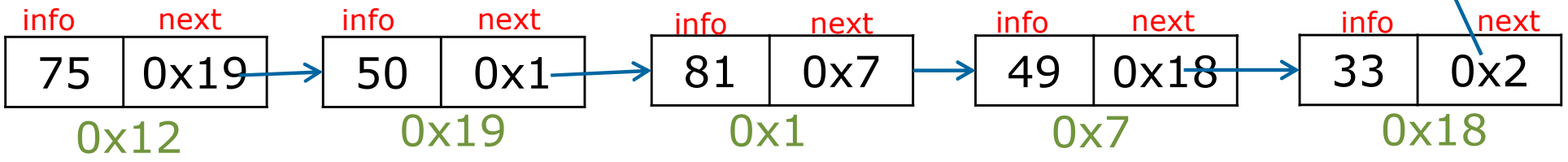

info
next
0x2
99
0x3

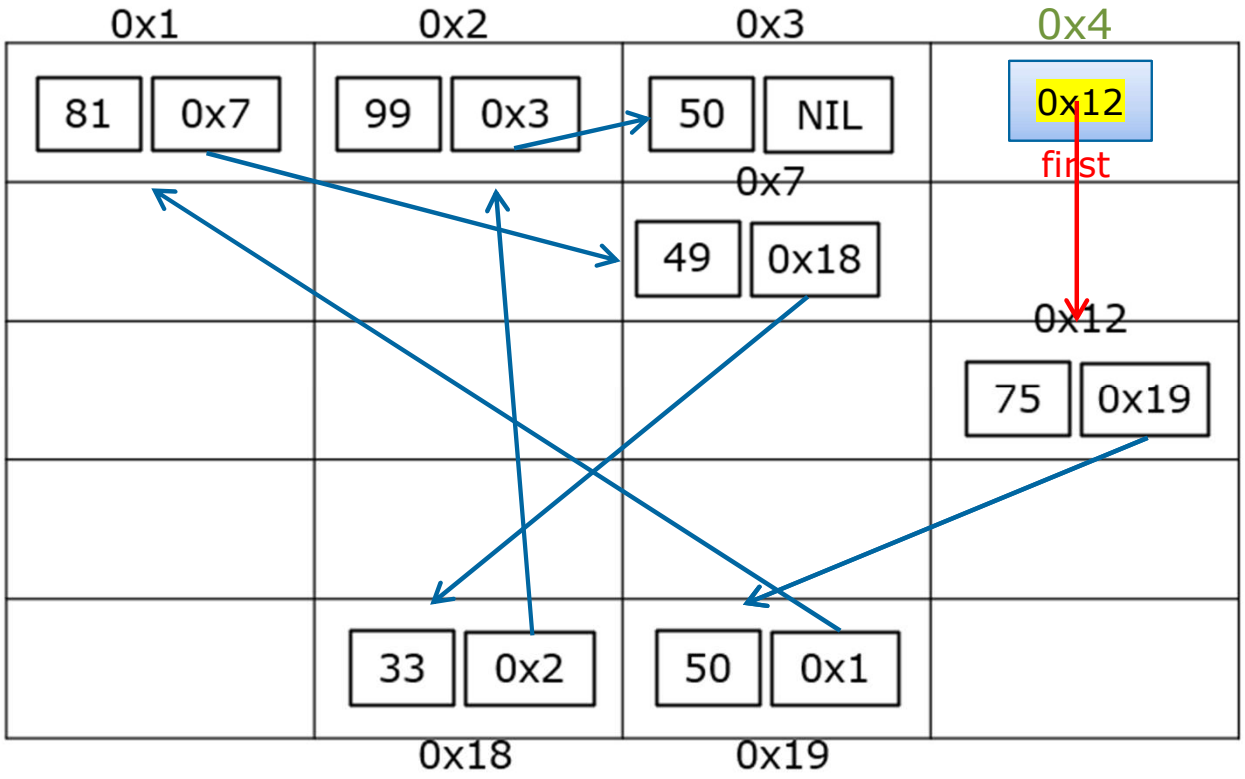

info
next
75
0x19
0x12
info
next
50
0x1
0x19
info
next
81
0x7
0x1
info
next
49
0x18
0x7
info
next
33
0x2
0x18

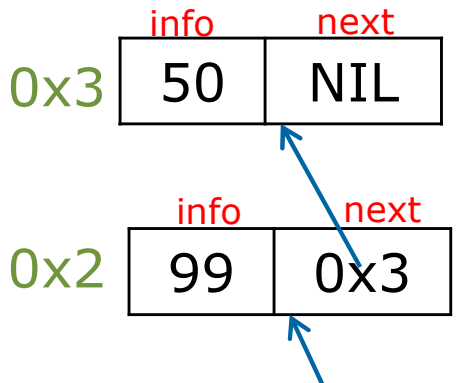

0x1
0x2
0x3
0x4
81
0x7
99
0x3
50
NIL
first
0x7
49
0x18
0x12
75
0x19
33
0x2
50
0x1
0x18
0x19

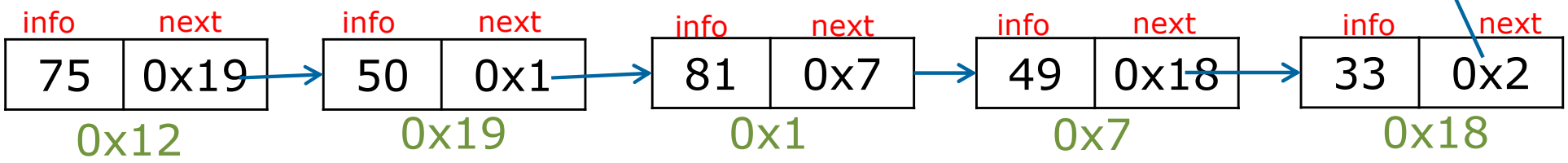

info
next
0x3
50
NIL

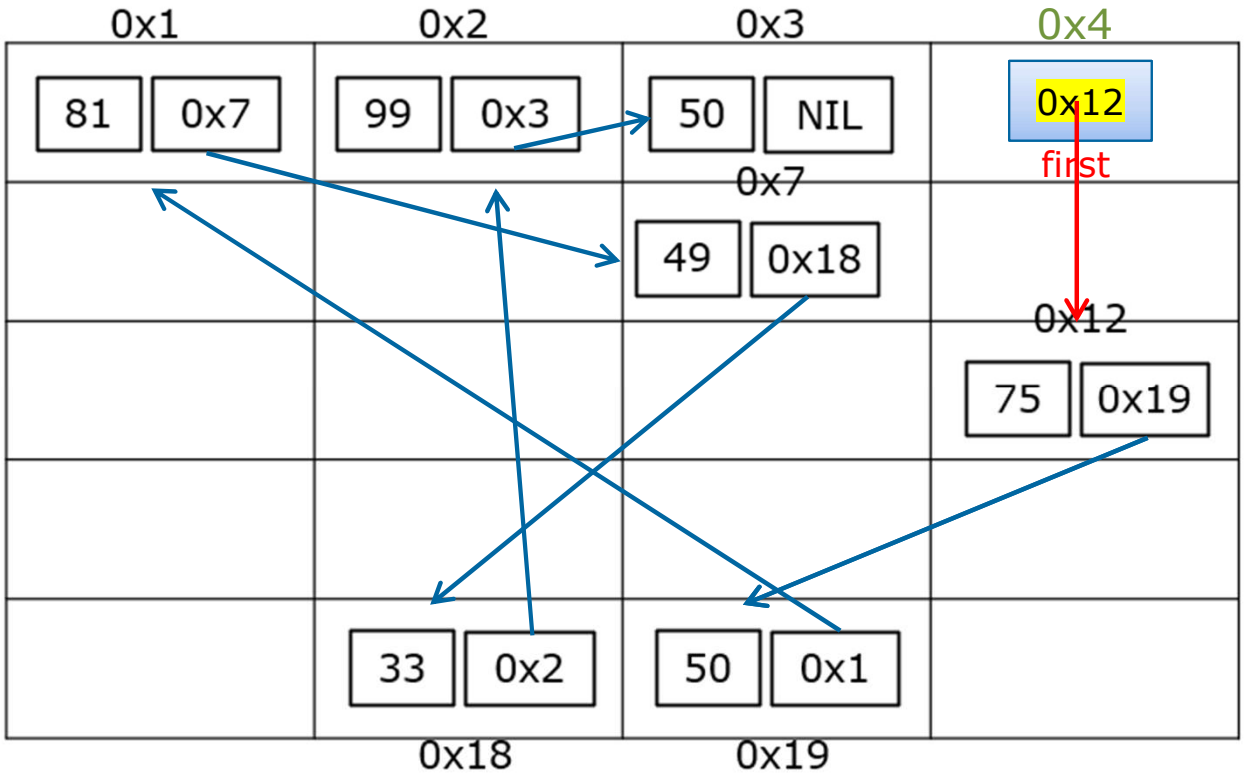

info
next
0x2
99
0x3

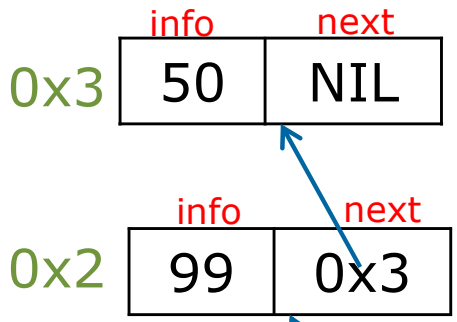

info
next
75
0x19
0x12
info
next
50
0x1
0x19
info
next
81
0x7
0x1
info
next
49
0x18
0x7
info
next
33
0x2
0x18

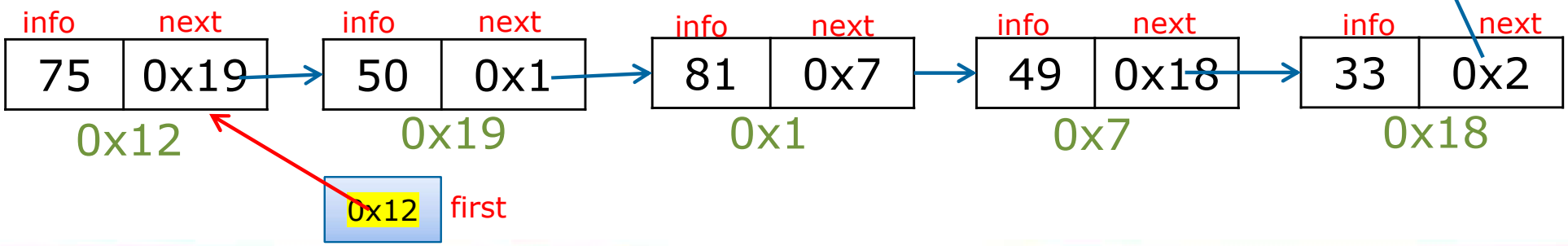

0x1
0x2
0x3
0x4
81
0x7
99
0x3
50
NIL
0x12
first
0x7
49
0x18
0x12
75
0x19
33
0x2
50
0x1
0x18
0x19

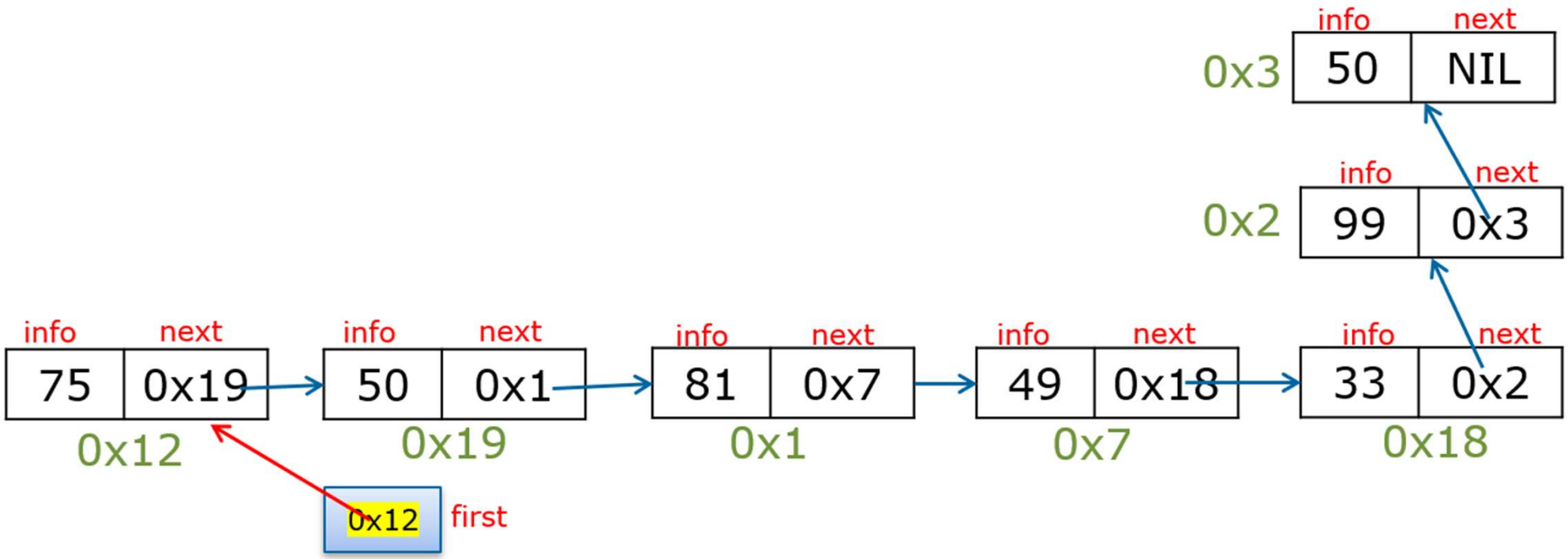

info
next
0x3
50
NIL
info
next
0x2
99
0x3

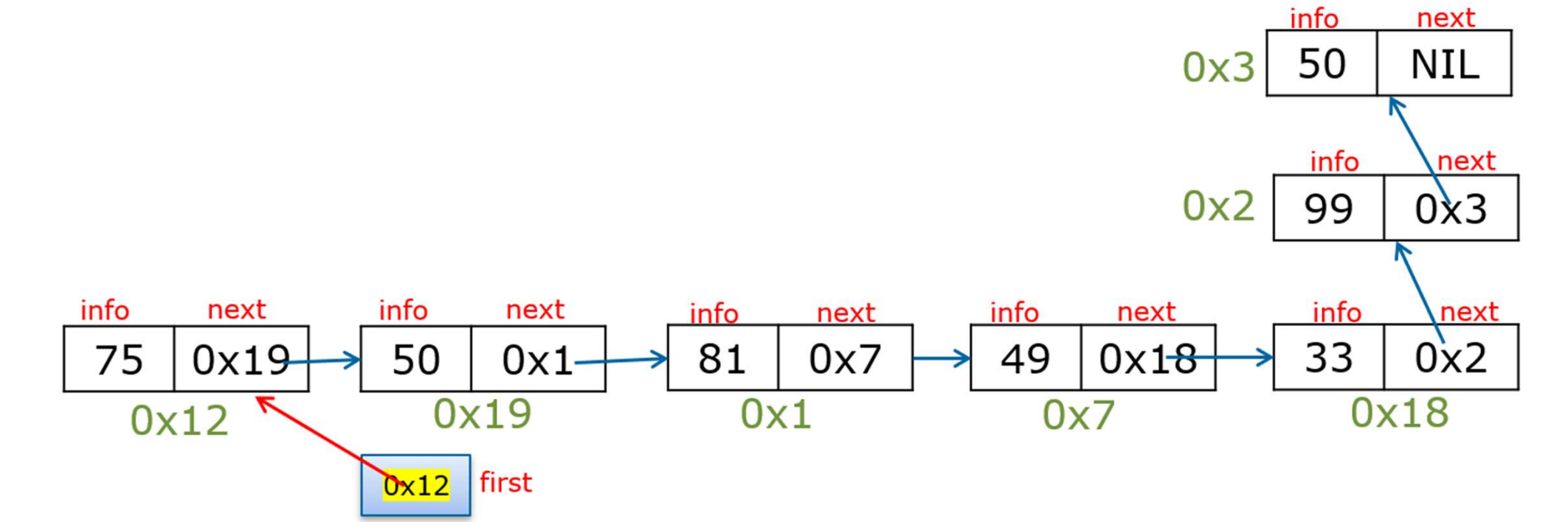

info
next
75
0x19
0x12
info
next
50
0x1
0x19
info
next
81
0x7
0x1
info
next
49
0x18
0x7
info
next
33
0x2
0x18

