CANTHO UNIVERSITY

Chapter 2: Common abstract data types

Lâm Hoài Bảo - FSE – CICT
Trương Minh Thái – FSE - CICT

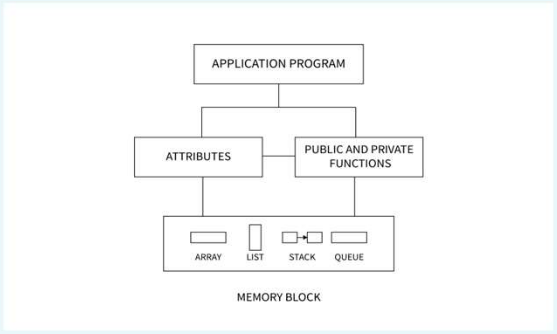
www.ctu.edu.vn

1

CANTHO UNIVERSITY

Content

- List abstract data type (LIST)
- Stack abstract data type (STACK)
- Queue abstract data type (QUEUE)



2

www.ctu.edu.vn

2

LIST

- List concept
- List operations
- List settings
  - Array-based list (ArrayList)
  - Using the cursor (Linked List)

Content

- Pointer-based List
- Operators
- Other linked lists
- Summary

CANTHO UNIVERSITY

List

- A collection of elements of a given type (ElementType)
  - $[A_0, A_1, \ldots, A_{n-1}]$
  - A_0 is at position 0, A_1 is at position 1, …
- Operators of a sequence DS
  - Add a new element
  - Remove an element
  - Access an element
  - …

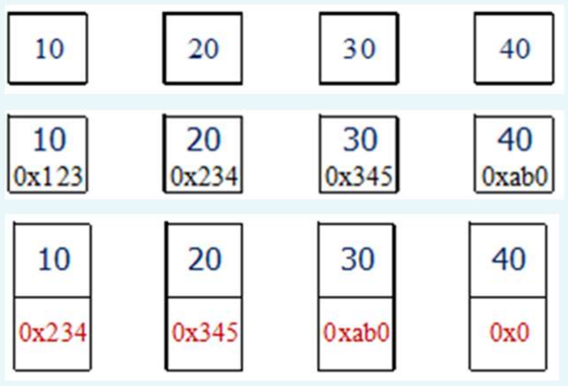
www.ctu.edu.vn 5

5

CANTHO UNIVERSITY

Linked List

- A sequence DS of which elements are discontiguous; instead, they use pointers to link together.
  - Example: L = [10, 20, 30, 40]



(1): values in memory

(2): address of memory to store the value

(3): link to the next memory

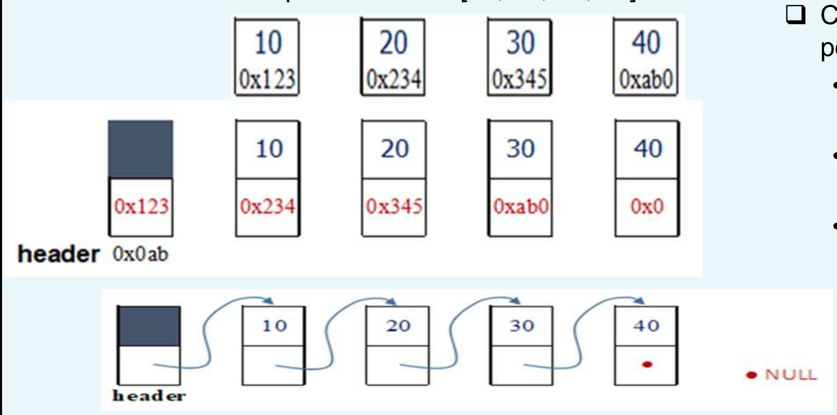
6

www.ctu.edu.vn

6

Linked List

- A sequence DS of which elements are discontiguous; instead, they use pointers to link together.
  - Example: L = [10, 20, 30, 40]



❑ Connect consecutive elements using pointers
- The header element points to the first element a1
- The element ai points to the element ai+1
- The aspect points to the special element called NULL

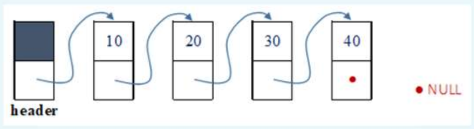
www.ctu.edu.vn

7

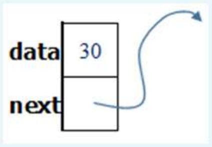
7

Linked List

- A sequence DS of which elements are discontiguous; instead, they use pointers to link together.
  - Example: L = [10, 20, 30, 40]



