자료구조

07 연결 리스트

2014년 1학기

미디어소프트웨어학과
민경하

학기내용

- 소개
- 성능분석
- 리스트
- 배열
- 스택
- 큐
- 연결 리스트
- 정렬
- 탐색
- 트리
- 그래프

내용

7.1 소개

- 자료구조의 분류

구성	이름	구현	
		인덱스	포인터
1차원	배열	O	
	스택	O	
	큐	O	
	연결 리스트		O
계층적	트리		O
-	그래프	O	O

7.1 소개

- 리스트 → 목록

No.	Name	Qnty	Origin	Destination
1	Beer	10	Germany	Korea
2	Gum	5	USA	China
3	Apple	4	Chile	Korea
4	Potato	8	Korea	Japan
5	Onion	8	Egypt	Israel
…	…	…	…	…

7.1 소개

- 리스트는 무엇인가?
  - 연속적인 대응 (mapping)
  - 원소와 인덱스는 1:1 대응이 성립함
  - 리스트의 두 가지 구현
    - 연속 (sequential) 리스트 → 인덱스 기반 구현 → 배열
    - 연결 (linked) 리스트 → 포인터 기반 구현

7.2 리스트 구현의 두 가지

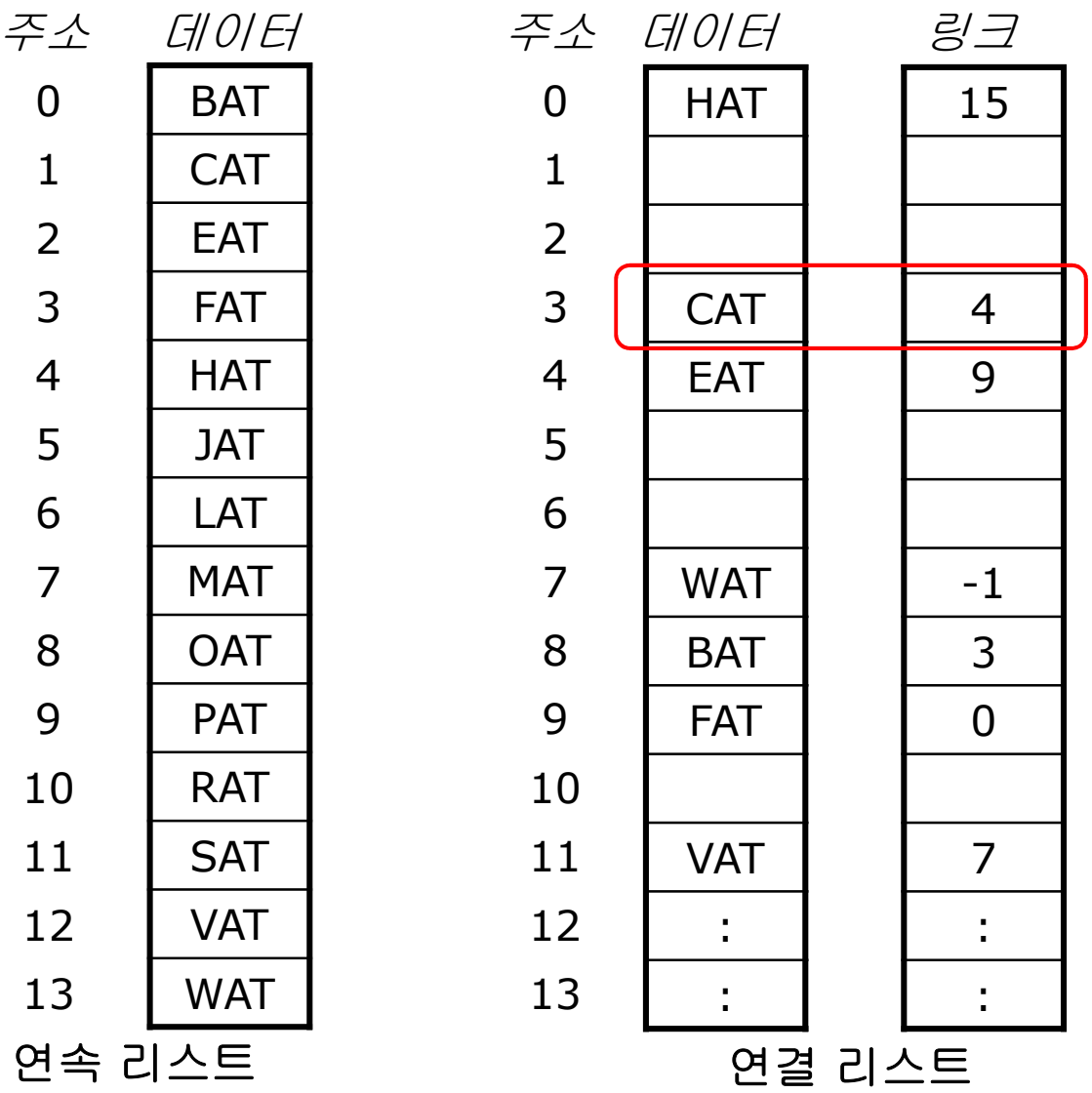
- 연속 리스트
  - 연속된 원소들은 기억 공간의 **연속된 위치**에 저장됨
  - 원소들 사이의 **간격은 일정**함 → 인덱스를 이용한 접근
    - a_i가 L_i 에서 저장됨 ➔ a_{i+1}은 $L_i + d$에 저장됨

      ➔ a_{i-1}은 $L_i - d$에 저장됨
- 연결 리스트
  - 연속된 원소들은 기억 공간의 **임의의 위치**에 저장됨
  - 각 원소들은 다음 원소를 **가리키는 링크**를 가지고 있음

7.2 리스트 구현의 두 가지

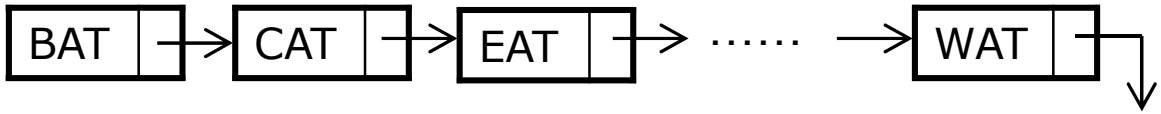
- 연속 리스트
  - 데이터
- 연결 리스트
  - 데이터 + 링크

➔ 노드



7.2 리스트 구현의 두 가지

- 연결 리스트
  - 데이터 + 링크
  - ➔ 노드

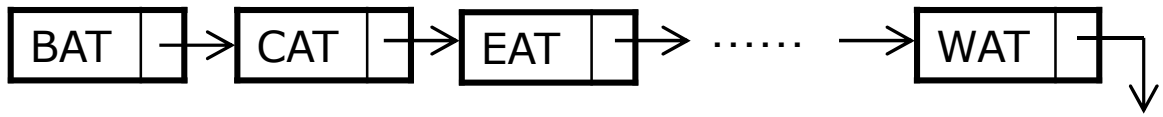


주소	데이터	링크
0	HAT	15
1		
2		
3	CAT	4
4	EAT	9
5		
6		
7	WAT	-1
8	BAT	3
9	FAT	0
10		
11	VAT	7
12	:	:
13	:	:

연결 리스트

7.3 연결 리스트 정의

- 단일 연결 리스트 (singly linked list)
  - 각 노드가 하나의 링크를 가짐
  - 다음 원소를 가리키는 링크만 존재함

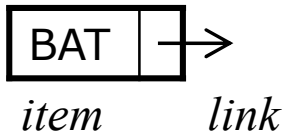


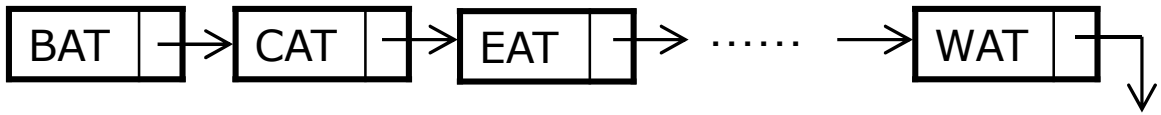
- 체인 (chain)
  - 0개 이상의 노드로 구성된 단일 연결 리스트

7.3 연결 리스트 정의

- 노드에 대한 자료구조

```
typedef class node *nptr;

class node {
    date_type item;
    nptr link;
};
```





7.4 연결 리스트 구현

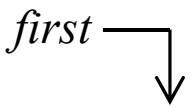
- 연결 리스트에 대한 연산
  (1) 생성 (create)
  (2) 검색 (search)
  (3) 삽입 (insert)
  (4) 제거 (remove)
  (5) 갱신 (modify)
  (6) 길이 (length)
  (7) 출력 (print)
  (8) ……

7.4 연결 리스트 구현

(1) 생성 (create)