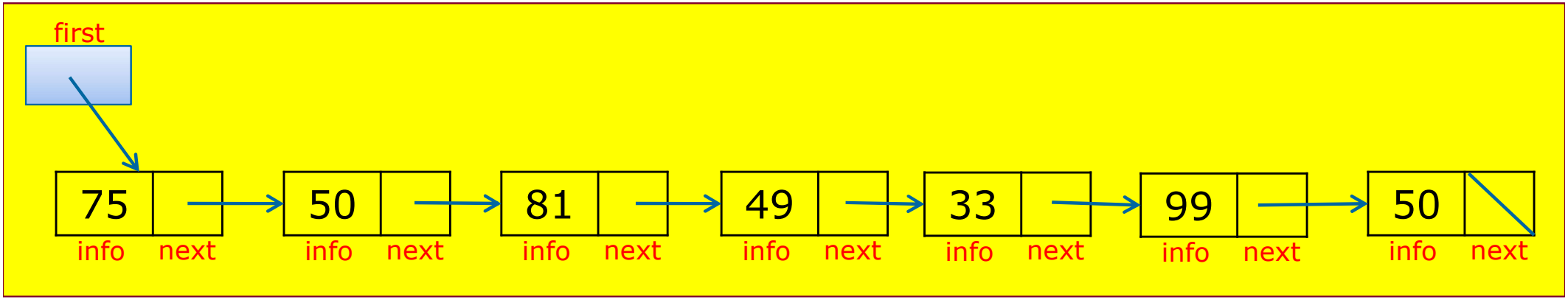

0x1
0x2
0x3
0x4
81
0x7
99
0x3
50
NIL
0x12
first
0x7
49
0x18
0x12
75
0x19
33
0x2
50
0x1
0x18
0x19

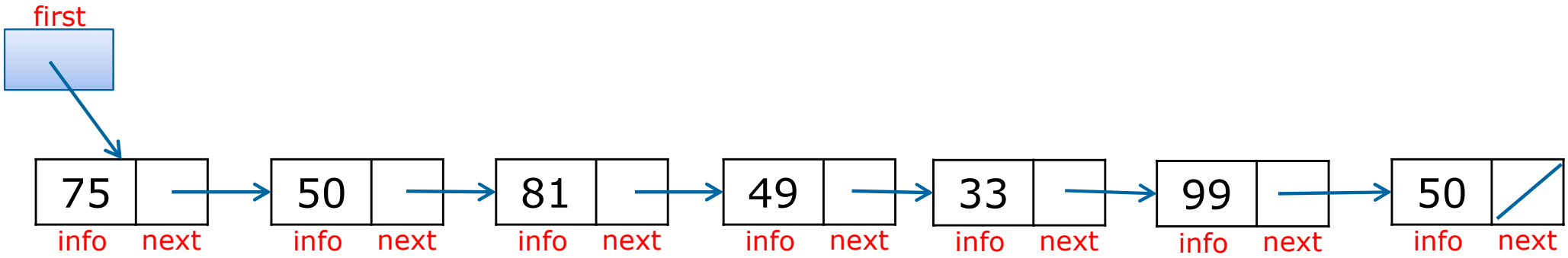

info
next
0x3
50
NIL
info
next
0x2
99
0x3

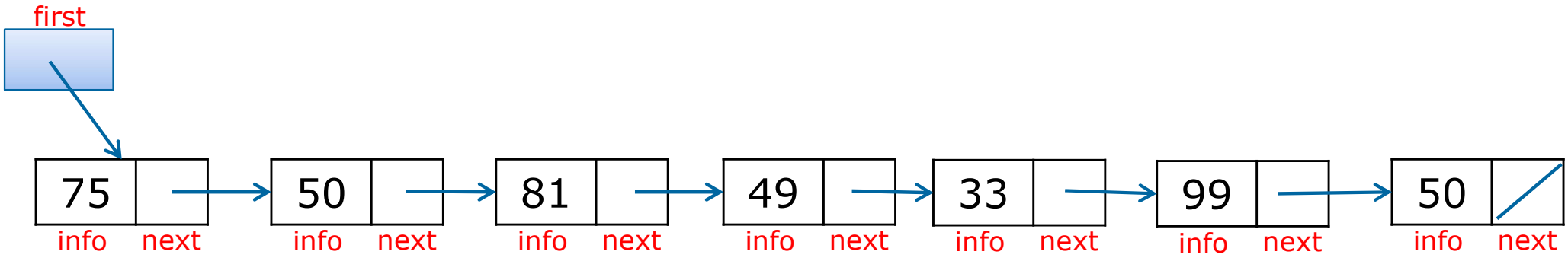

info
next
75
0x19
0x12
info
next
50
0x1
0x19
info
next
81
0x7
0x1
info
next
49
0x18
0x7
info
next
33
0x2
0x18
0x12
first

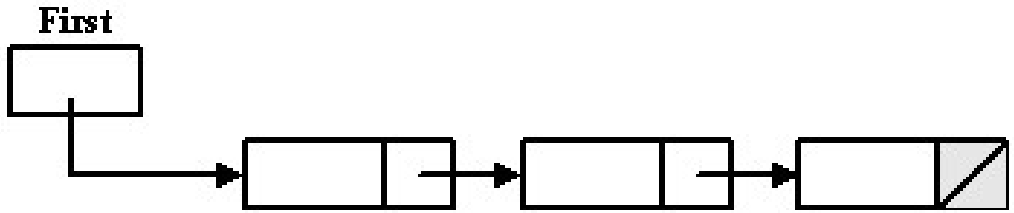

info
next
0x3
50
NIL
info
next
0x2
99
0x3
info
next
75
0x19
0x12
info
next
50
0x1
0x19
info
next
81
0x7
0x1
info
next
49
0x18
0x7
info
next
33
0x2
0x18
0x12
first


info
next
0x3
50
NIL
info
next
0x2
99
0x3
info
next
75
0x19
0x12
info
next
50
0x1
0x19
info
next
81
0x7
0x1
info
next
49
0x18
0x7
info
next
33
0x2
0x18
0x12
first


first
75
info
next
50
info
next
81
info
next
49
info
next
33
info
next
99
info
next
50
info
next


first
75
info
next
50
info
next
81
info
next
49
info
next
33
info
next
99
info
next
50
info
next


first
75
info
next
50
info
next
81
info
next
49
info
next
33
info
next
99
info
next
50
info
next

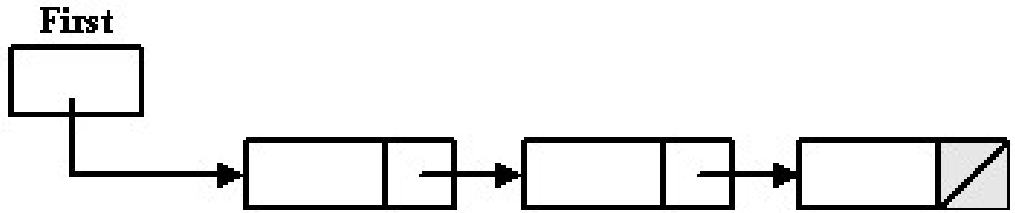

First

Linked List

- A linked list is a data structure used for storing collections of data.
- A linked list has the following properties:
  - Successive elements are connected by pointers.
  - The last element points to NIL.
  - Can grow or shrink in size during execution of a program.
  - Can be made just as long as required (until systems memory exhausts).
  - Does not waste memory space (but takes some extra memory for pointers). It allocates memory as list grows.

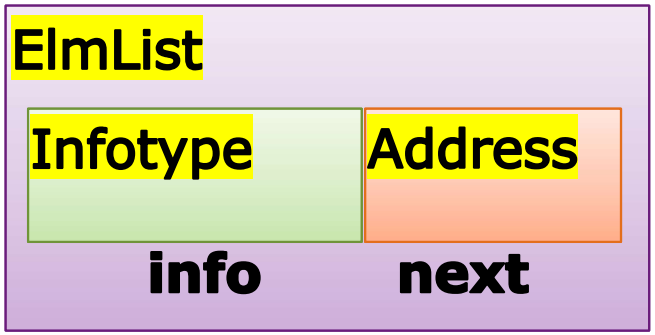
Structure

- Consists of nodes/elements



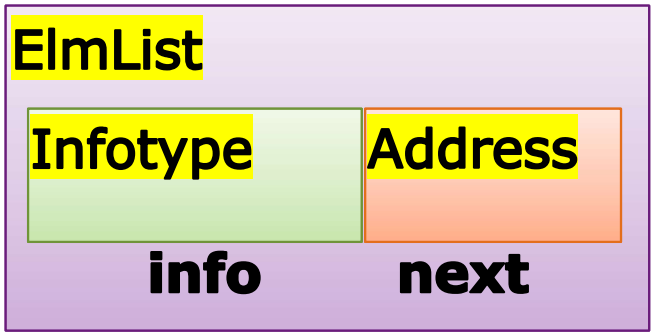
- Generally, each element is divided into 2 parts

Element List



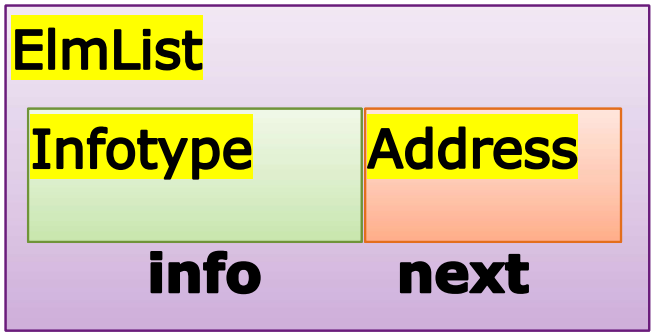
Element List

```
type ElmList <


>
```



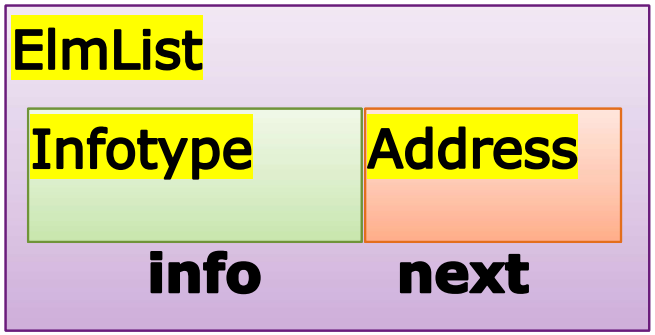
Element List

```
type ElmList <
    info  :
    next :
>
```



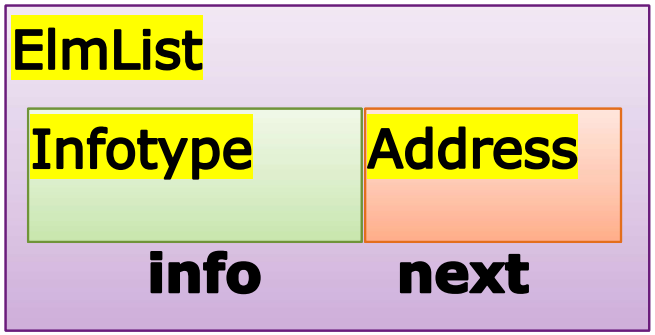
Element List

```
type ElmList <
    info  : integer,
    next :
>
```



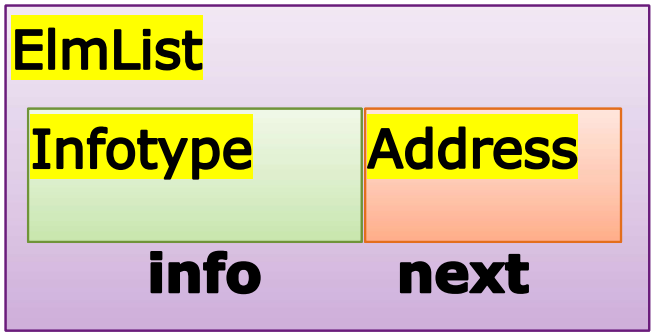
Element List

```
type ElmList <
    info  : integer,
    next : pointer to ElmList
>
```



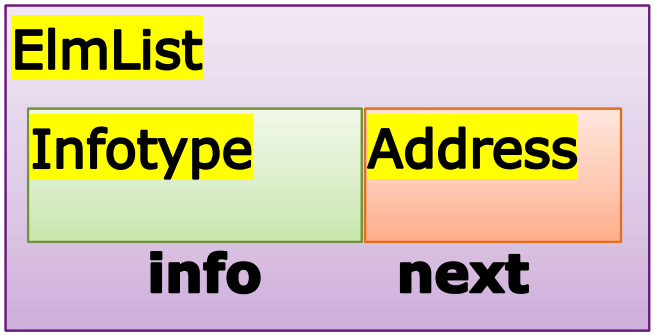
Element List

```
type ElmList <
    info  : integer,
    next : pointer to ElmList
>
```



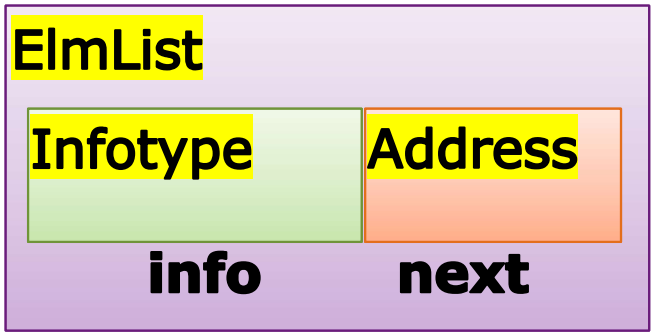
Element List

```
type ElmList <
    info  : string,
    next : pointer to ElmList
>
```