- header: a pointer refers to the first node, **header ≡ L**
- Each node:
  - data field
  - An address (next) to keep a reference to the next node

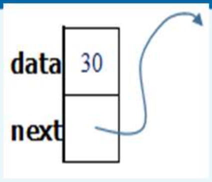


www.ctu.edu.vn

8

8

Declaration



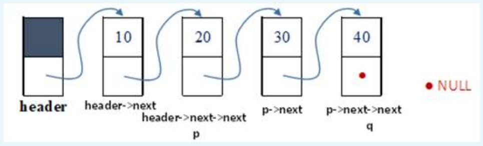
```
typedef … ElementType;
typedef struct NodeTag{
    ElementType data;
    //Pointer to the next node
    struct NodeTag *next;
}Node;

typedef Node *List;
```

9

Example [1]

•List of integers: [10, 20, 30, 40]



header->next->data →10
header->next->next->data → 20

q->next → 20

```
typedef int ElementType;
typedef struct NodeTag{
    ElementType data;
    //Pointer to the next node
    struct NodeTag *next;
}Node;
typedef Node *List;
List header;
```

10

Example [2]

- A polygon is composed of vertices, each vertex is a pair of coordinates (x, y)

```
typedef struct{
    int x, y;
}Point;

typedef struct NodeTag{
    Point vertex;
    struct NodeTag* Next;
}Node;
typedef Node* List;
```

Agenda

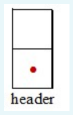
- Pointer-based list
- Operators
- Other linked lists
- Summary

List operators

Operator	Description
makeNull(&L)	Initialize an empty list
len(L)	Number of elements
empty(L)	Check whether the list is empty?
fullList(L)	Check whether the list is full?
print(L)	Traverse the list to print out all elements
getAt(p, L)	Return the element at position p
setAt(p, x, &L)	Update the element at position p by a new value x
insertAt(p, x, &L)	Insert x at position p
popAt(p, &L)	Remove and return the element at position p
insertFirst(x, &L)	Insert x to the first position
popFirst(&L)	Remove and return the first element
append(x, &L)	Append a new element to the list
popLast(&L)	Remove and return the last element
locate(x, L)	Return the position of the first appearance of x in the list

List construction

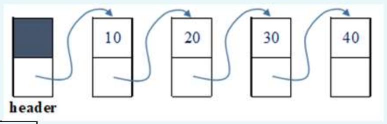
- Create a new node, let header be the node's address
- The next field of the new node is NULL
- The list is header



```
void makeNull(List *pL){
    Node *header = (Node*)malloc(sizeof(Node));
    header->next = NULL;
    (*pL) = header;
}
```

List traversal

- Visits each element of the list
- Algorithm



```
ALGORITHM traverse(L):
    p ← L
    while p->next != NULL:
        Process p->next->data
        p ← p->next
```

- Print all elements of the list

```
void print(List L){
    Node *p = L;
    while (p->next != NULL){
        printf("%d ", p->next->data);
        p = p->next;
    }
}
```

T(n) = O(n)

Length of list

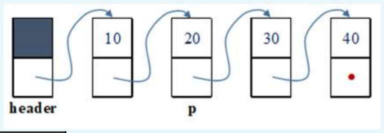
- Replace Process by updating the count variable
- Algorithm

```
ALGORITHM len(L):
    p ← L
    d ← 0
    while p->next != NULL:
        d++
        p ← p->next
    return d
```

T(n) = O(n)

Get the pointer referring to the ith element

- Example: getPosition(2, p) → p
- Start from the first pointer (L), move to the next pointer, update position. Loop until position i or the end of the list

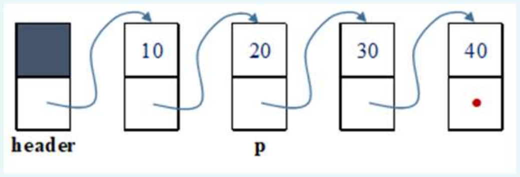


```
ALGORITHM getPosition(i, L):
    p ← L
    j ← 0
    while p->next != NULL and j < i:
        j++
        p ← p->next
return p
```