– 텅 빈 연결 리스트를 생성하는 연산

```
void main ( )
{
    nptr first;
}
```

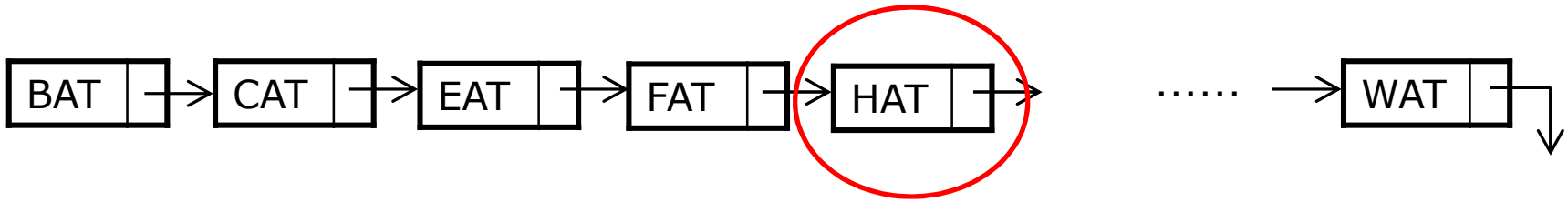


7.4 연결 리스트 구현

(2) 검색 (search)

– 우리가 찾는 정보를 저장하는 노드를 찾는 연산

```
void main ( )
{
     nptr temp = first->search ( "HAT" );
}
```



7.4 연결 리스트 구현

(2) 검색 (search)

```
nptr node::search ( data_type item )
{
    nptr curr = this;       //      this의 의미는?

    while ( curr != NULL ) {
        if ( curr->item == item )
            return curr;
        curr = curr->link;
    }

    return NULL;
}
```

7.4 연결 리스트 구현

(2) 검색 (search)

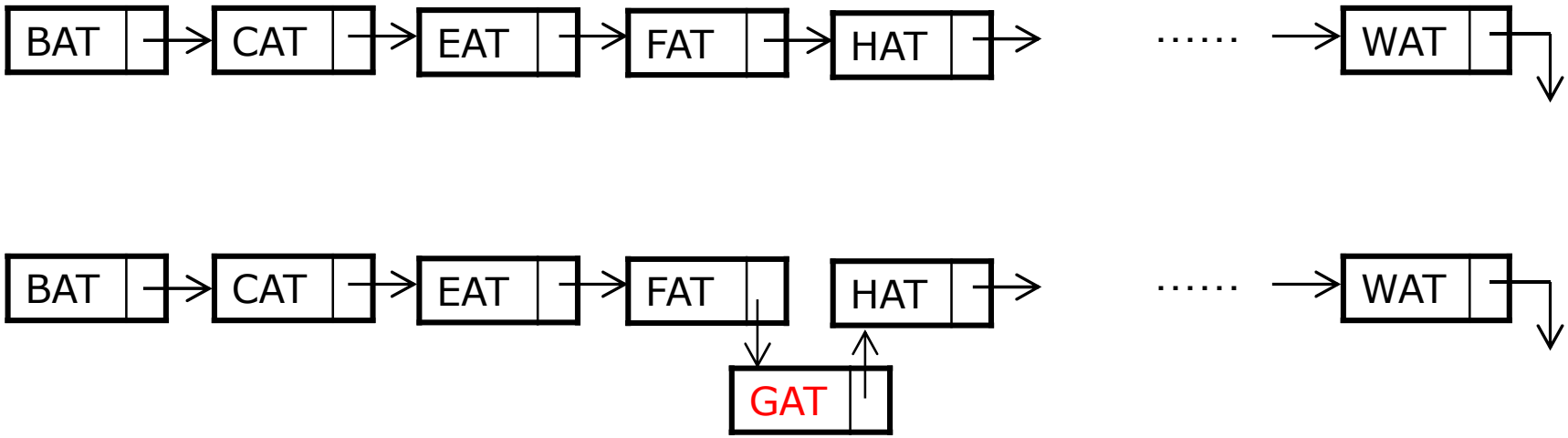
```
nptr node::search ( data_type item )
{
    nptr curr;
//  this의 의미는?
    for ( curr = this; curr != NULL; curr = curr->link ) {
        if ( curr->item == item )
            return curr;
    }

    return NULL;
}
```

7.4 연결 리스트 구현

(3) 삽입 (insert)

- 새로운 데이터를 연결 리스트에 추가하는 연산

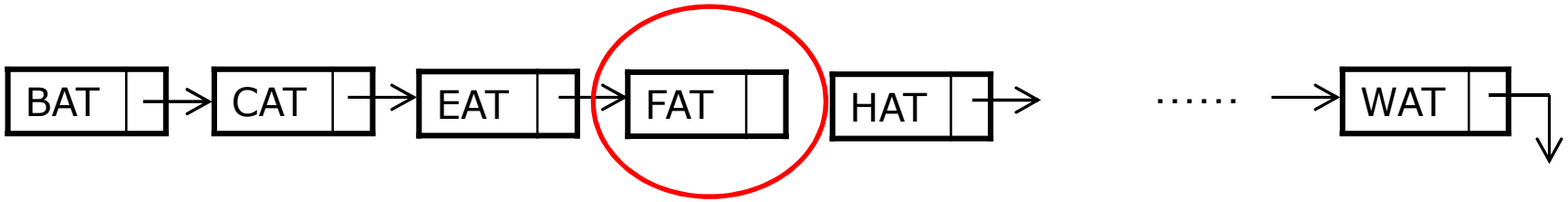
```
void main ( )
{
    first->insert ( "GAT" );
}
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

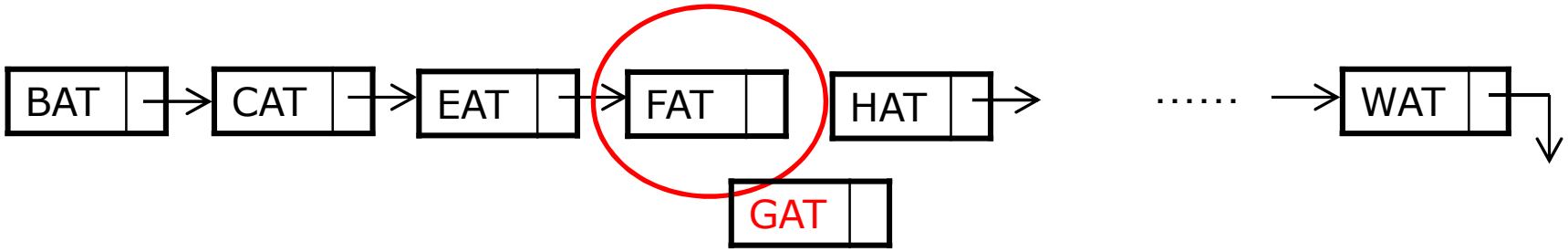
1. 데이터를 삽입할 적절한 위치를 설정



7.4 연결 리스트 구현

(3) 삽입 (insert)

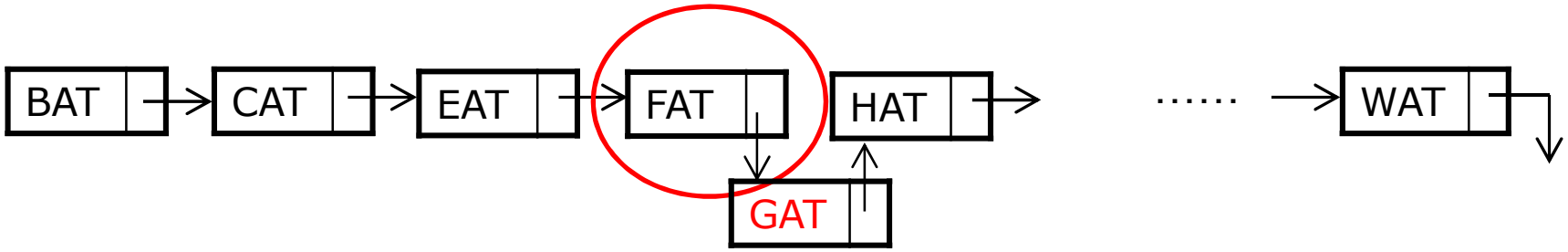
1. 데이터를 삽입할 적절한 위치를 설정
2. 삽입할 데이터를 저장하는 새로운 노드를 생성



7.4 연결 리스트 구현

(3) 삽입 (insert)

1. 데이터를 삽입할 적절한 위치를 설정
2. 삽입할 데이터를 저장하는 새로운 노드를 생성
3. 링크를 변경해서 노드를 연결 리스트에 삽입



7.4 연결 리스트 구현

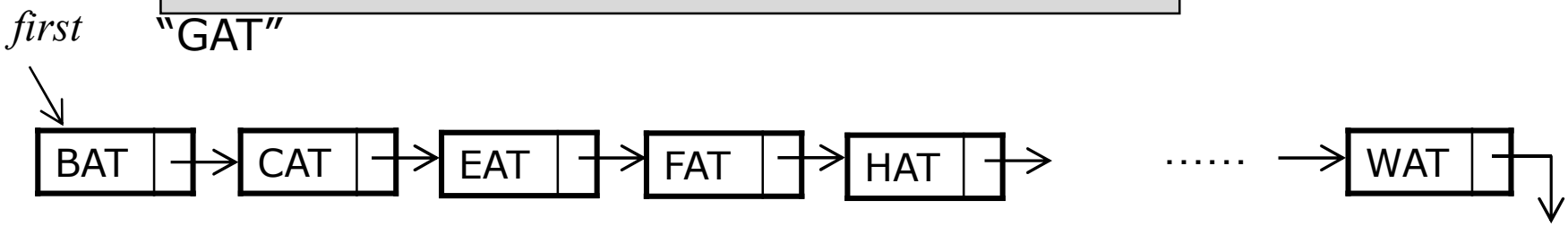
(3) 삽입 (insert)