Element List

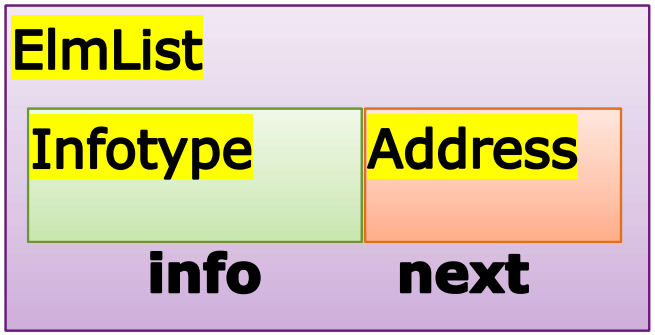
```
type ElmList <
    info  : Infotype,
    next : pointer to ElmList
>
```



type Infotype : integer

Element List

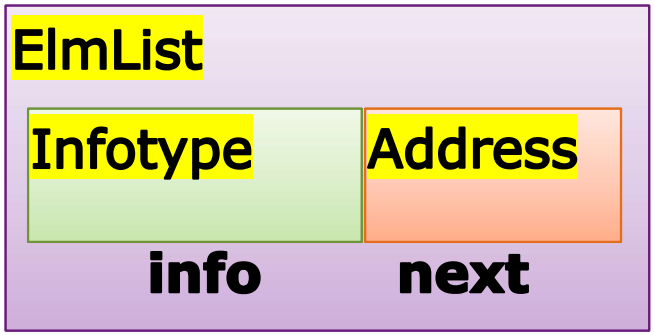
```
type ElmList <
    info  : Infotype,
    next : pointer to ElmList
>
```



type Infotype : integer

type Infotype : string

Element List



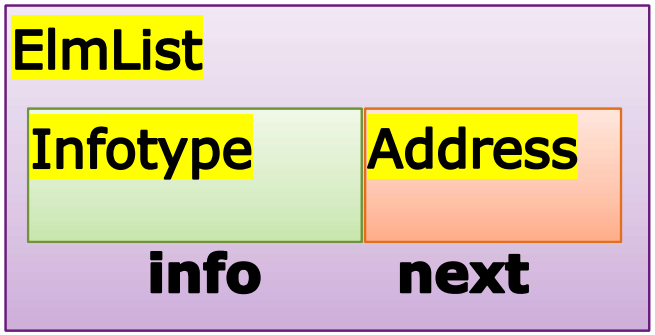

```
type ElmList <
    info  : Infotype,
    next : pointer to ElmList
>
```

type Address : pointer to ElmList

Element List

```
type ElmList <
    info  : Infotype,
    next : Address
>
```

type Address : pointer to ElmList



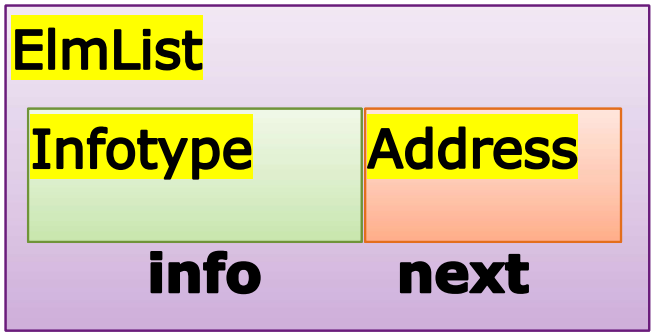
Infotype

- The data that we want to store
- Define your own Infotype
  - Basic type example

```
type Infotype : integer,
type Infotype : char
```

  - Record type example

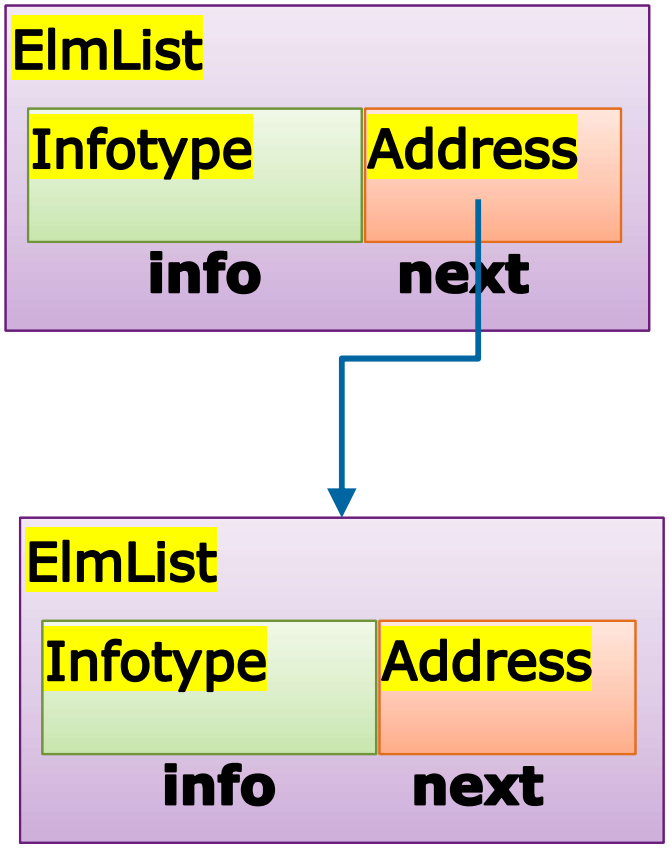
```
type Infotype :
Mahasiswa <
    nim : string,
    name : string
>
```



Address

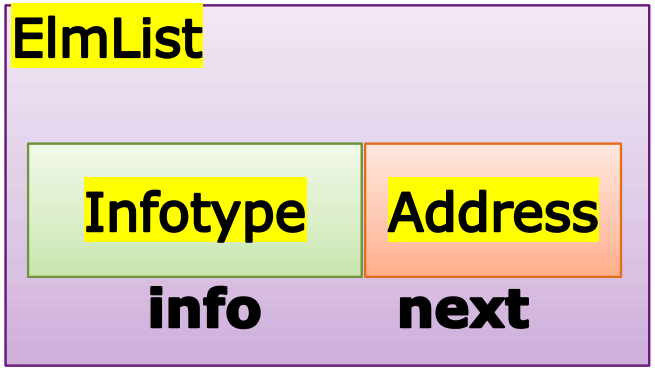
- Pointer to element

  type Address : pointer to ElmList

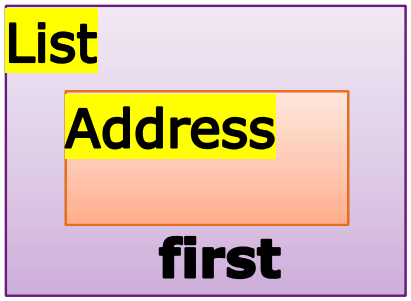


ADT Element List

```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
    info  : Infotype
    next : Address
>
```

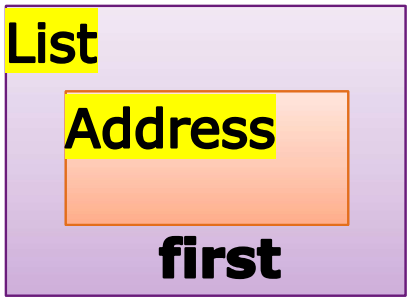


Linked List



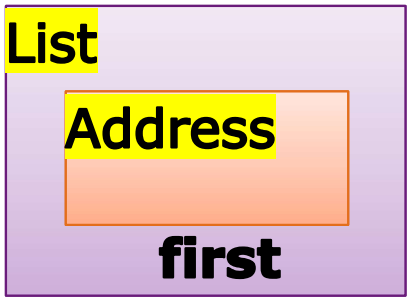
Linked List

type List :



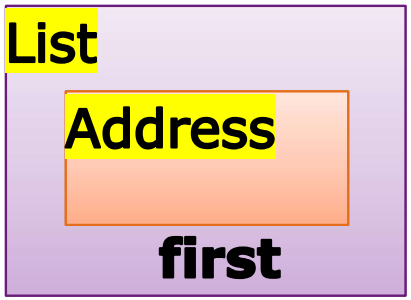
Linked List

```
type List : < first : Address >
```



Linked List

type List : < first : Address >

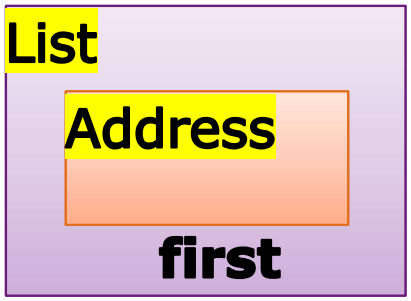


- Create the list variable, L.

Linked List

type List : < first : Address >

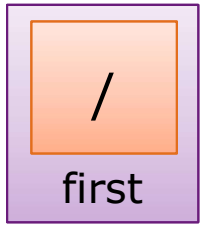
Dictionary
L : List



- Create the list variable, L.

Create New List

```
type List : < first : Address >

Dictionary
    L : List

Algorithm
```

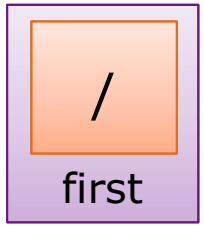


L

- On the creation of new list, there is no element, thus set first of L to NIL

Create New List

```
type List : < first : Address >

Dictionary
    L : List

Algorithm
    L.first = NIL
```

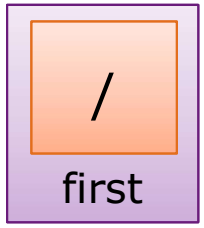


L

- On the creation of new list, there is no element, thus set first of L to NIL

Create New List

```
type List : < first : Address >

Dictionary
    L : List

Algorithm
    L.first = NIL
```



L

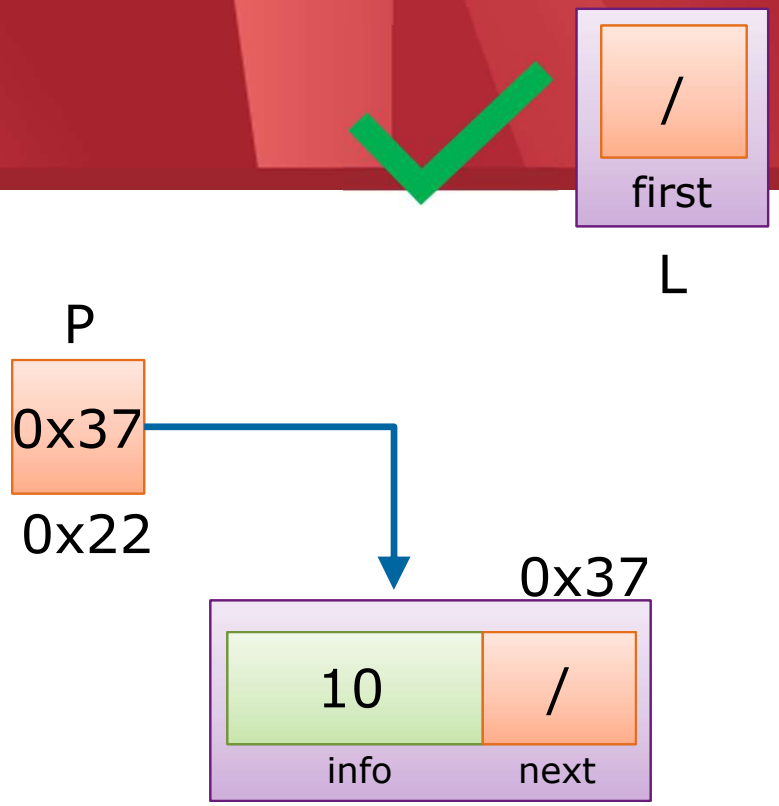
Penulisan pseudocode untuk assignment ← diubah menjadi = untuk menghindari kebingungan karena kita akan pakai banyak tanda panah mulai sekarang.

- On the creation of new list, there is no element, thus set first of L to NIL

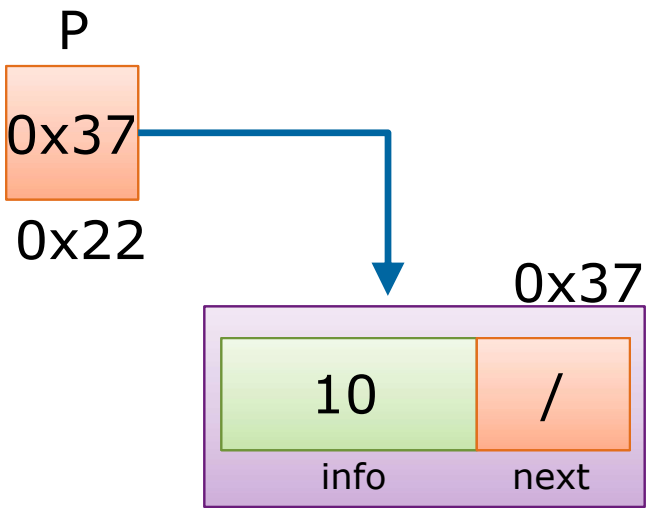
Create New Element

L

Create New Element

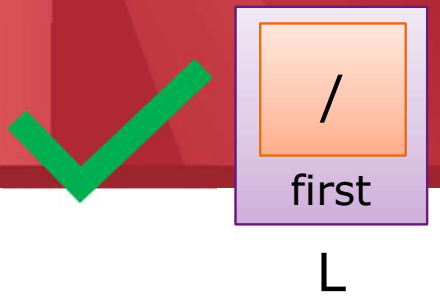


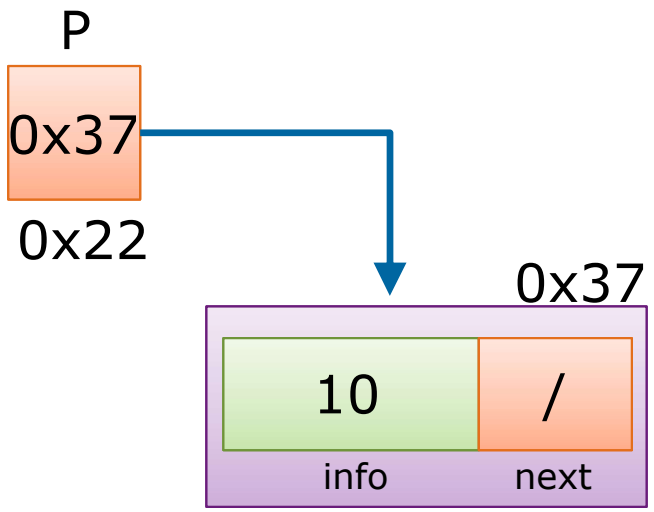
Create New Element




```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```

Create New Element





Dictionary

P :

Algorithm

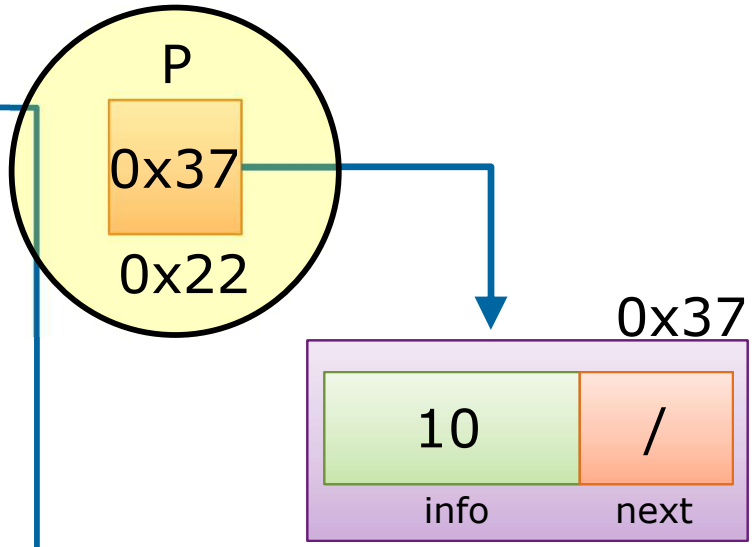
```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```