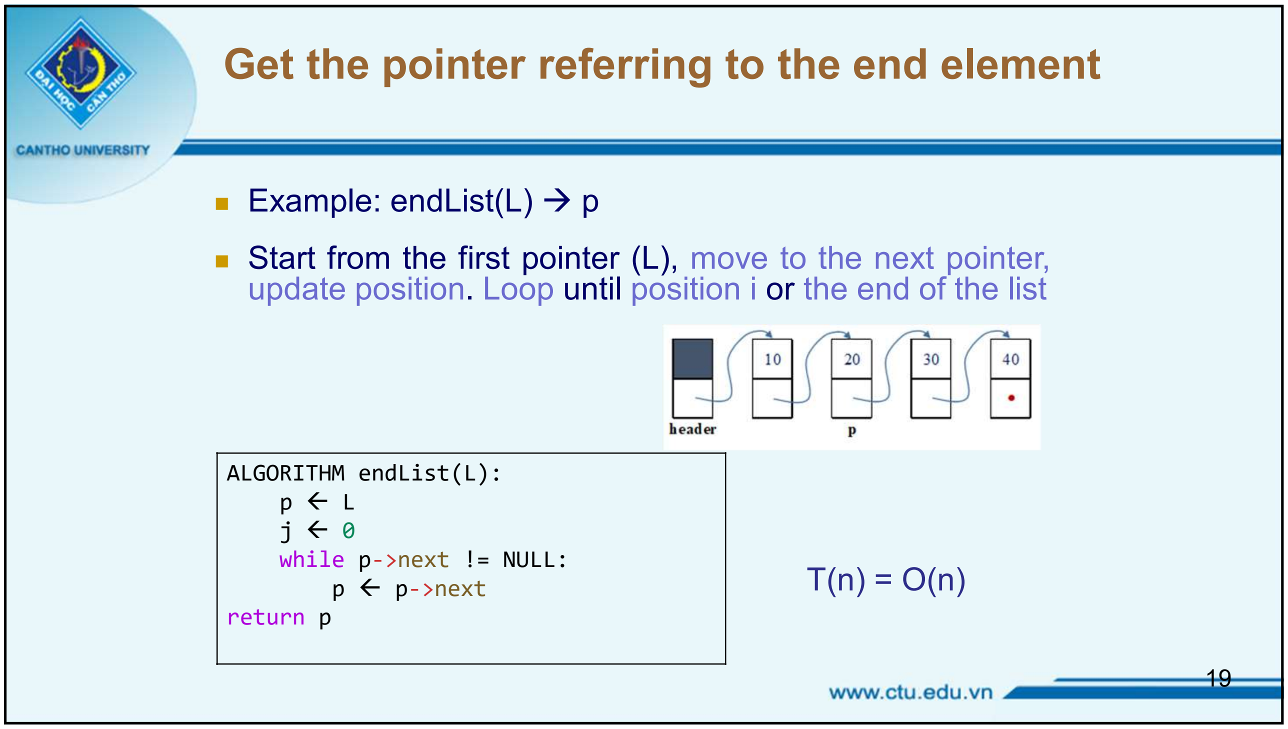
T(n) = O(n)

Get the pointer referring to the first element

- Example: first(L) → header

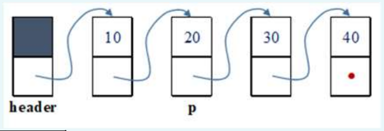


```
ALGORITHM first (L):
    p ← L
    return p
```