```
void node::insert ( data_type item )
{
//  1. 삽입할 위치를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item > item )
            break;
        curr = curr->link;
    }
//  2. 새로운 노드를 생성
    nptr nnode = (nptr) malloc ( sizeof(node) );
    nnode->item = item;
//  3. 링크를 변경해서 노드를 리스트에 삽입
    nnode->link = curr->link;
    curr->link = nnode;
}
```

7.4 연결 리스트 구현

(3) 삽입 (insert)

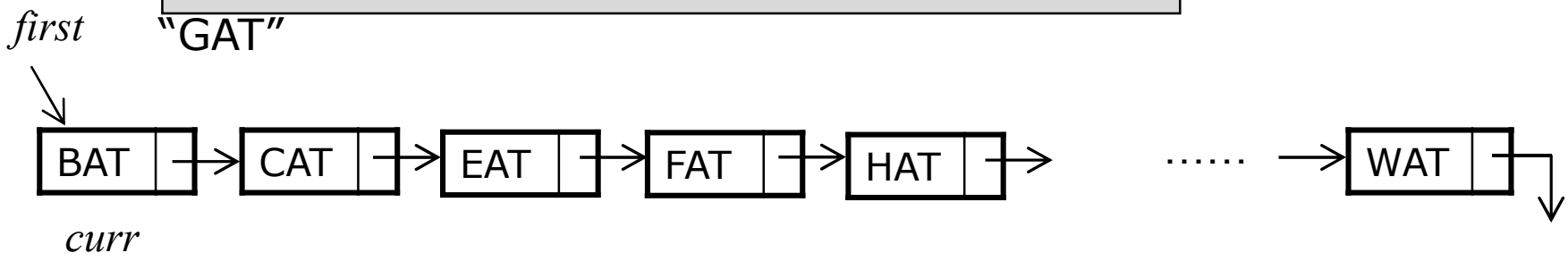
```
void node::insert ( data_type item )
{
//  1. 삽입할 위치를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item > item )
            break;
        curr = curr->link;
    }
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

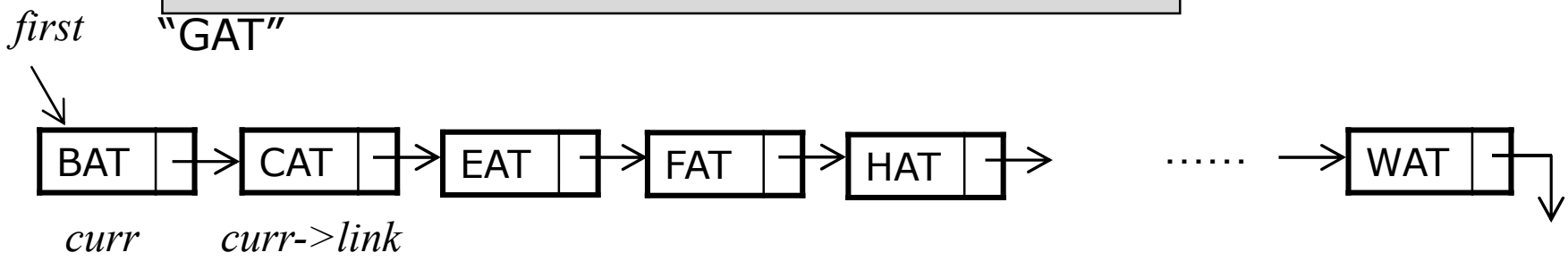
```
void node::insert ( data_type item )
{
//  1. 삽입할 위치를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item > item )
            break;
        curr = curr->link;
    }
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

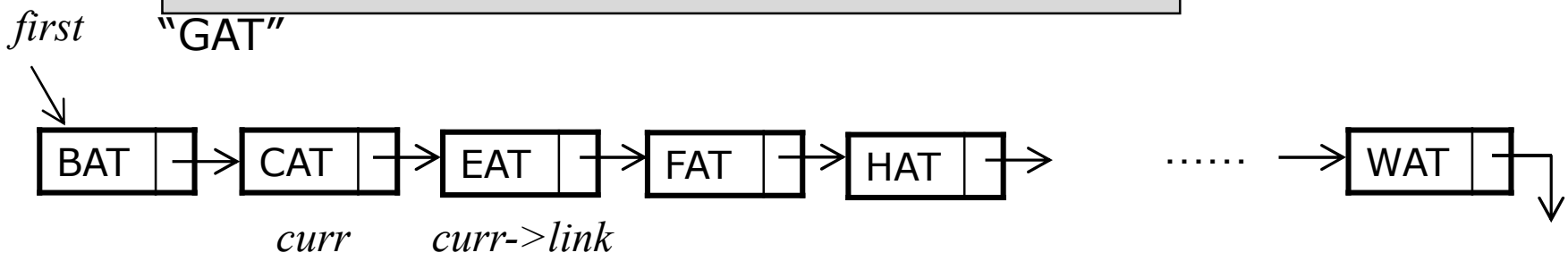
```
void node::insert ( data_type item )
{
//  1. 삽입할 위치를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item > item )
            break;
        curr = curr->link;
    }
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

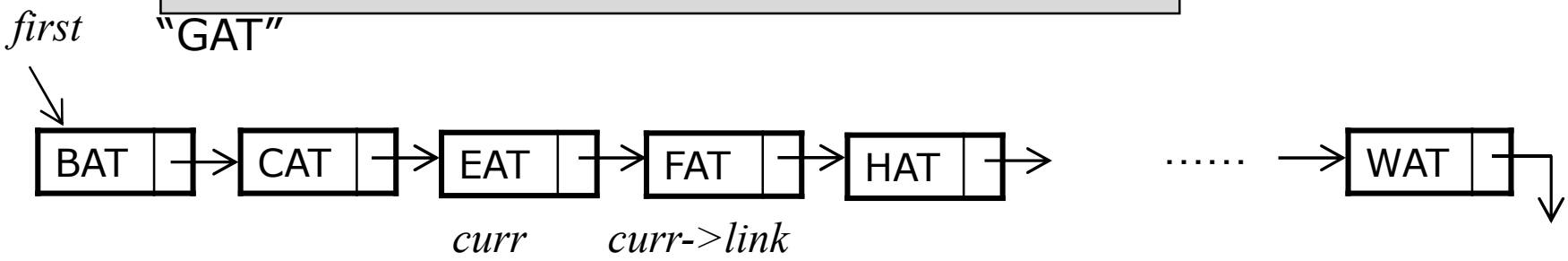
```
void node::insert ( data_type item )
{
//  1. 삽입할 위치를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item > item )
            break;
        curr = curr->link;
    }
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

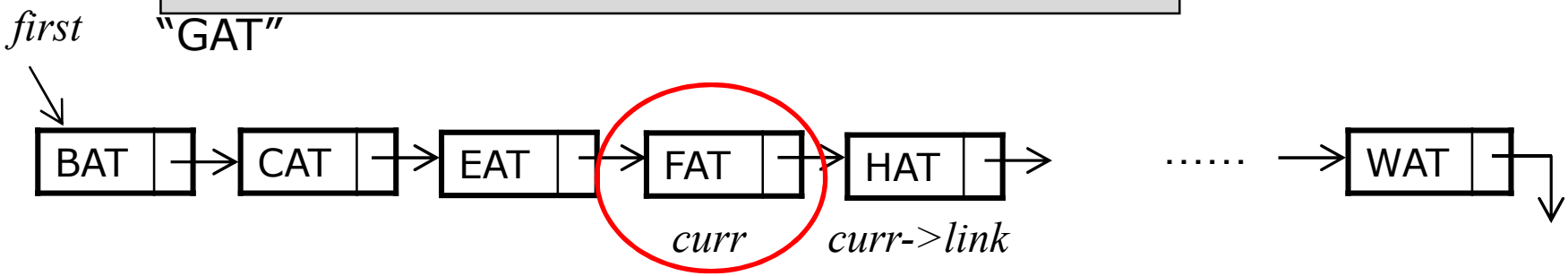
```
void node::insert ( data_type item )
{
//  1. 삽입할 위치를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item > item )
            break;
        curr = curr->link;
    }
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