Create New Element

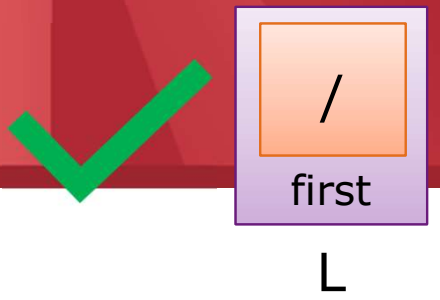
Dictionary

P : Address

Algorithm



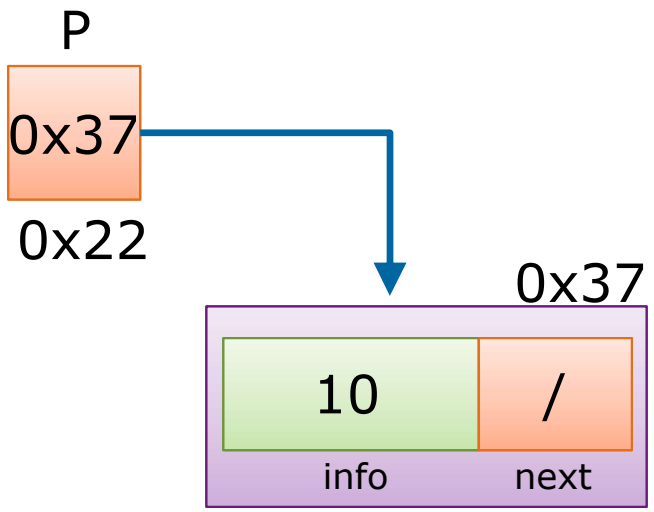

```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```



Create New Element

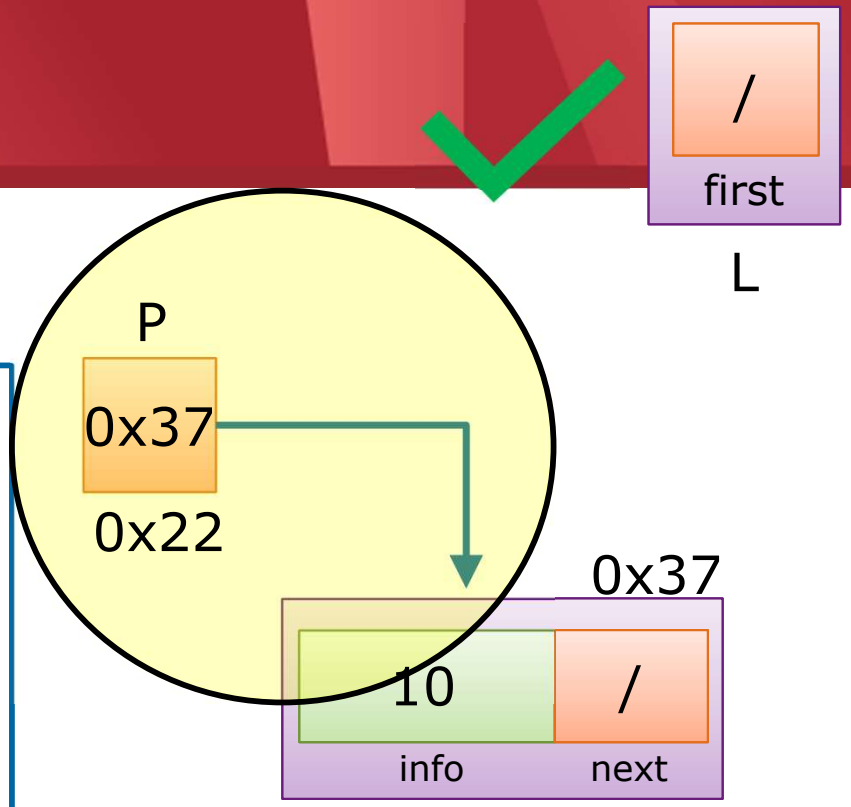
Dictionary
P : Address

Algorithm
allocate(P)




```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```

Create New Element



Dictionary

P : Address

Algorithm

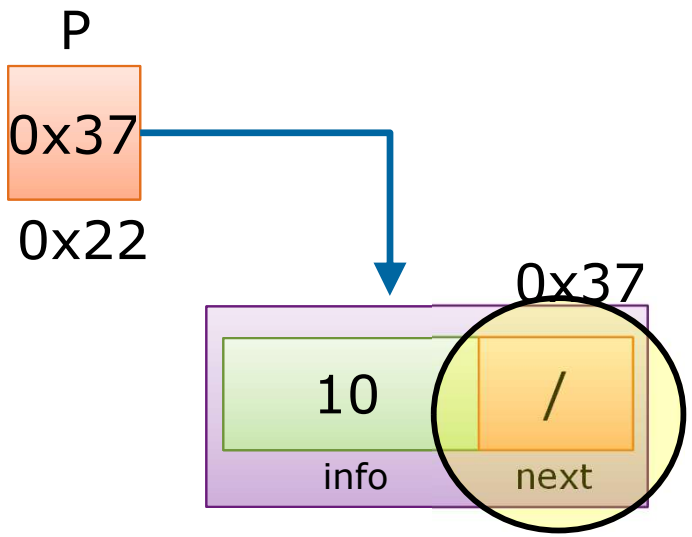
allocate(P)

```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```

Create New Element

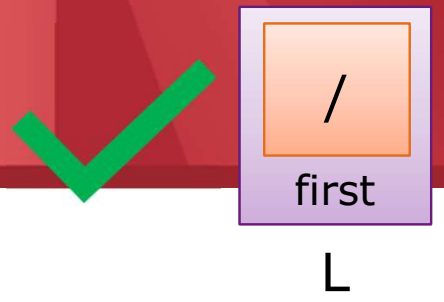
Dictionary
P : Address

Algorithm
allocate(P)
P → next = NIL



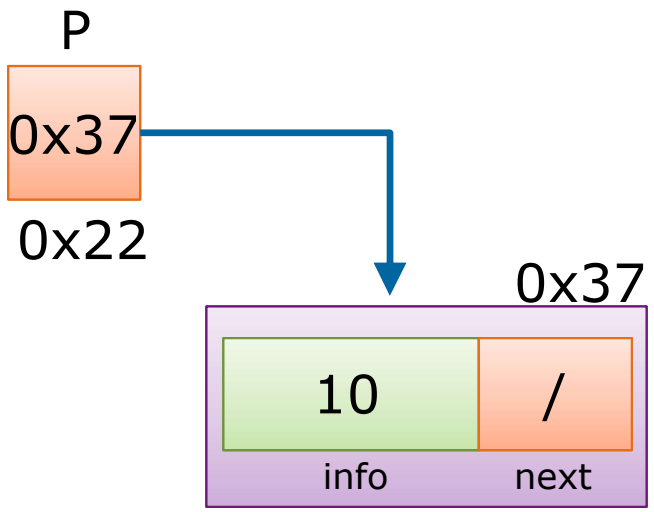

```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```

Create New Element



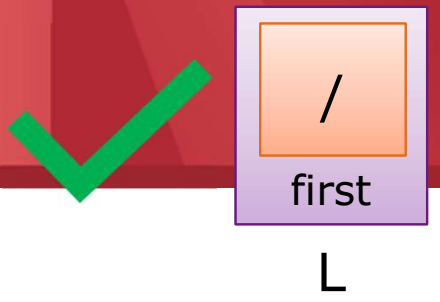
Dictionary
P : Address

Algorithm
allocate(P)
P → next = NIL
P → info = 10



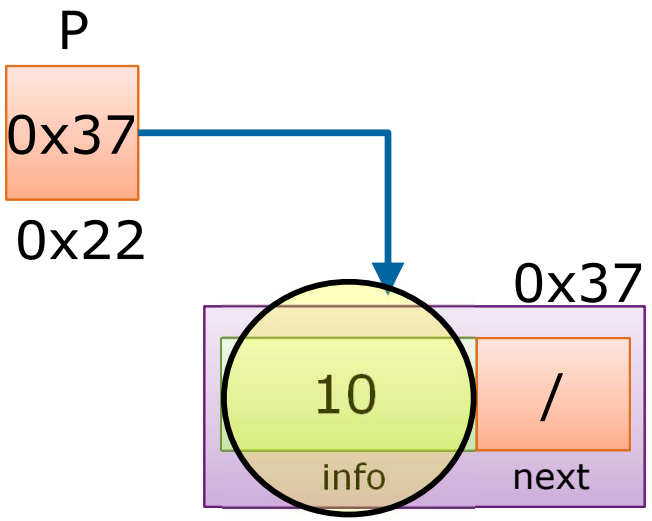

```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```

Create New Element



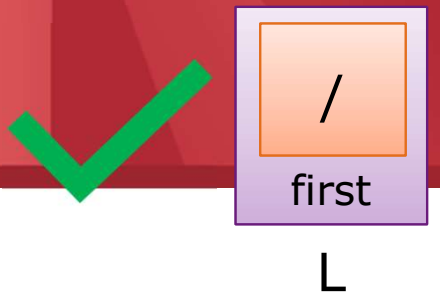
Dictionary
P : Address

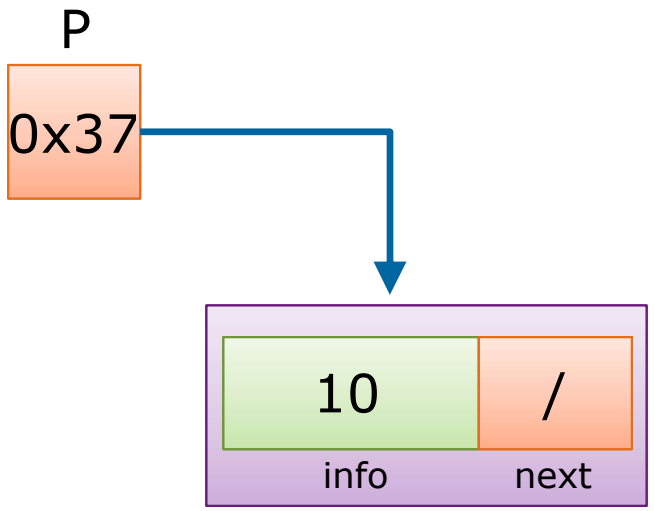
Algorithm
allocate(P)
P → next = NIL
P → info = 10




```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```

Create New Element

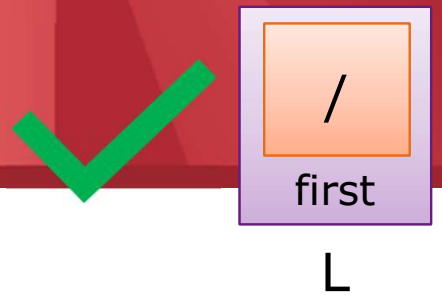


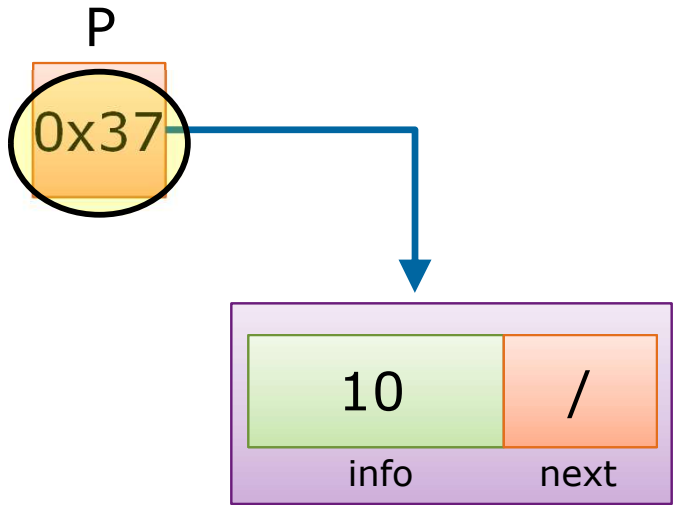



```
Dictionary
    P : Address

Algorithm
    allocate( P )
    P → next = NIL
    P → info = 10
    return P
```

```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```

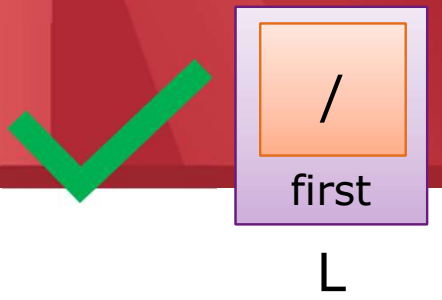
Create New Element



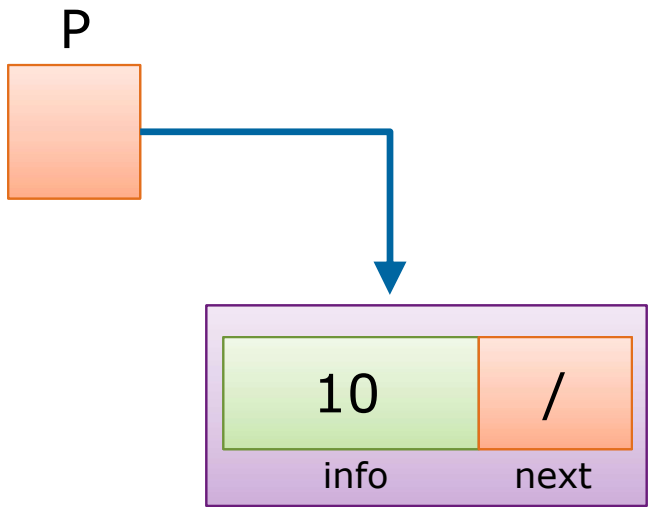