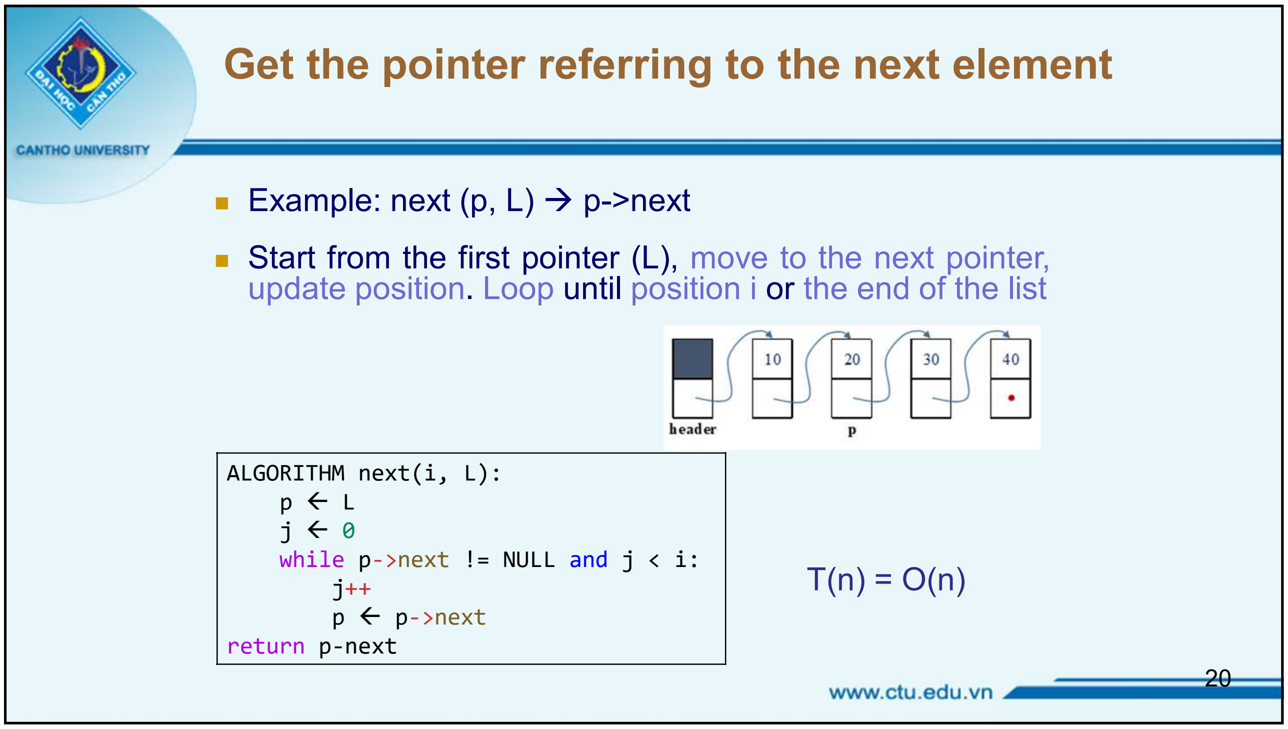
Get the pointer referring to the end element

- Example: endList(L) → p
- Start from the first pointer (L), move to the next pointer, update position. Loop until position i or the end of the list



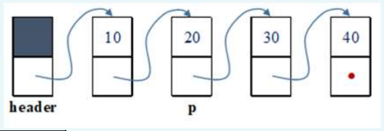
```
ALGORITHM endList(L):
    p ← L
    j ← 0
    while p->next != NULL:
        p ← p->next
return p
```

T(n) = O(n)



Get the pointer referring to the next element

- Example: next (p, L) → p->next
- Start from the first pointer (L), move to the next pointer, update position. Loop until position i or the end of the list



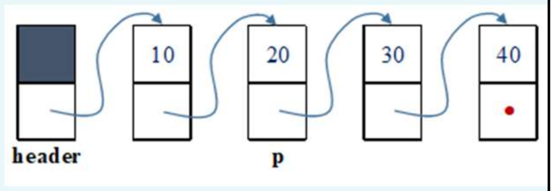
```
ALGORITHM next(i, L):
    p ← L
    j ← 0
    while p->next != NULL and j < i:
        j++
        p ← p->next
return p-next
```

T(n) = O(n)

Get/Set

- Get the element at position i
  - Retrieve the pointer at position i, return the appropriate element
    - `getAt(2, L)` → 30

```
ALGORITHM getAt(i, L):
    p <- getPosition(i, L)
    return p->next->data
```



- Update element at position i

```
ALGORITHM setAt(x, i, *pL):
    p <- getPosition(i, *pL)
    p->next->data <- x
```

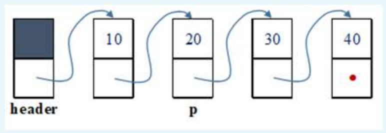
- Both cost T(n) = O(n)

www.ctu.edu.vn 21

21

Insert an element to position i

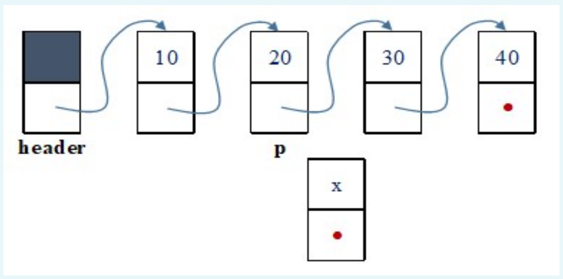
- Example:



  - insertAt(x =100, i = 2, &L) → [10, 20, 100, 30, 40]

- Algorithm

```
ALGORITHM insertAt(x, i, *pL):
    p ← getPosition(i, *pL)
    q ← malloc(Node)
    q->data ← x
    q->next ← p->next
    p->next ← q
```

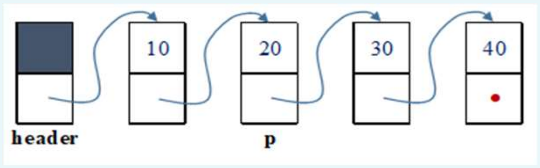


www.ctu.edu.vn 22

22

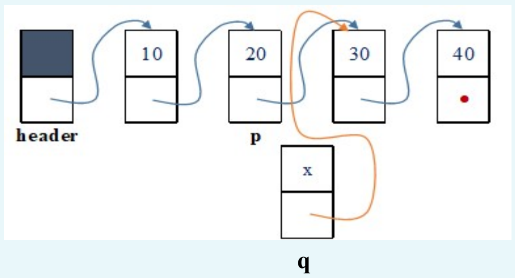
Insert an element to position i

- Example:



  - insertAt(x =100, i = 2, &L) → [10, 20, 100, 30, 40]
- Algorithm

```
ALGORITHM insertAt(x, i, *pL):
    p ← getPosition(i, *pL)
    q ← malloc(Node)
    q->data ← x
    q->next ← p->next
    p->next ← q
```

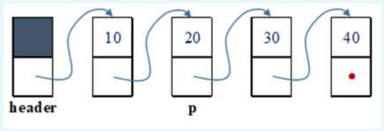


www.ctu.edu.vn 23

23

Insert an element to position i

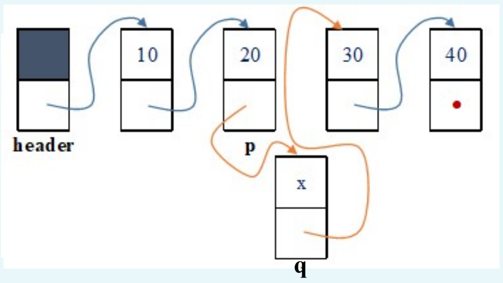
- Example:



  - insertAt(x =100, i = 2, &L) → [10, 20, 100, 30, 40]
- Algorithm

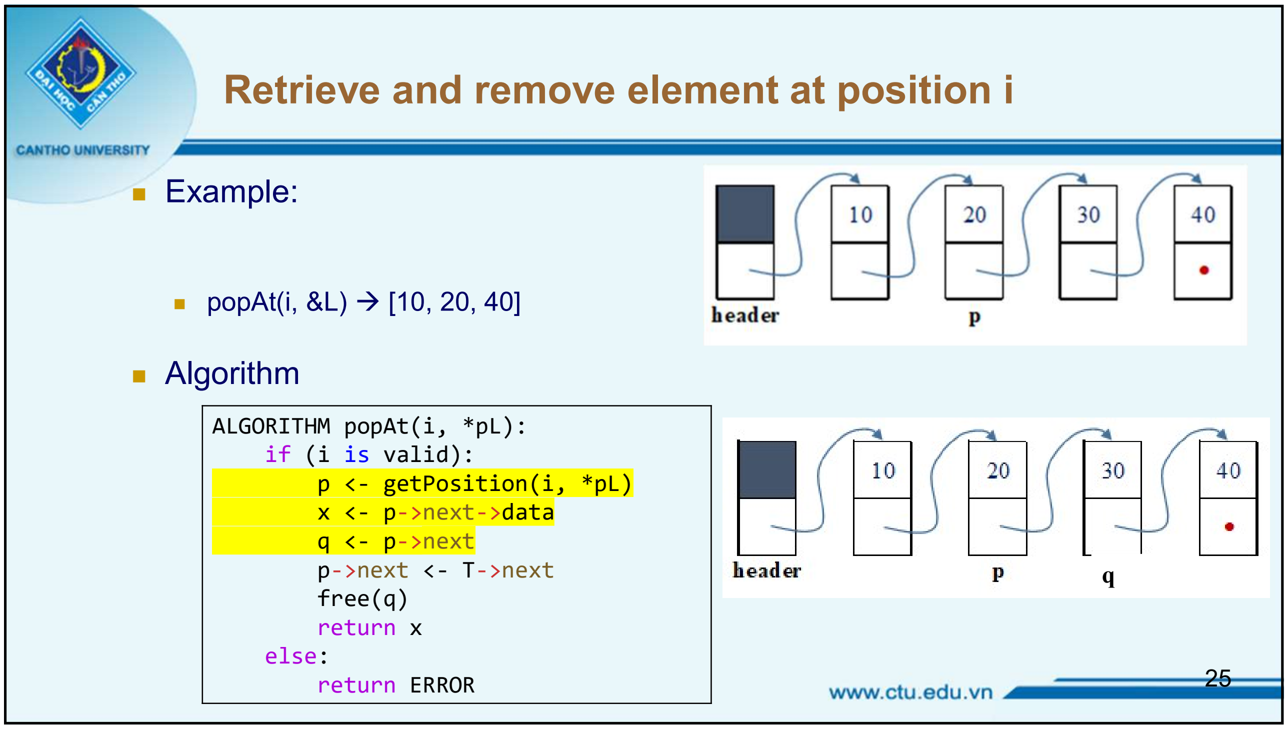
```
ALGORITHM insertAt(x, i, *pL):
    p ← getPosition(i, *pL)
    q ← malloc(Node)
    q->data ← x
    q->next ← p->next
    p->next ← t
```