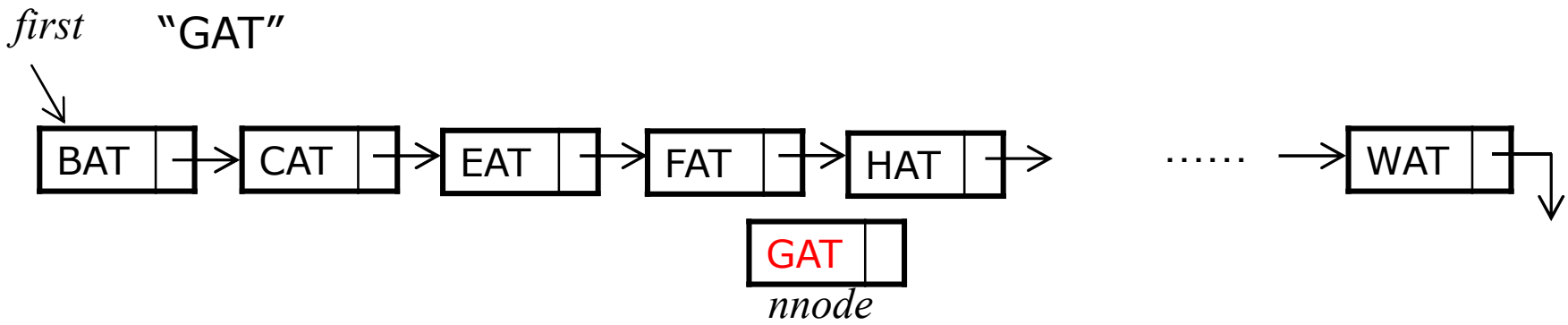
```
void node::insert ( data_type item )
{
//  1. 삽입할 위치를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item > item )
            break;
        curr = curr->link;
    }
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

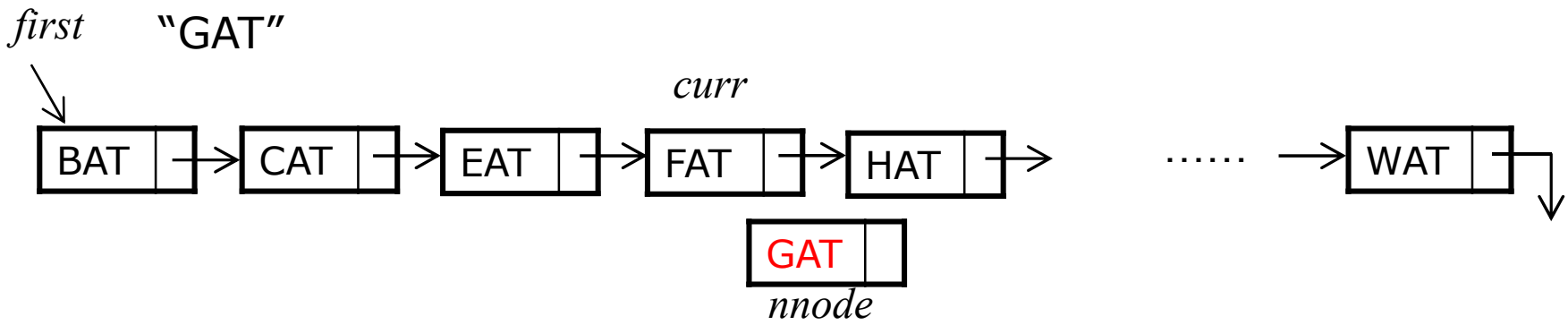
```
void node::insert ( data_type item )
{
//  2. 새로운 노드를 생성
    nptr nnode = (nptr) malloc ( sizeof(node) );
    nnode->item = item;
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

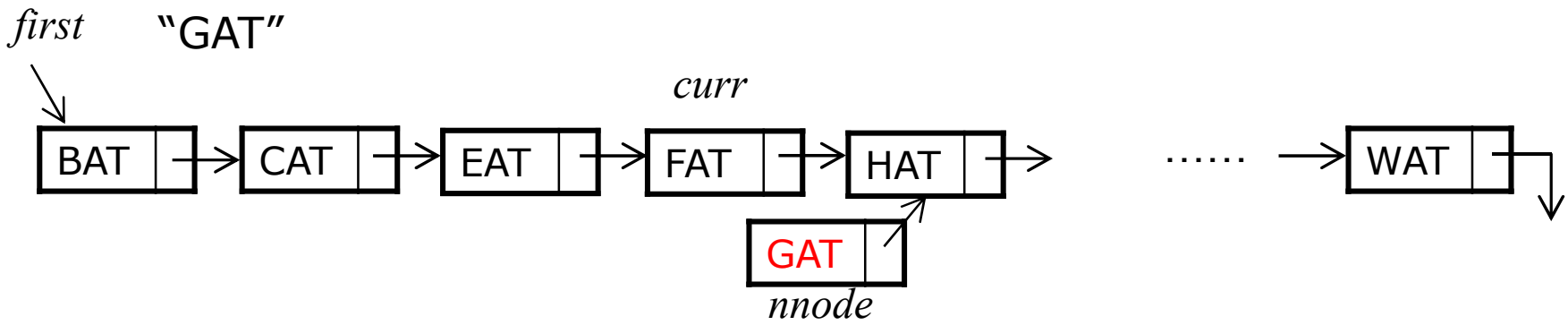
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
    nnode->link = curr->link;
    curr->link = nnode;
}
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

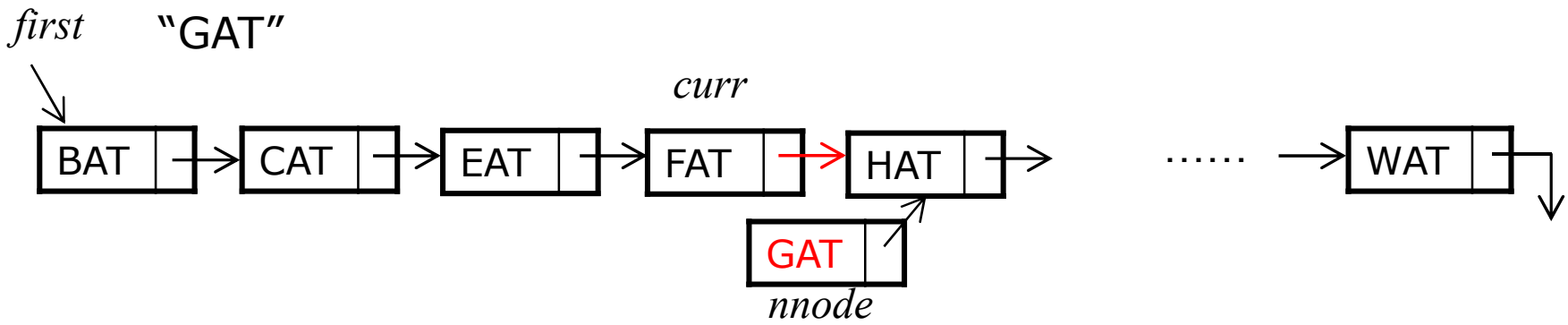
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
    nnode->link = curr->link;
    curr->link = nnode;
}
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