```
Dictionary
    P : Address

Algorithm
    allocate( P )
    P → next = NIL
    P → info = 10
    return P
```

```
type Infotype : integer
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```



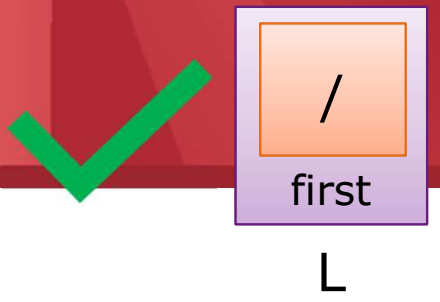
Create New Element

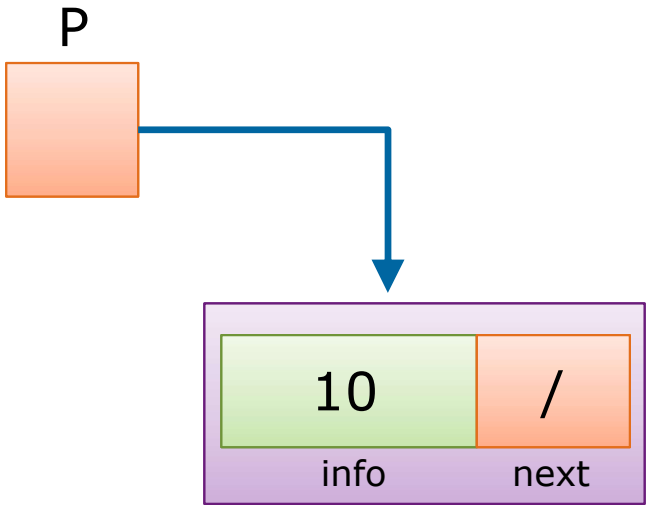



```
Dictionary
    P : Address

Algorithm
    allocate( P )
    P → next = NIL
    P → info = 10
    return P
```

```
type Infotype : Mahasiswa
type Address : pointer to ElmList

type ElmList <
    info  : Infotype
    next : Address
>
```

Create New Element



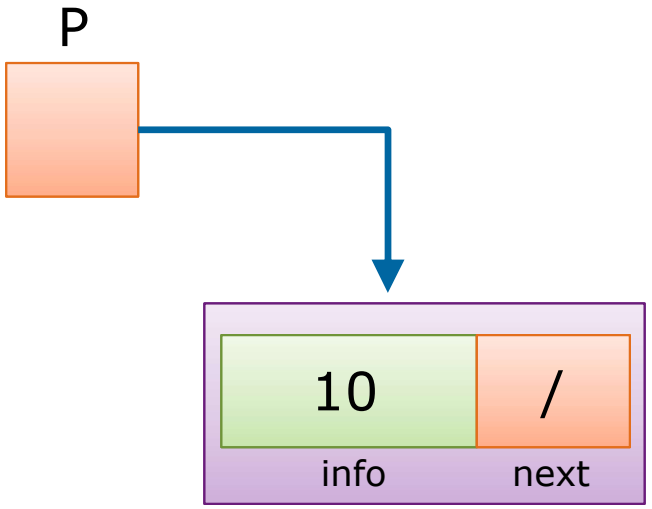



```
Dictionary
    P : Address

Algorithm
    allocate( P )
    P → next = NIL
    P → info = 10
    return P
```

```
type Infotype : Mahasiswa
type Address : pointer to ElmList

type ElmList <
    info  : Infotype
    next : Address
>
```

```
type Mahasiswa <
    nim    : string
    name : string
>
```

Create New Element



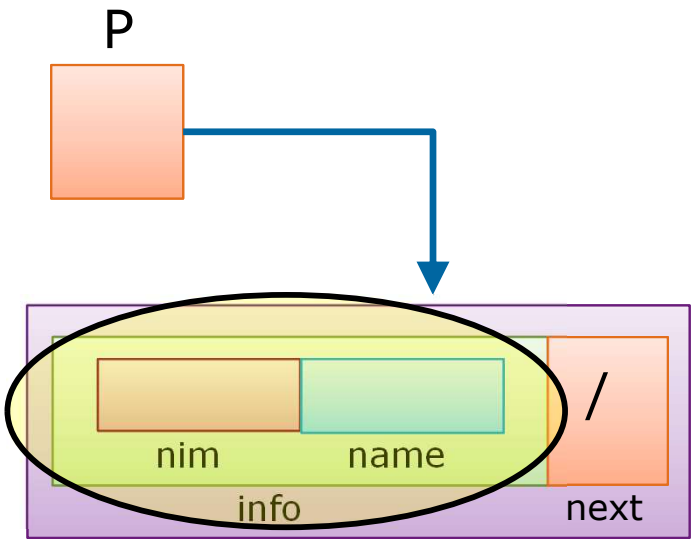

```
Dictionary
    P : Address

Algorithm
    allocate( P )
    P → next = NIL
    P → info = 10
    return P
```

```
type Infotype : Mahasiswa
type Address : pointer to ElmList

type ElmList <
    info  : Infotype
    next : Address
>
```

```
type Mahasiswa <
    nim    : string
    name : string
>
```

L

Create New Element



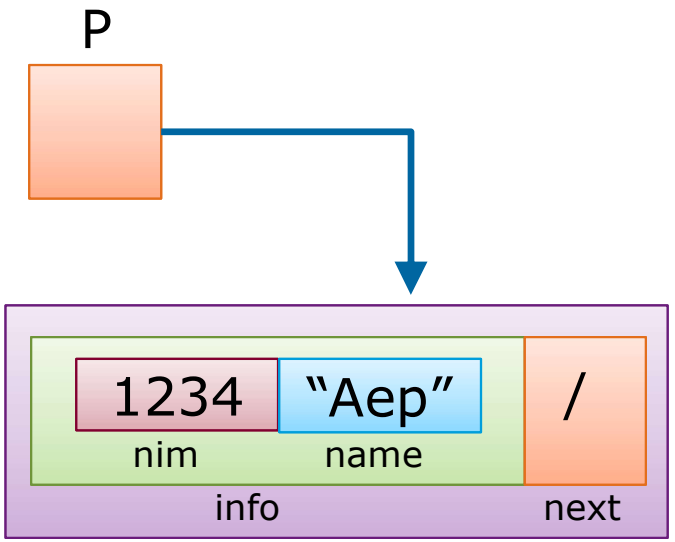

```
Dictionary
    P : Address

Algorithm
    allocate( P )
    P → next = NIL
    P → info = 10
    return P
```

```
type Infotype : Mahasiswa
type Address : pointer to ElmList

type ElmList <
    info  : Infotype
    next : Address
>
```

```
type Mahasiswa <
    nim    : string
    name : string
>
```

Create New Element


```
Dictionary
    P : Address

Algorithm
    allocate( P )
    P → next = NIL
    ( P → info ).nim = 1234
    ( P → info ).nama = "Aep"
    return P
```



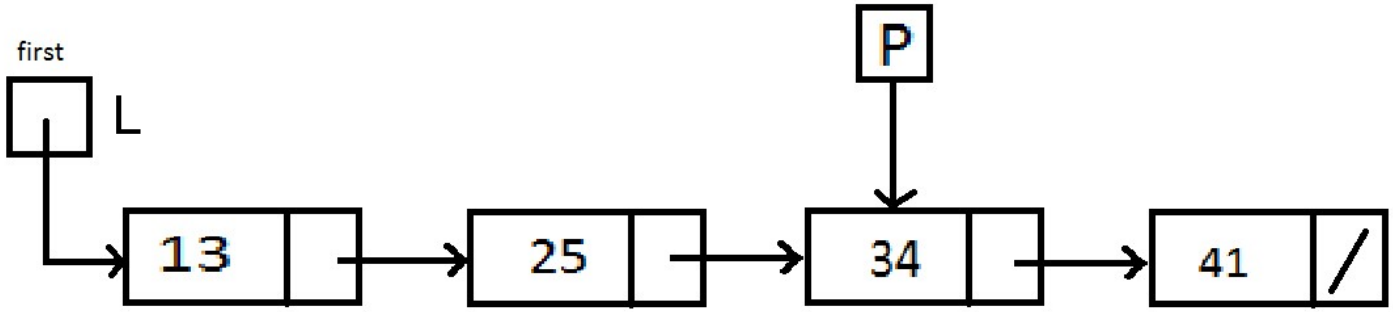

```
type Infotype : Mahasiswa
type Address : pointer to ElmList

type ElmList <
      info  : Infotype
      next : Address
>
```

```
type Mahasiswa <
      nim    : string
      name : string
>
```

Question?

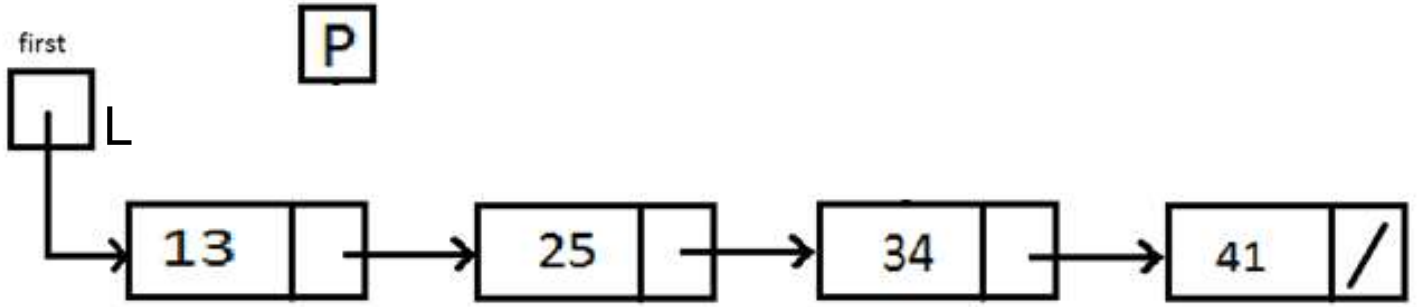

TRAIN YOUR BRAIN

Exercise



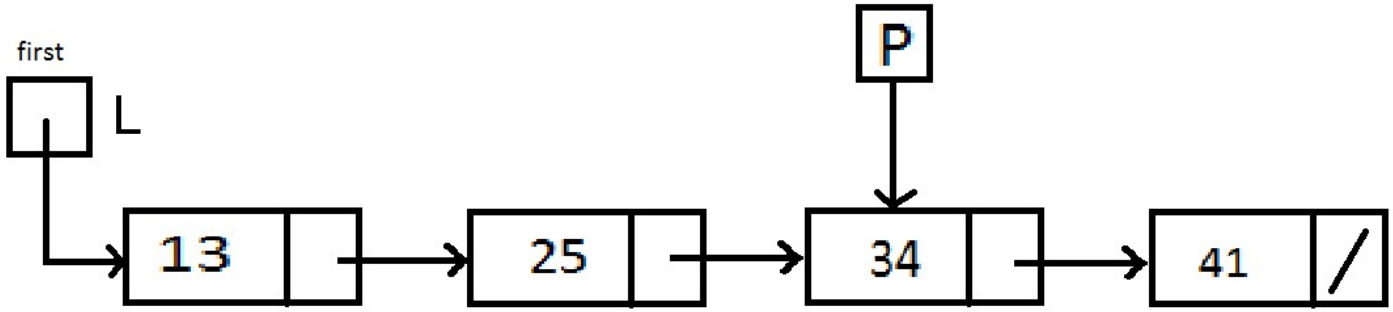
Task	Answer
output(P → info)	
output((L.first) → info)	
output((P → next) → info)	
P = (L.first) → next output(P → next) → info	

Exercise



Task	Answer
Make P points the first element	
Make P points the second element	
Make P points the last element	
Output info the first element of the list	
Output info of the last element	

Exercise



Task	Answer
Copy info element P into first element	
Copy info the second element into P	
Set info of first element = 10	

Diketahui struktur sebagai berikut:

```
type IpkElem <
   ipk  : float,
   next : pointer to IpkElem >
```

Buat fungsi **createNewIpkElem** yang mengembalikan objek IpkElem yang sudah diisi dengan nilai yang dikirim sebagai parameter:

```
function createNewIpkElem( newipk : float ) -> IpkElem
{ mengembalikan pointer yang menunjuk ke objek IpkElem yang telah
  terisi newipk untuk ipk dan NIL untuk next. }
dictionary



algorithm



endfunction
```

Diketahui struktur sebagai berikut:

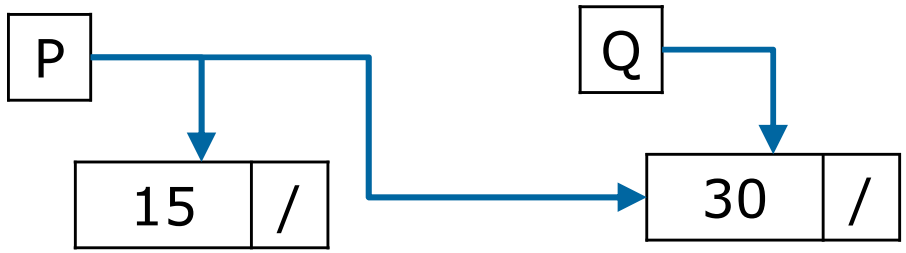
```
type BookElem <
   title  : string,
   next : pointer to BookElem >
```

Buat fungsi **createNewBookElem** yang mengembalikan objek BookElem yang sudah diisi dengan nilai yang dikirim sebagai parameter:

```
function createNewBookElem( newtitile : string ) -> BookElem
{ mengembalikan pointer yang menunjuk ke objek BookElem yang telah
  terisi newtitle untuk title dan NIL untuk next. }
dictionary



algorithm



endfunction
```

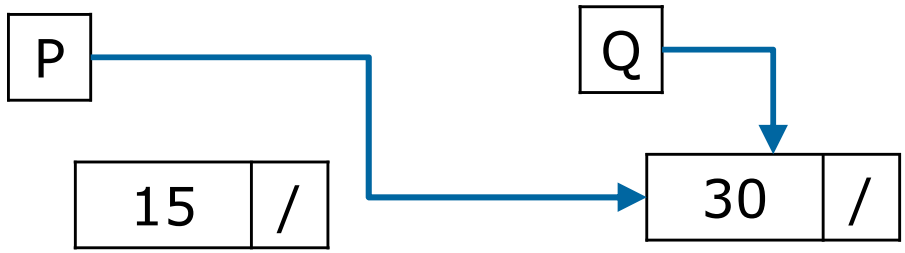
Careful with Pointer

- Suppose we have allocated two elements using pointer P and Q
- Then we change the pointer P so that P points to the element pointed by Q
  - P ← Q
- Then what will happen to element (15) ?