T(n) = O(n)



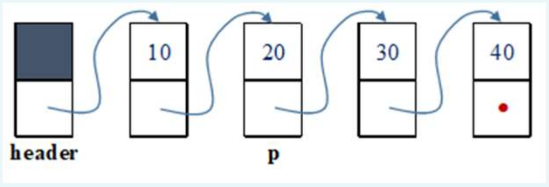
www.ctu.edu.vn 24

24



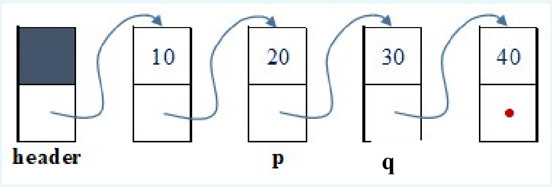
Retrieve and remove element at position i

- Example:
  - popAt(i, &L) → [10, 20, 40]

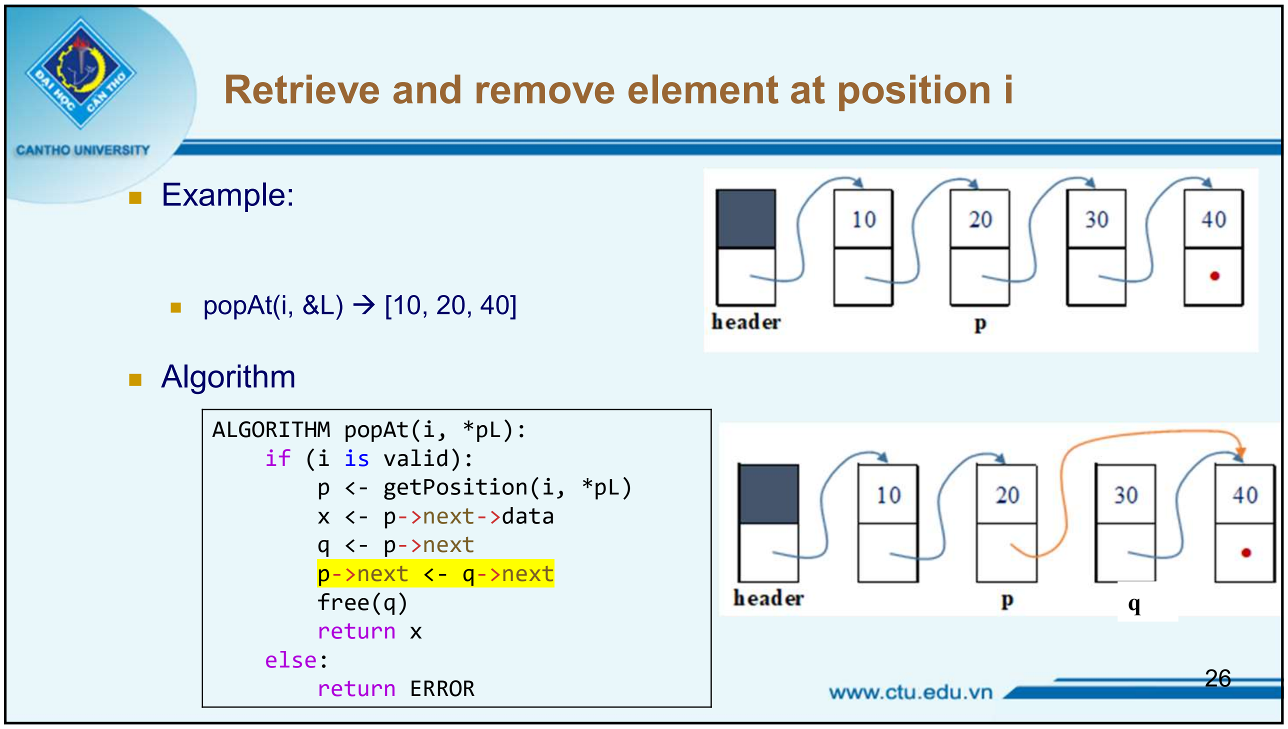


- Algorithm

```
ALGORITHM popAt(i, *pL):
    if (i is valid):
        p <- getPosition(i, *pL)
        x <- p->next->data
        q <- p->next
        p->next <- T->next
        free(q)
        return x
    else:
        return ERROR
```

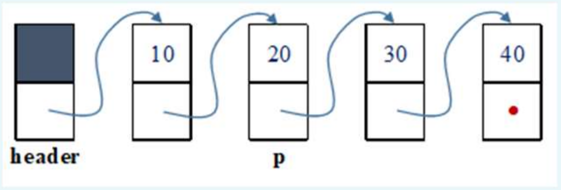


www.ctu.edu.vn 25



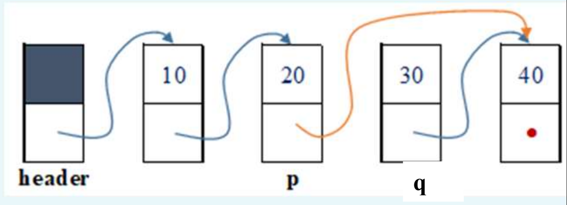
Retrieve and remove element at position i

- Example:
  - popAt(i, &L) → [10, 20, 40]



- Algorithm

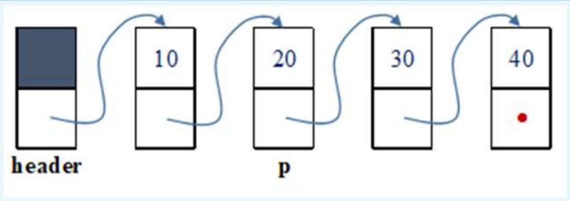
```
ALGORITHM popAt(i, *pL):
    if (i is valid):
        p <- getPosition(i, *pL)
        x <- p->next->data
        q <- p->next
        p->next <- q->next
        free(q)
        return x
    else:
        return ERROR
```



www.ctu.edu.vn 26

Retrieve and remove element at position i

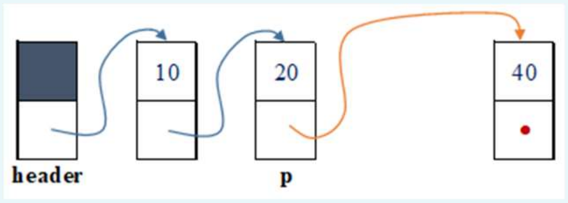
- Example:
  - popAt(i, &L) → [10, 20, 40]



- Algorithm

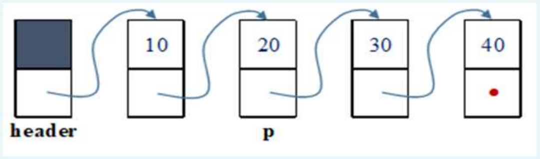
```
ALGORITHM popAt(i, *pL):
    if (i is valid):
        p <- getPosition(i, *pL)
        x <- p->next->data
        q <- p->next
        p->next <- q->next
        free(q)
        return x
    else:
        return ERROR
```

T(n) = O(n)



Insert to the first position

- Example
  - Example: `insertFirst(x = 100, &L)`
  - `L = [100, 10, 20, 30, 40]`



- Mã giả

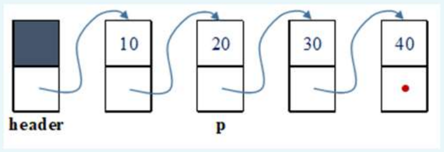
```
ALGORITHM insertFirst(x, *pL):
    q <- malloc(Node)
    q->data <- x
    q->next <- (*pL)->next
    (*pL)->next <- q
```

- T(n) = O(1)

CANTHO UNIVERSITY

Retreve and remove the first element

- Example:
  - popFirst(&L) → 10, L =[10, 20, 30, 40]



- Pseudo-code

```
ALGORITHM popFirst(*pL):
```

- T(n) = O(...)