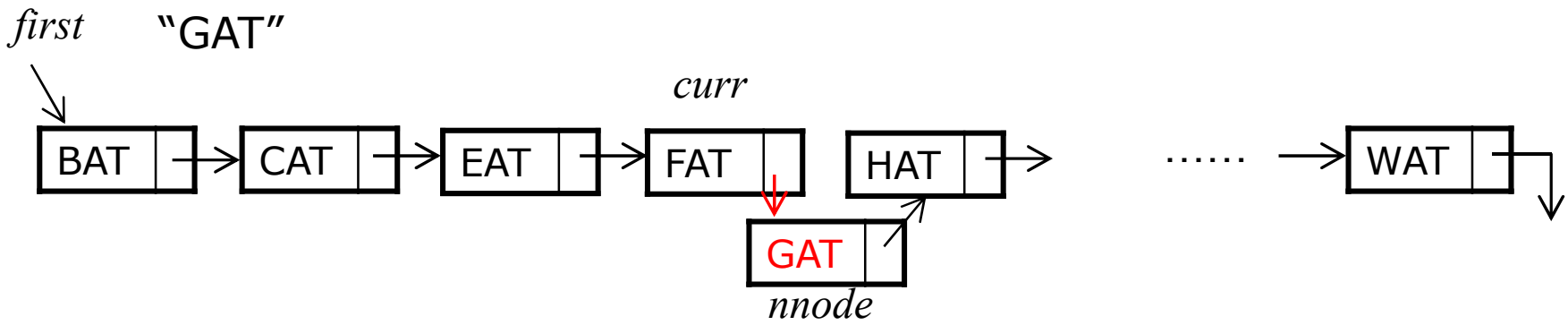
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
    nnode->link = curr->link;
    curr->link = nnode;
}
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
    nnode->link = curr->link;
    curr->link = nnode;
}
```



7.4 연결 리스트 구현

(3) 삽입 (insert)

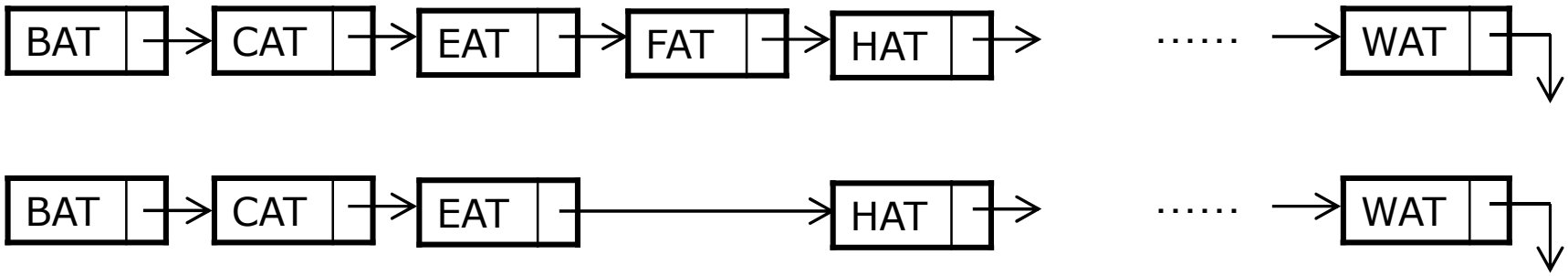
- 예외적인 경우 (degenerate case)
  - `first` 앞에 삽입하는 경우
  - `first`가 `NULL`인 경우
  - 또?

7.4 연결 리스트 구현

(4) 제거 (remove)

- 연결 리스트에서 특정 원소를 포함하는 노드를 삭제하는 연산

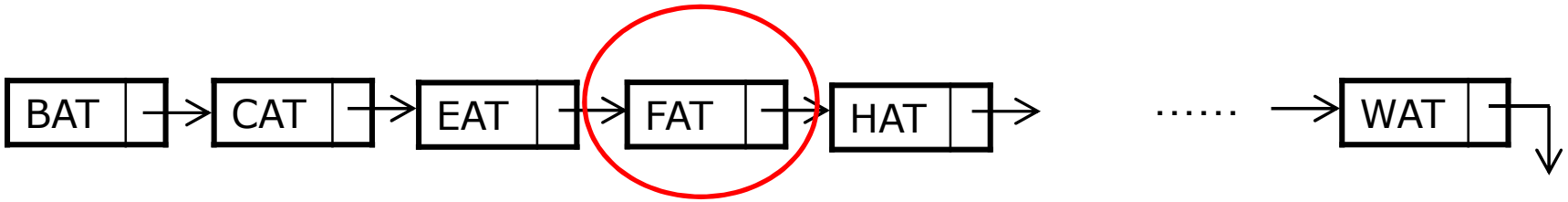
```
void main ( )
{
     first->remove ( "FAT" );
}
```



7.4 연결 리스트 구현

(4) 제거 (remove)

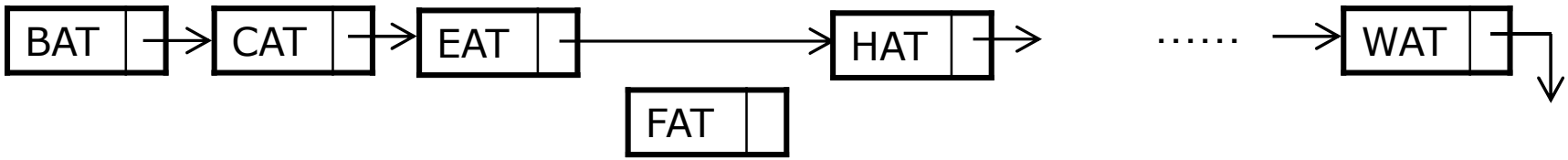
1. 삭제할 원소를 포함하는 노드를 검색



7.4 연결 리스트 구현

(4) 제거 (remove)

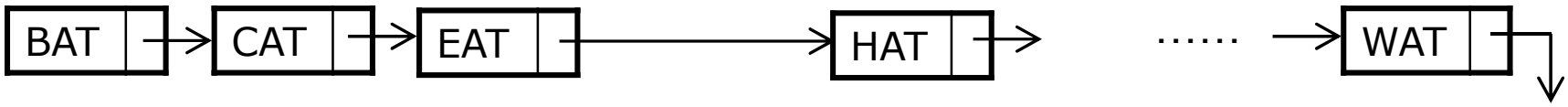
1. 삭제할 원소를 포함하는 노드를 검색
2. 연결 리스트의 링크를 변경해서 노드를 리스트로부터 삭제



7.4 연결 리스트 구현

(4) 제거 (remove)

1. 삭제할 원소를 포함하는 노드를 검색
2. 연결 리스트의 링크를 변경해서 노드를 리스트로부터 삭제
3. 삭제된 노드를 free함



7.4 연결 리스트 구현

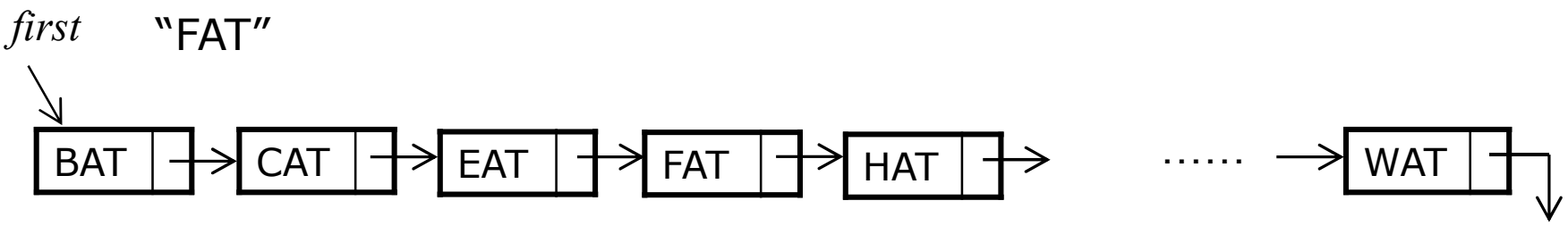
(4) 제거 (remove)