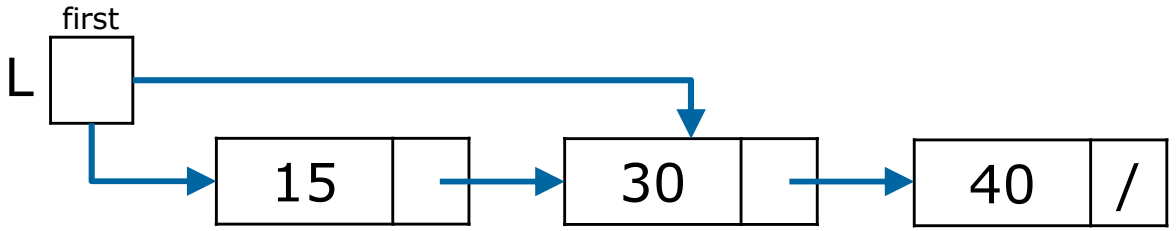
Careful with Pointer

- The element (15) is **lost**
  - Not deleted, but lost, and cannot be accessed
  - Memory still used (wasted memory / leaked memory)



Careful with Pointer

- Suppose we have



- What will happen if we run the operation

  L.first = (L.first) → next

- Element 15 cannot be accessed
  - Again, lost, and cannot be accessed
  - Memory still used (wasted memory)

Question?

Fakultas Informatika
School of Computing
Telkom University

CDK2AAB4
STRUKTUR DATA

Singly Linked List

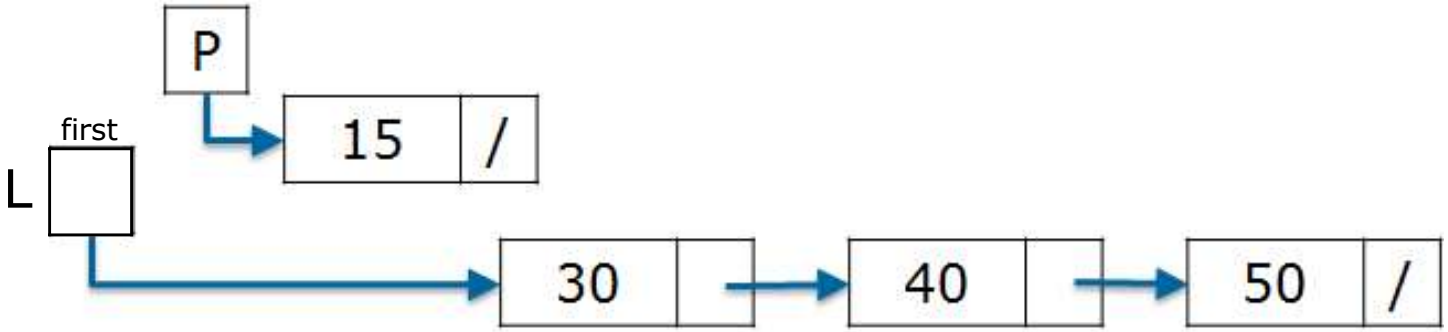
Insertion and Deletion

Inserting new Element

- **Insert first**
  - New element became the first element of the list
- **Insert last**
  - New element became the last element of the list
- **Insert after / Insert before**
  - Put the element somewhere in the middle