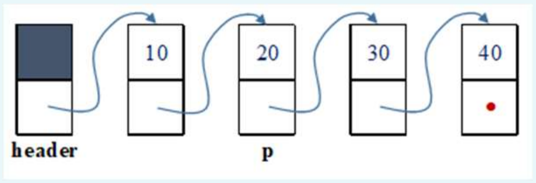
www.ctu.edu.vn 29

29

CANTHO UNIVERSITY

Append an element to the list

- Example
  - Append (x=100, &L) → L = [10, 20, 30, 40, 100]



- Pseudo-code

```
ALGOTHIM append(x, *pL):
    d <- len(*pL)
    p <- getPosition (d, *pL)
    insertAt(x, d, pL)
```

- T(n) = O(...)

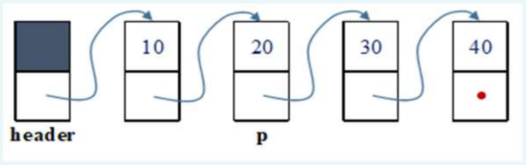
www.ctu.edu.vn 30

30

Retrieve and remove the last element

- Example
  - `popLast(&L) → 40; L = [10, 20, 30]`
- Pseudo-code

```
ALGOTHIM popLast(x, *pL):
```

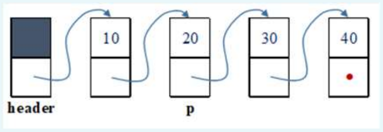


- T(n) = O(...)

Find an element in the list

- Look up an element in the list and return the pointer referring to it
- Example
  - `locate(x = 30, L) → p; //`
- Start from the first pointer, traverse the list until see the first occurrence of x. If not found, return the last pointer
- Pseudo-code

```
ALGORITHM locate(x, L):
    p = L
    while (p->next != NULL):
        if (x == p->next->data):
            return p
        p = p->next
    return p
```



- T(n) = O(...)

Agenda

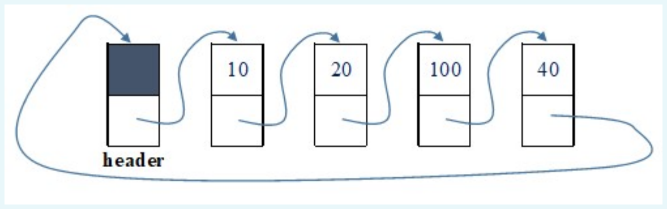
- Pointer-based list
- Operators
- Other linked lists
- Summary

www.ctu.edu.vn 33

33

Circle linked list

- Elements link together to form a circle



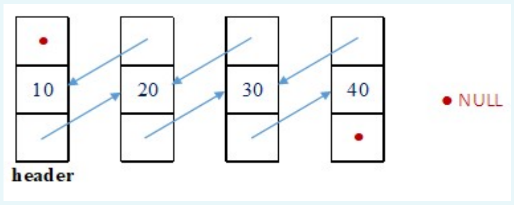
```
ALGORITHM makenull(*pL):
    header ← malloc(sizeof(Node))
    header->next ← header
    (*pL) ← header
```

www.ctu.edu.vn 34

34

Doule linked list

- Each node has another pointer to previous node



- Declaration

```
typedef ... ElementType;
typdef struct NodeTag{
      ElementType data;
      struct NodeTag *next; //Add. of next node
      struct NodeTag *previous;//Add. of prev. node
} Node;
```

www.ctu.edu.vn 35

35

Agenda

- Pointer-based list
- Operators
- Other linked lists
- Summary

www.ctu.edu.vn 36

36

Summary

DS	Construction	Static	Dynamic		
	makenull()	getAt() setAt()	insertAt() popAt()	insertFirst() popFirst()	insertLast() popLast()
Array	O(1)	O(1)	O(n)	O(n)	O(1)
Linked List	O(1)	O(n)	O(n)	O(1)	O(n)

www.ctu.edu.vn 37

37

Thanks for your attention!

www.ctu.edu.vn 38

38

CANTHO UNIVERSITY

Q&A

www.ctu.edu.vn

39