```
void node::delete ( data_type item )
{
//  1. 삭제할 원소를 포함하는 노드를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item == item )
            break;
        curr = curr->link;
    }
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->link;
    curr->link = dnode->link;
//  3. 삭제된 노드를 free함
    free ( dnode );
}
```

7.4 연결 리스트 구현

(4) 제거 (remove)

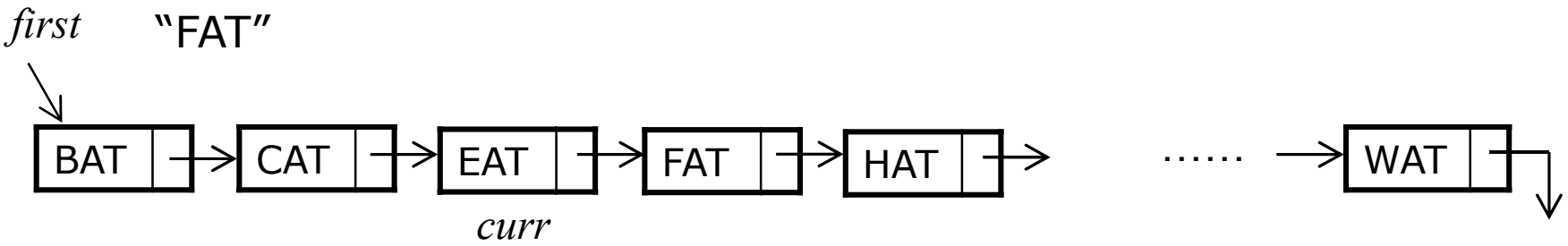
```
void node::delete ( data_type item )
{
//  1. 삭제할 원소를 포함하는 노드를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item == item )
            break;
        curr = curr->link;
    }
```



7.4 연결 리스트 구현

(4) 제거 (remove)

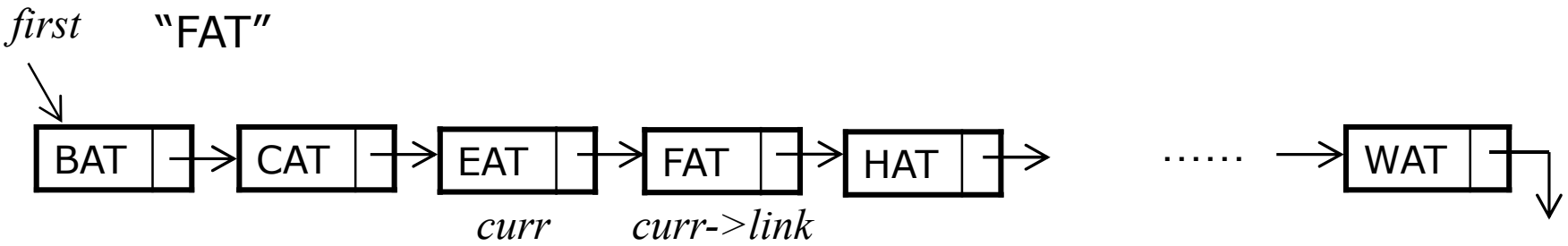
```
void node::delete ( data_type item )
{
//  1. 삭제할 원소를 포함하는 노드를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item == item )
            break;
        curr = curr->link;
    }
```



7.4 연결 리스트 구현

(4) 제거 (remove)

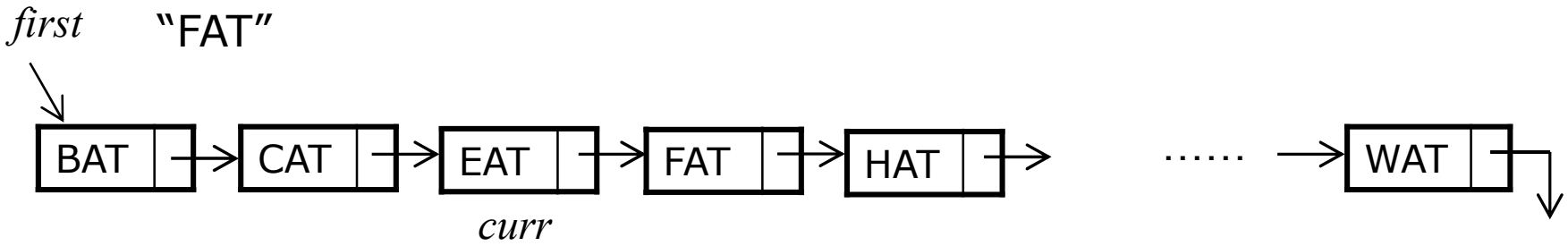
```
void node::delete ( data_type item )
{
//  1. 삭제할 원소를 포함하는 노드를 찾음
    nptr curr = this;
    while ( curr->link != NULL ) {
        if ( curr->link->item == item )
            break;
        curr = curr->link;
    }
```



7.4 연결 리스트 구현

(4) 제거 (remove)

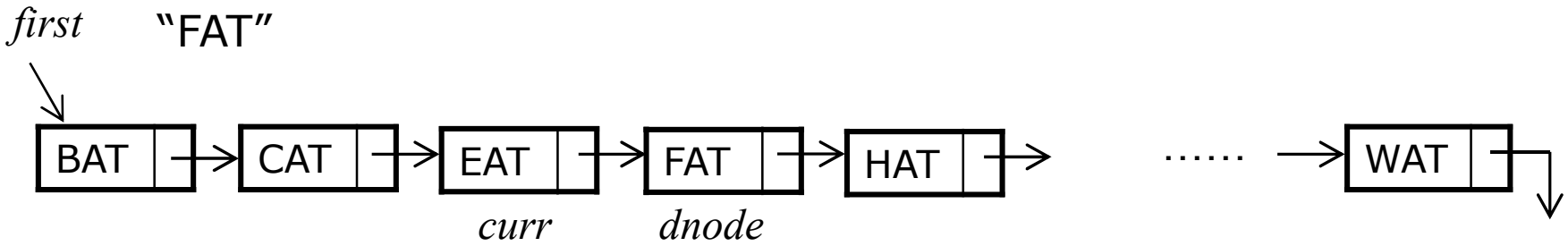
```
void node::delete ( data_type item )
{
//  2 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->link;
    curr->link = dnode->link;
}
```



7.4 연결 리스트 구현

(4) 제거 (remove)

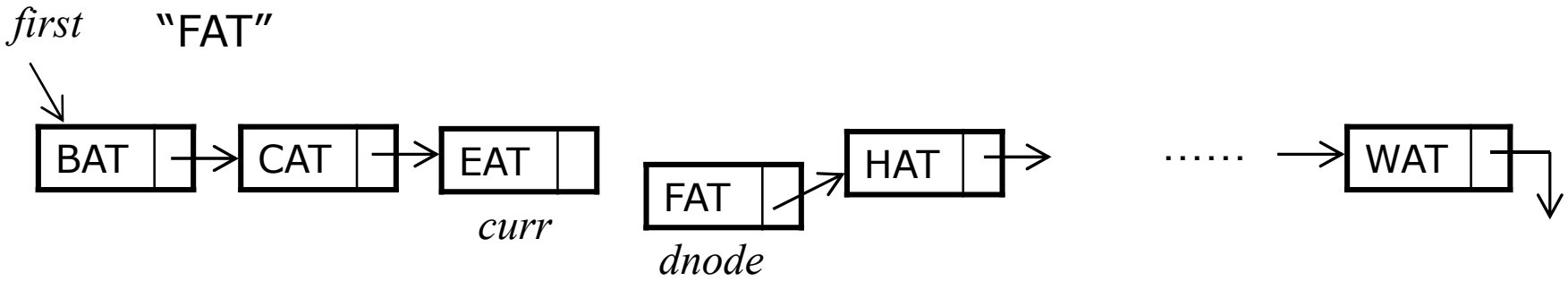
```
void node::delete ( data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->link;
    curr->link = dnode->link;
}
```



7.4 연결 리스트 구현

(4) 제거 (remove)

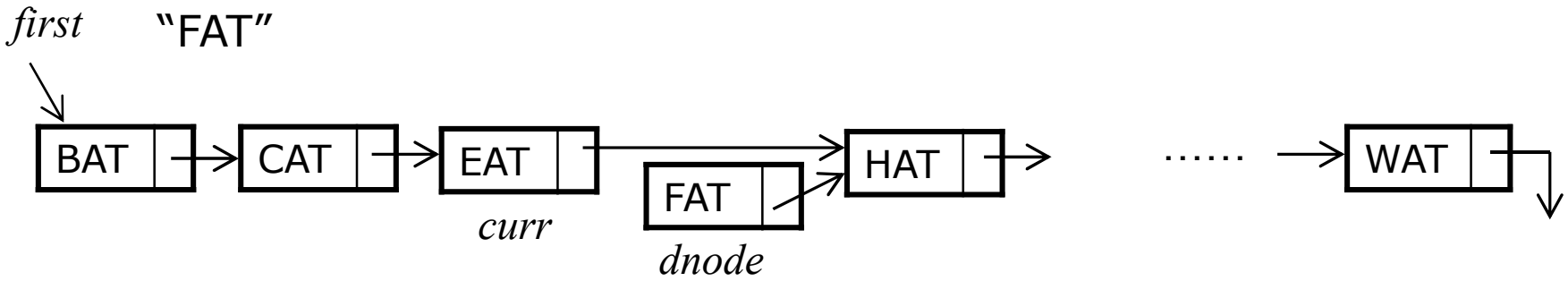
```
void node::delete ( data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->link;
    curr->link = dnode->link;
}
```



7.4 연결 리스트 구현

(4) 제거 (remove)

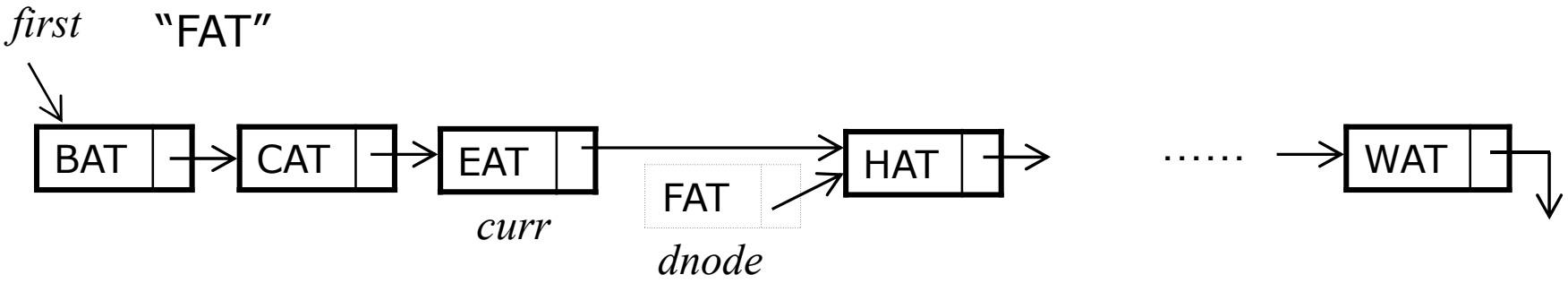
```
void node::delete ( data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->link;
    curr->link = dnode->link;
}
```



7.4 연결 리스트 구현

(4) 제거 (remove)

```
void node::delete ( data_type item )
{
//  3. 삭제된 노드를 free함
    free ( dnote );
}
```



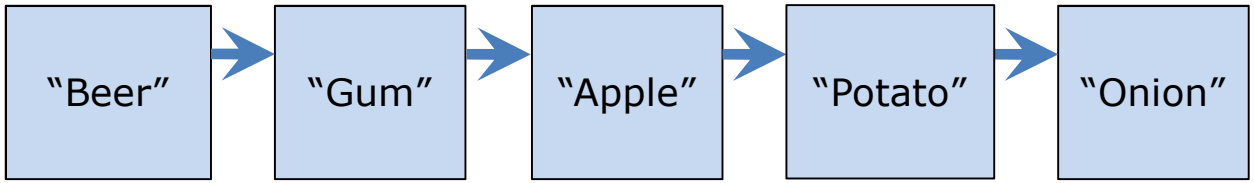
7.4 연결 리스트 구현

(4)제거 (remove)

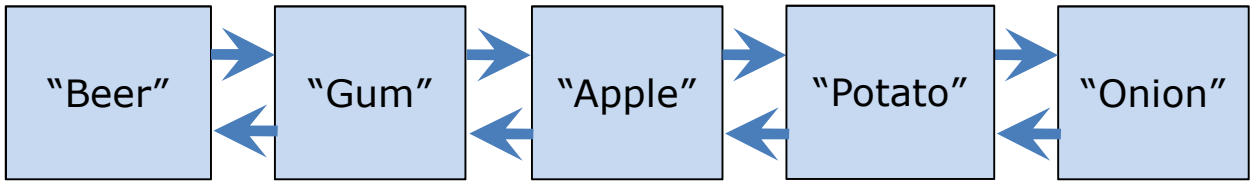
- 예외적인 경우
  - `first`를 제거할 경우
  - `first`가 `NULL`인 경우

7.5 이중 연결 리스트

- 단일 연결 리스트



- 이중 연결 리스트

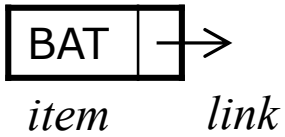


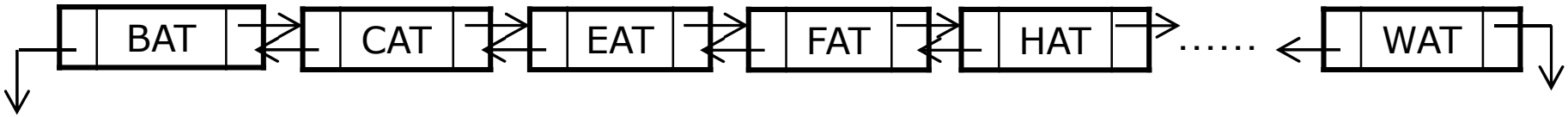
7.5 이중 연결 리스트

- 단일 연결 리스트
- 이중 연결 리스트