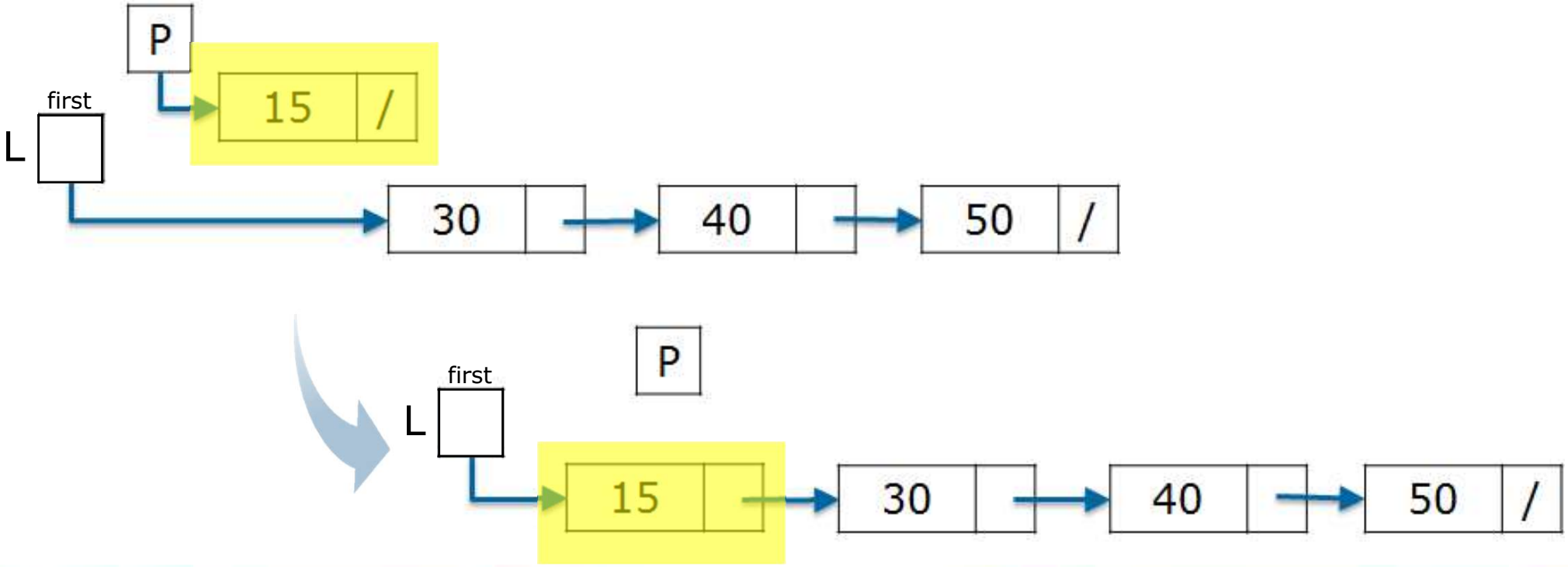
Insert First

- Insert element P into List L so that P become the first element of L



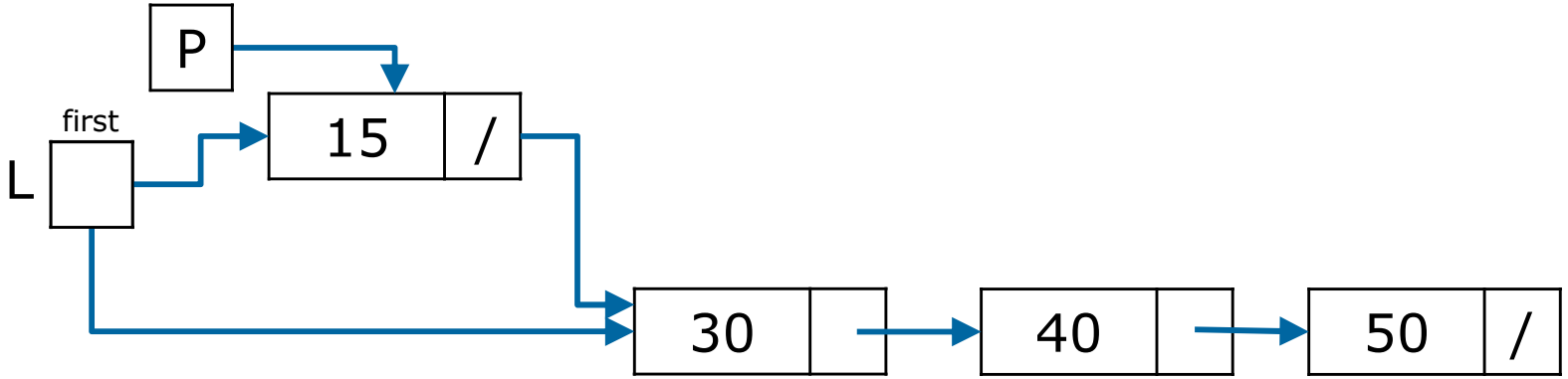
Insert First

- Insert element P into List L so that P become the first element of L



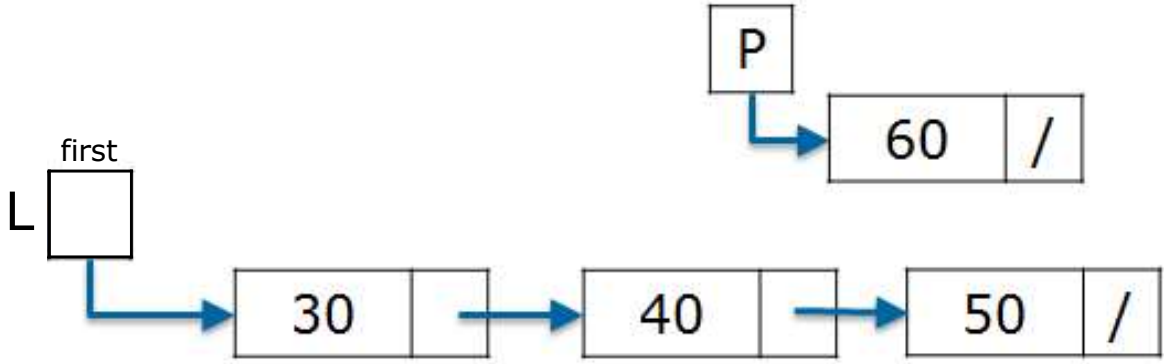
Insert First

Algorithm



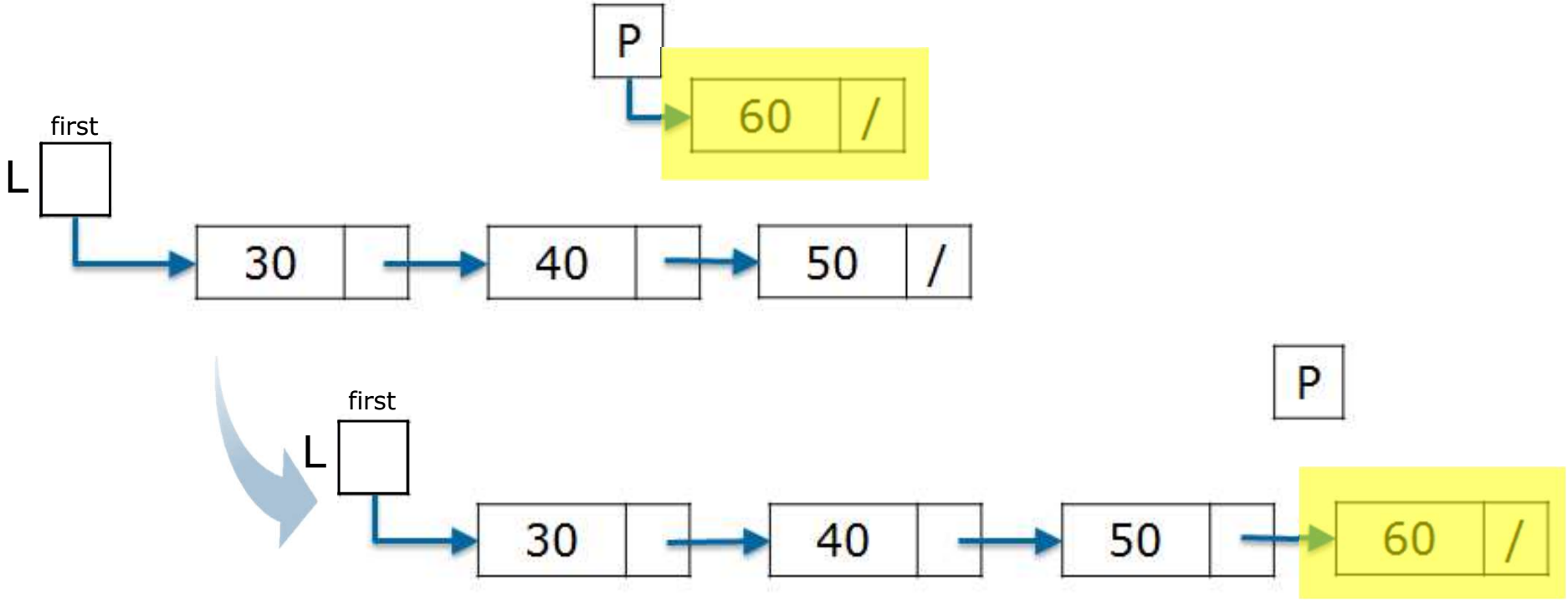
Insert Last

- Insert element P into List L so that P become the last element of L



Insert Last

- Insert element P into List L so that P become the last element of L



Insert Last

Dictionary

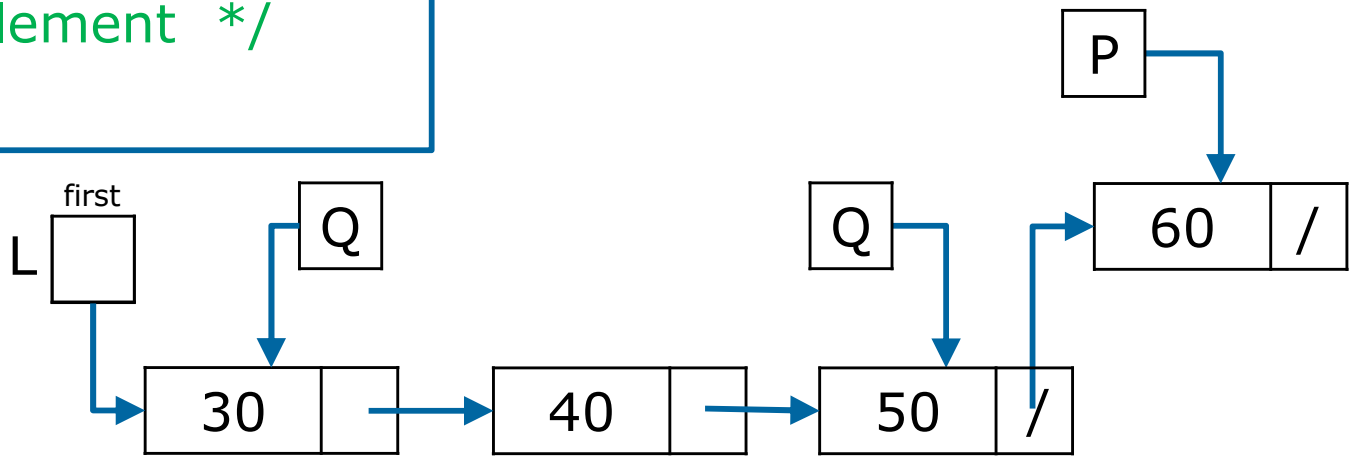
Q : Address

Algorithm

Q ← L.first

/* create a mechanism so that Q points the last element */

Q → next = P



Insert Last

Dictionary

Q : Address

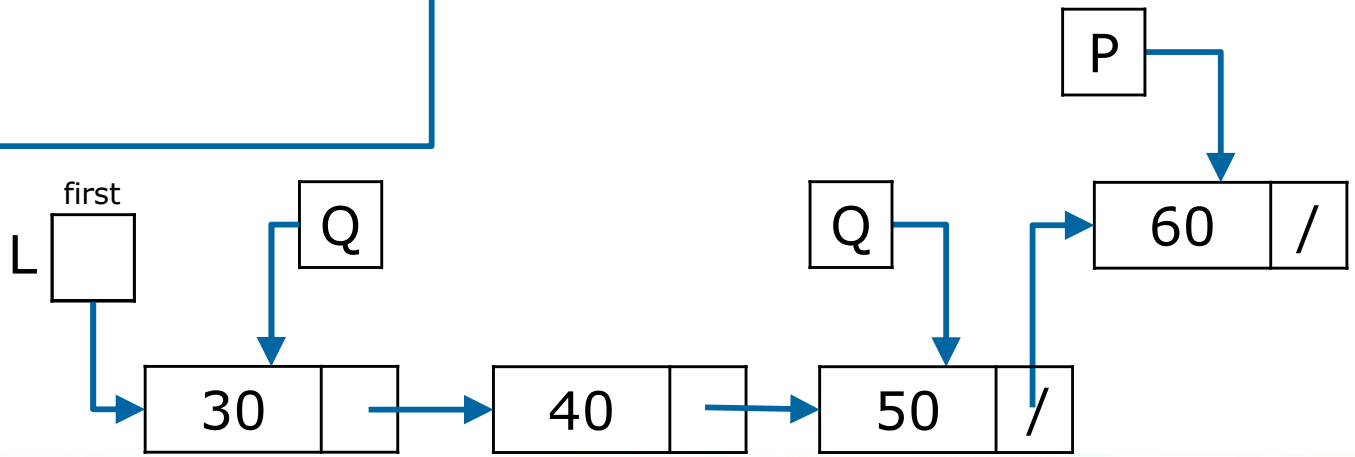
Algorithm

Q ← L.first

while

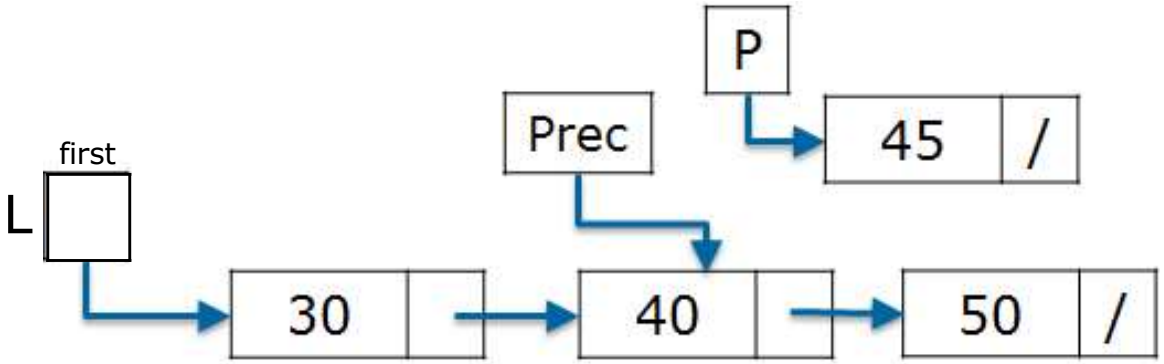
endwhile

Q → next = P



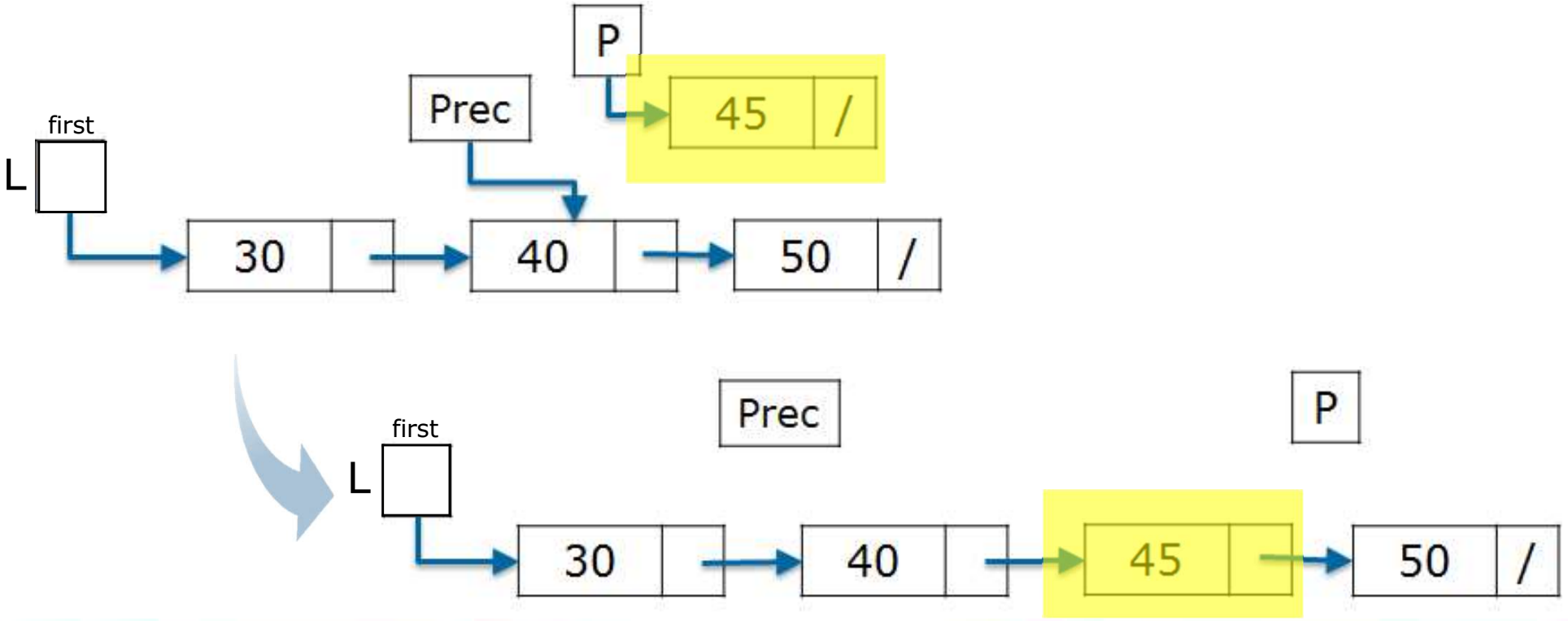
Insert After

- Insert element P into List L so that P become the next element of Prec



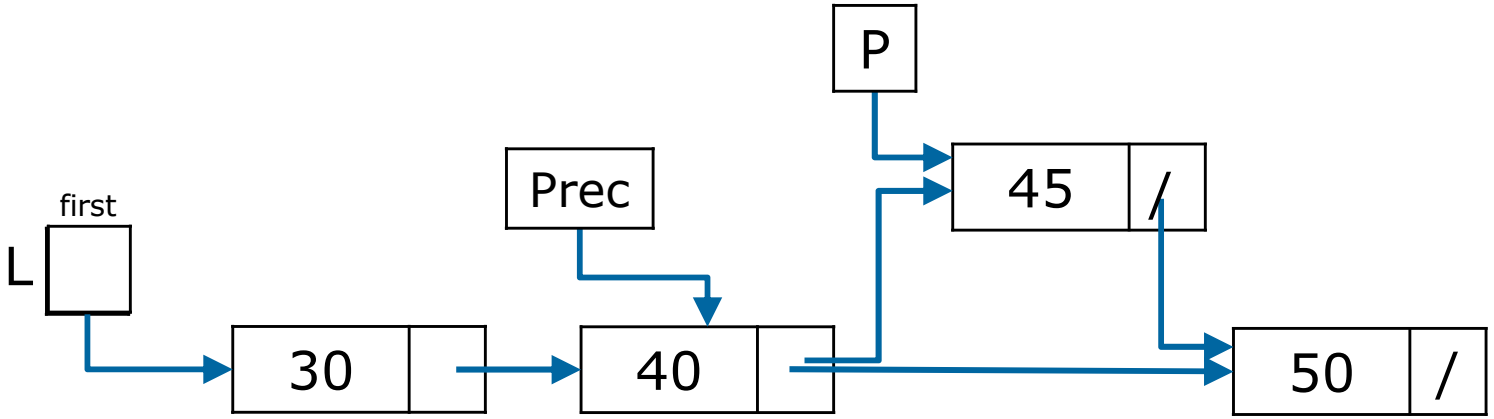
Insert After

- Insert element P into List L so that P become the next element of Prec



Insert After

Algorithm

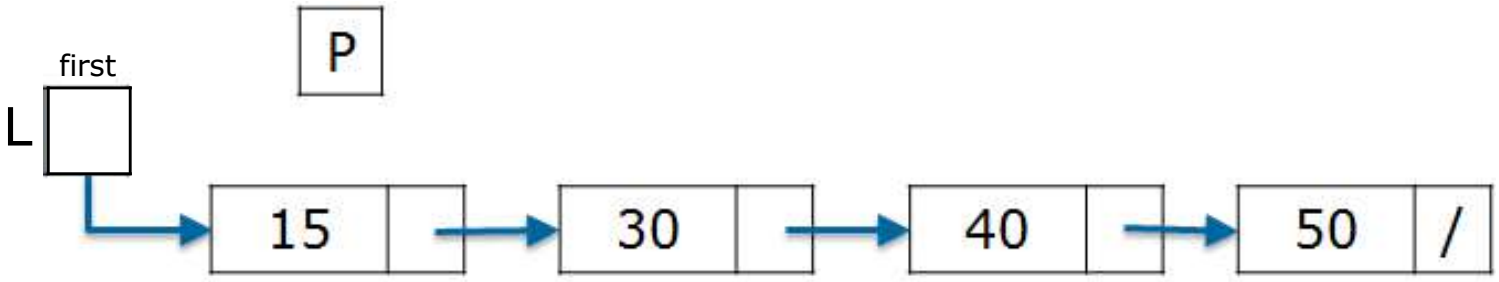


Deleting the Element

- **Delete first**
  - Remove the first element of the list
- **Delete last**
  - Remove the last element of the list
- **Delete after**
  - Remove an element next to a particular element

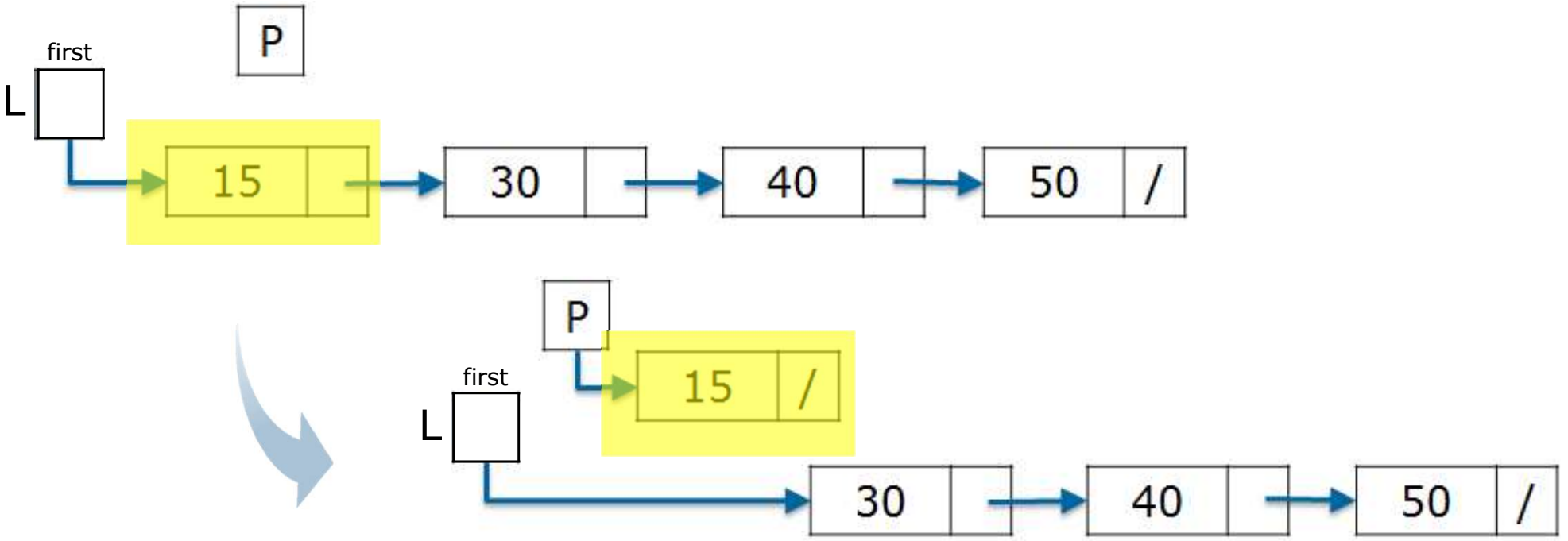
Delete First

- Remove the first element of L



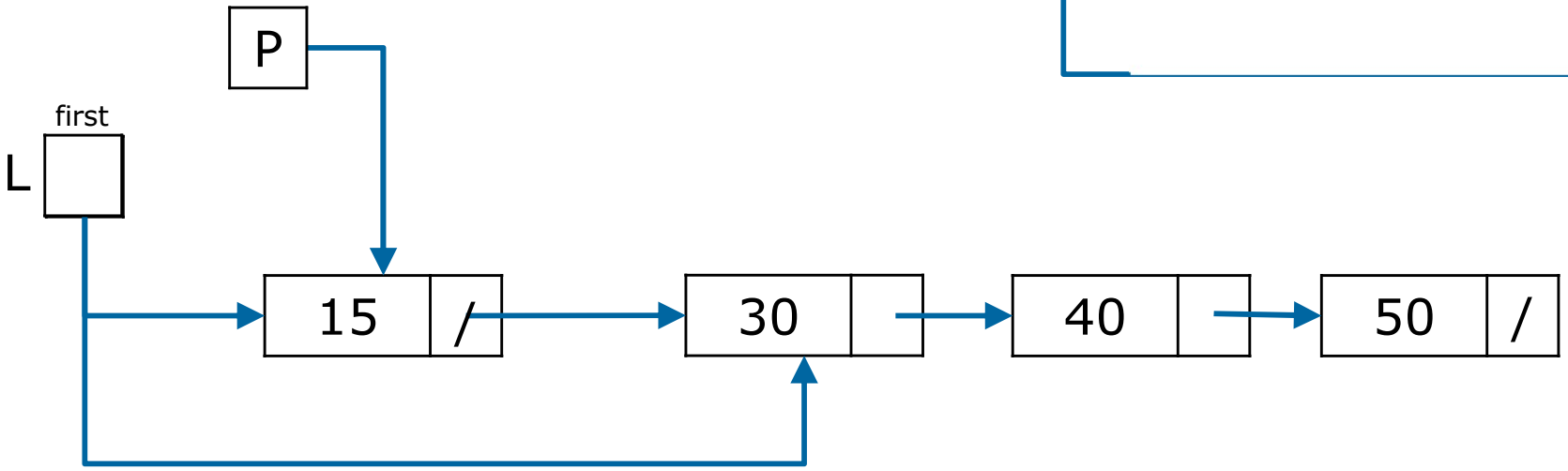
Delete First

- Remove the first element of L



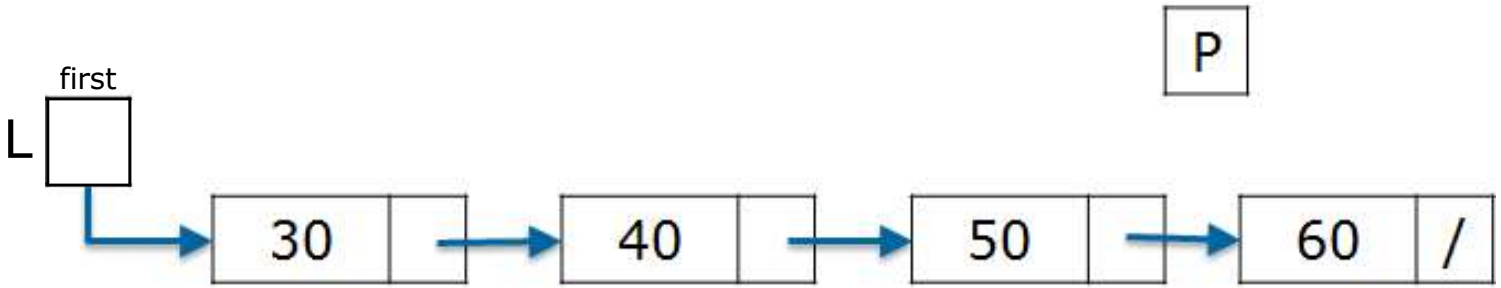
Delete First

Algorithm



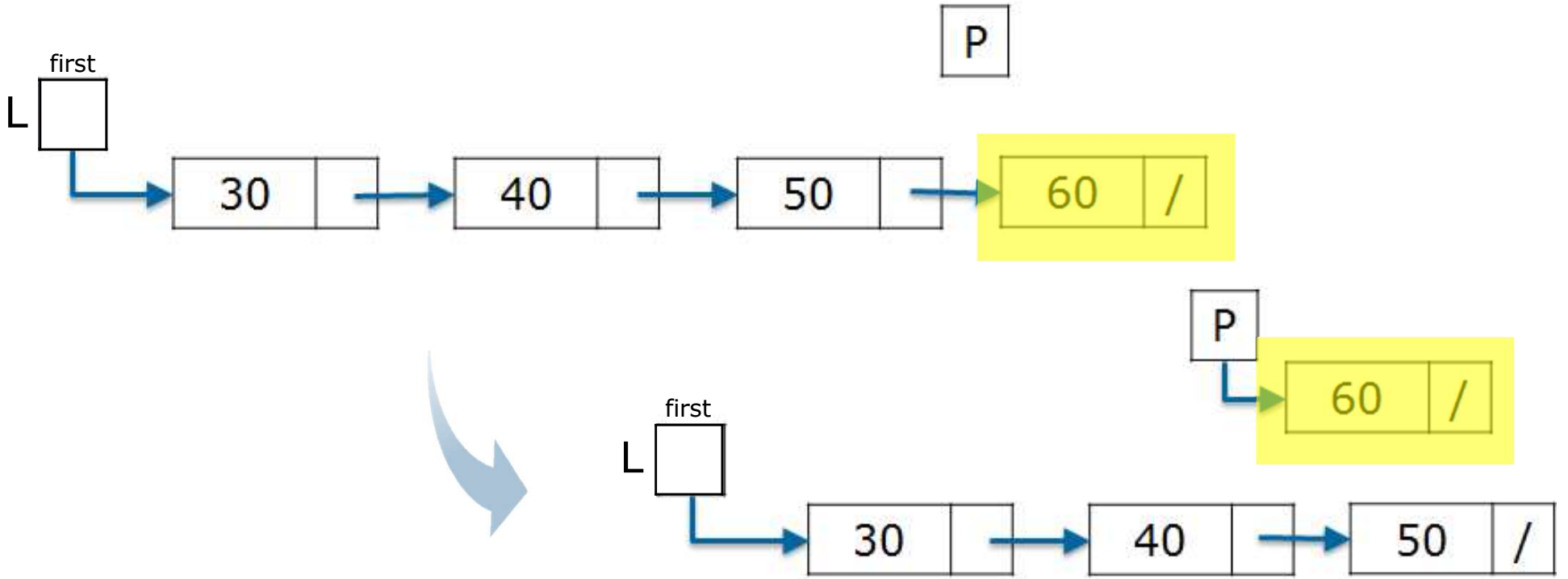
Delete Last

- Remove the last element of L



Delete Last

- Remove the last element of L



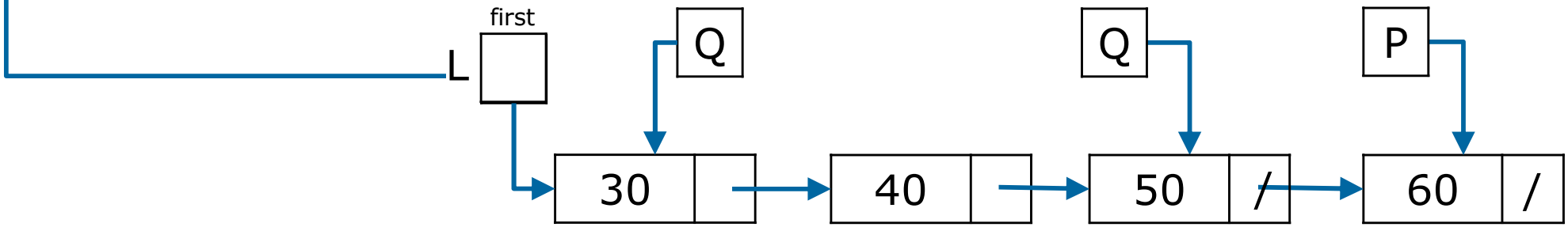
Delete Last

Dictionary

Q : Address

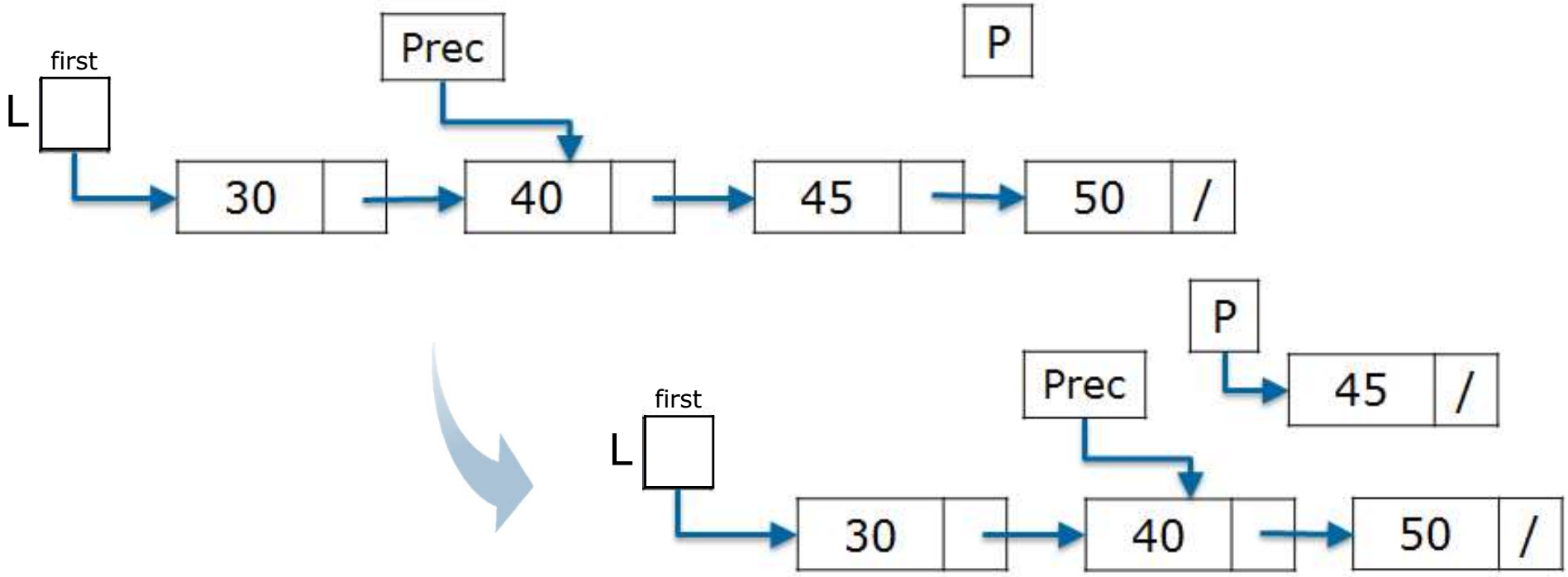
Algorithm

```
Q = L.first
/* create a mechanism so that Q points
   the element before the last element */
P = Q → next
Q → next = NIL
```



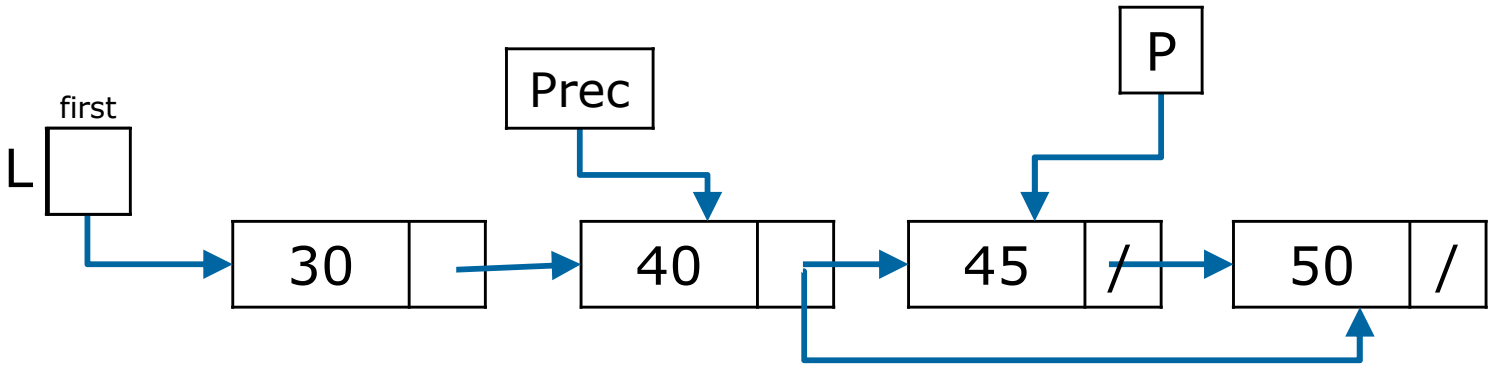
Delete After

- Remove element after the element pointed by Prec



Delete After

Algorithm



Mind the special conditions

- Empty list
- Only 1 element in list

CAREFUL

- NOTE that these delete functions only **REMOVE** the element from the list and not completely delete it
- The “P” element are **still in the memory**
- The “P” element still need to be **DEALOCATED** to be fully deleted

Question?

Fakultas Informatika
School of Computing
Telkom University

CDK2AAB4
STRUKTUR DATA

Searching and FindMax

Singly Linked List

Linear or Sequential Search Algorithm

- It **searches an element** from array or linked list by examining each of the elements and **comparing it with** the search element starting with the **first** element **to the last element** in the list.

Reference: Parmar, V. P. & Kumbharana, C. K. (2015). **Comparing Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented through Static Array, Dynamic Array and Linked List**. International Journal of Computer Applications (0975 – 8887).

Linear Search Algorithm (Array)

Linear Search (Linked List)

Linear Search Algorithm

- Linear search with array or linked list.
  Which one is better?

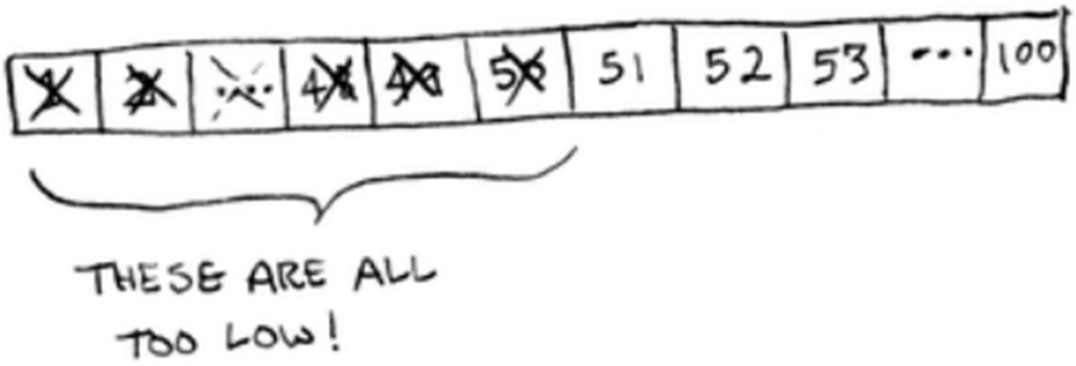
Binary Search Algorithm

- **Binary search** is an efficient algorithm for finding an element in **a sorted array** by repeatedly **dividing** the search interval **in half**, **comparing the middle element to the target value**. If the target is smaller or larger, the search continues on the left or right half respectively, until the target is found, or the search space is empty.

Reference: Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Binary Search Illustration

- Is there 75 inside the array?



Binary Search with Array vs. Linked List?

- Which one is better?

FindMax with Array vs. Linked List?

- Which one is better?

Question?

Exercise

- Discuss complexity comparison between array and Singly Linked List (SLL) of all primitives you've learnt. Which one better for insert and delete (first, last, after)

Fakultas Informatika
School of Computing
Telkom University

CDK2AAB4
STRUKTUR DATA

Primitives Comparison

Array vs. SLL

Discussion

- Let’s discuss complexity comparison between array and Singly Linked List (SLL) of all primitives you’ve learnt. Which one better for search, findMax, insert, and delete

Array vs. SLL (worst case comparison)

Primitives	U-Array	O-Array	U-SLL	O-SLL
Search				
FindMax				
Insert				
Delete				

Referensi

- Karumanchi, N. (2017). **Data Structures And Algorithms Made Easy (5th ed.)**. CareerMonk Pub.
- Weiss, M. A. (2014). **Data Structures and Algorithm Analysis in C++ (4th ed.)**. Addison-Wesley Pub.
- Drozdek, A. (2013). **Data Structures and Algorithms in C++ (4th ed.)**. Cengage Learning.
- Gilberg, R. F., Forouzan, B. A. (2005). **Data Structures: A Pseudocode Approach with C (2nd ed.)**. Thomson Learning, Inc.

Fakultas Informatika
School of Computing
Telkom University

THANK YOU