```
class node *nptr;
class node {
    data_type item;
    nptr link;
};
```

```
class node *nptr;
class node {
    data_type item;
    nptr llink, rlink;
};
```





7.5 이중 연결 리스트

- 연산

  (1) 삽입

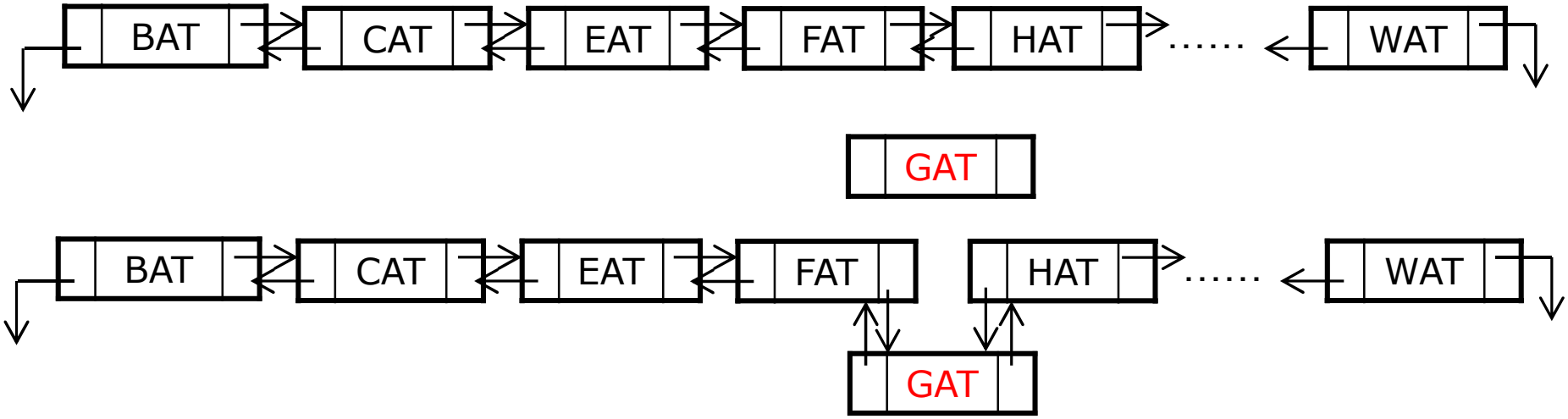
  (2) 제거

7.5 이중 연결 리스트

(1) 삽입

- 새로운 데이터를 연결 리스트에 추가하는 연산

```
void main ( )
{
    first->insert ( "GAT" );
}
```



7.5 이중 연결 리스트

(1) 삽입

1. 데이터를 삽입할 적절한 위치를 설정
2. 삽입할 데이터를 저장하는 새로운 노드를 생성
3. **링크를 변경**해서 노드를 연결 리스트에 삽입

7.5 이중 연결 리스트

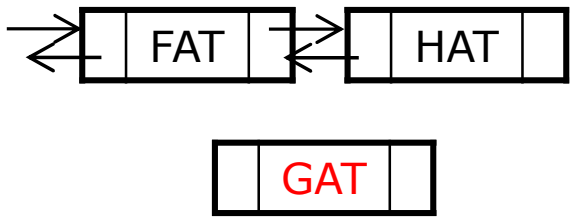
(1) 삽입

```
void node::insert ( data_type item )
{
//   1. 삽입할 위치를 찾음
     nptr curr = this;
     while ( curr->link != NULL ) {
          if ( curr->link->item > item )
               break;
          curr = curr->link;
     }
//   2. 새로운 노드를 생성
     nptr nnode = (nptr) malloc ( sizeof(node) );
     nnode->item = item;
//   3. 링크를 변경해서 노드를 리스트에 삽입
}
```

7.5 이중 연결 리스트

(1) 삽입

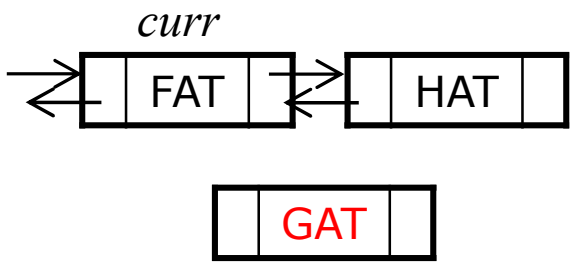
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
}
```



7.5 이중 연결 리스트

(1) 삽입

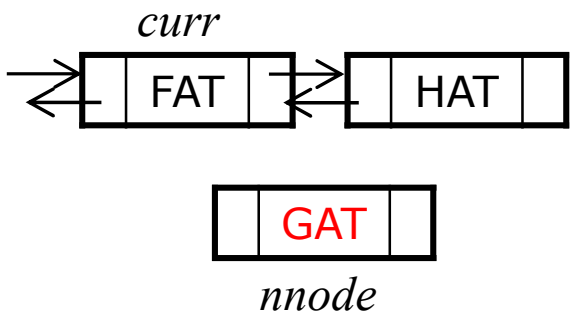
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
}
```



7.5 이중 연결 리스트

(1) 삽입

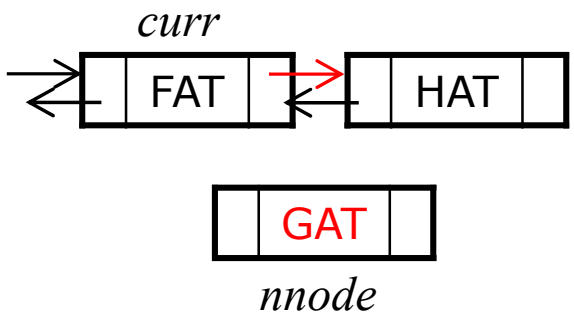
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
}
```



7.5 이중 연결 리스트

(1) 삽입

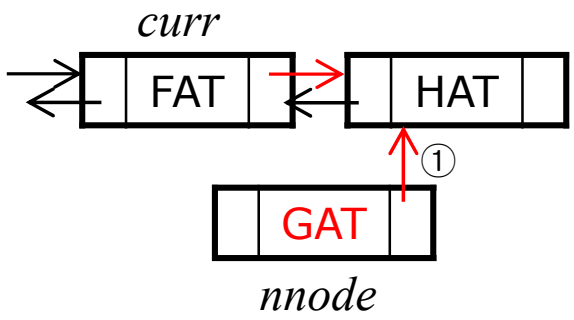
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
//  3.1 forward direction ( rlink: → )
}
```



7.5 이중 연결 리스트

(1) 삽입

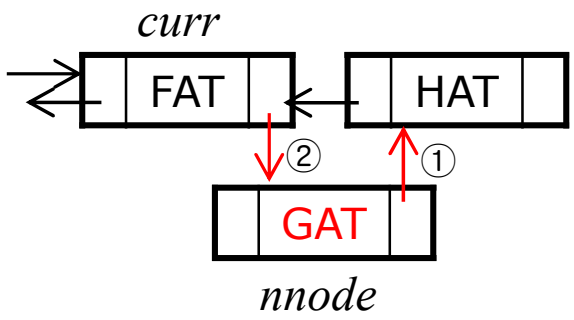
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
//  3.1 forward direction ( rlink: → )
    nnode->rlink = curr->rlink;
}
```



7.5 이중 연결 리스트

(1) 삽입

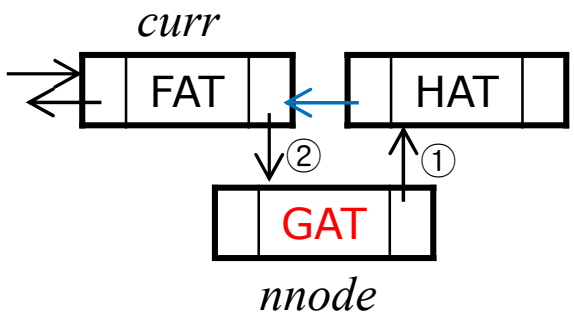
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
//  3.1 forward direction ( rlink: → )
    nnode->rlink = curr->rlink;
    curr->rlink = nnode;
}
```



7.5 이중 연결 리스트

(1) 삽입

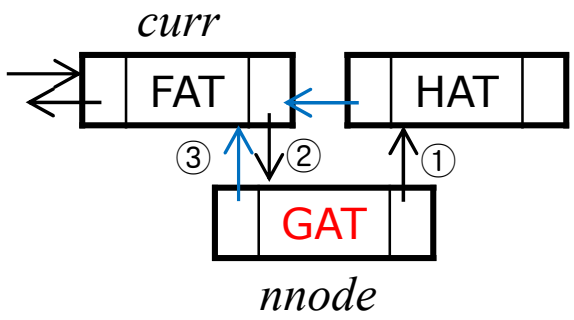
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
//  3.1 forward direction ( rlink: → )
    nnode->rlink = curr->rlink;
    curr->rlink = nnode;
//  3.2 backward direction ( llink: ← )
}
```



7.5 이중 연결 리스트

(1) 삽입

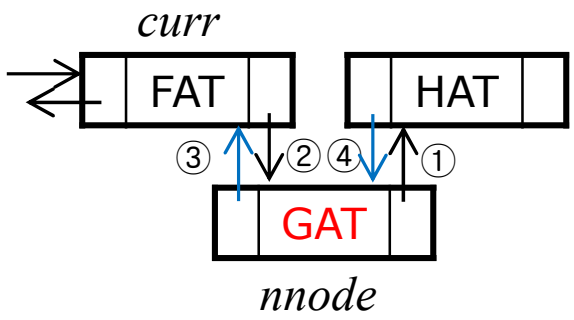
```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
//  3.1 forward direction ( rlink: → )
    nnode->rlink = curr->rlink;
    curr->rlink = nnode;
//  3.2 backward direction ( llink: ← )
    nnode->llink = curr;
}
```



7.5 이중 연결 리스트

(1) 삽입

```
void node::insert ( data_type item )
{
//  3. 링크를 변경해서 노드를 리스트에 삽입
//  3.1 forward direction ( rlink: → )
    nnode->rlink = curr->rlink;
    curr->rlink = nnode;
//  3.2 backward direction ( llink: ← )
    nnode->llink = curr;
    nnode->rlink->llink = nnode;
}
```

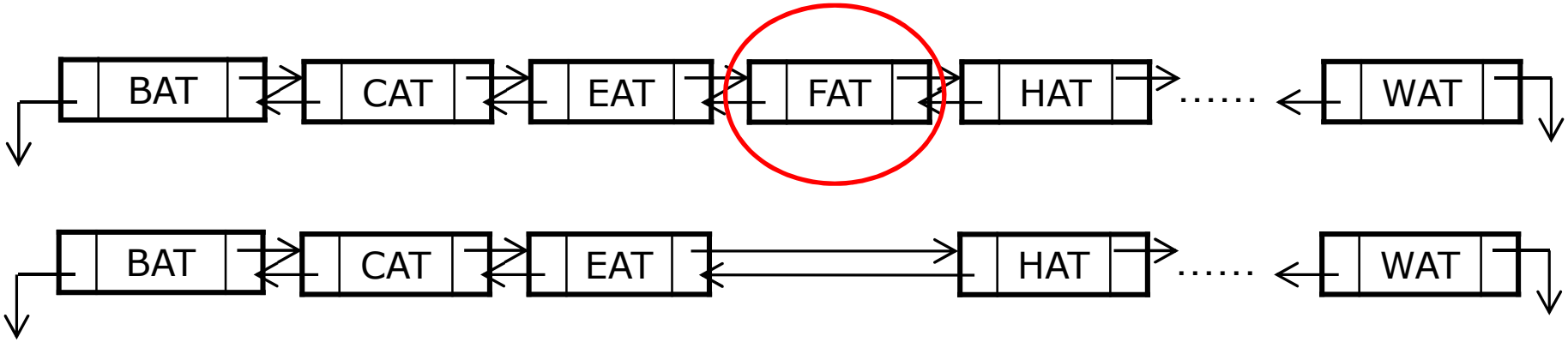


7.5 이중 연결 리스트

(2) 제거

- 연결 리스트에서 특정 원소를 포함하는 노드를 삭제하는 연산

```
void main ( )
{
    first->remove ( "FAT" );
}
```



7.5 이중 연결 리스트

(2) 제거

1. 삭제할 원소를 포함하는 노드를 검색
2. 연결 리스트의 **링크를 변경**해서 노드를 리스트로부터 삭제
3. 삭제된 노드를 free함

7.5 이중 연결 리스트

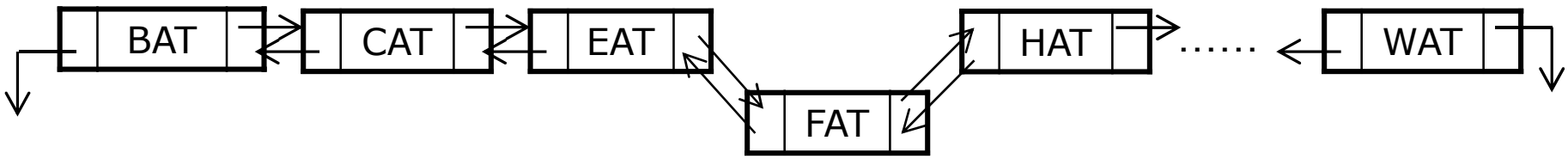
(2) 제거

```
void node::delete ( data_type item )
{
//   1. 삭제할 원소를 포함하는 노드를 찾음
     nptr curr = this;
     while ( curr->link != NULL ) {
          if ( curr->link->item == item )
               break;
          curr = curr->link;
     }
//   2. 링크를 변경해서 노드를 삭제함

//   3. 삭제된 노드를 free함
     free ( dnode );
}
```

7.5 이중 연결 리스트

(2) 제거

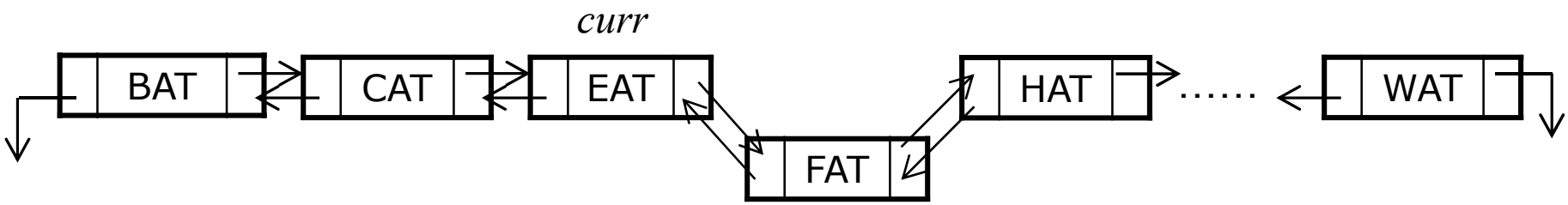
```
void Delete ( nptr first, data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->rlink;
}
```



7.5 이중 연결 리스트

(2) 제거

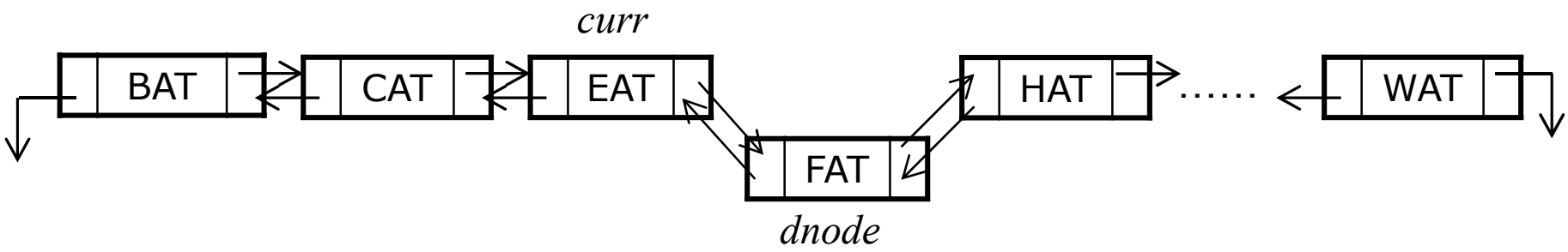
```
void Delete ( nptr first, data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->rlink;
}
```



7.5 이중 연결 리스트

(2) 제거

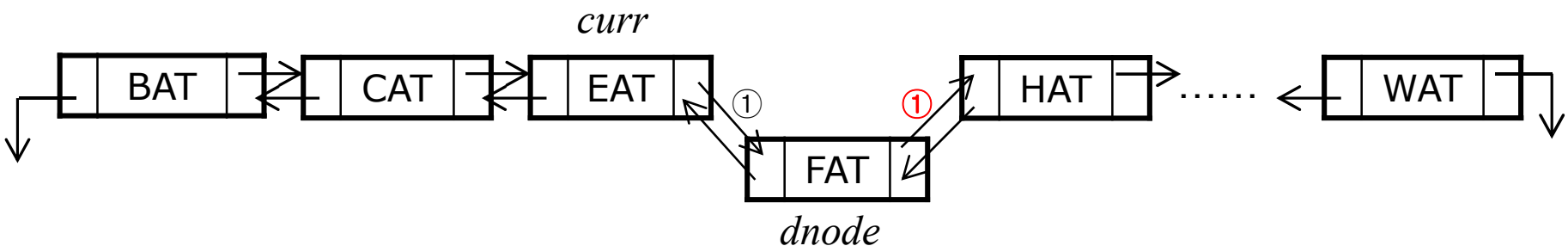
```
void Delete ( nptr first, data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->rlink;
}
```



7.5 이중 연결 리스트

(2) 제거

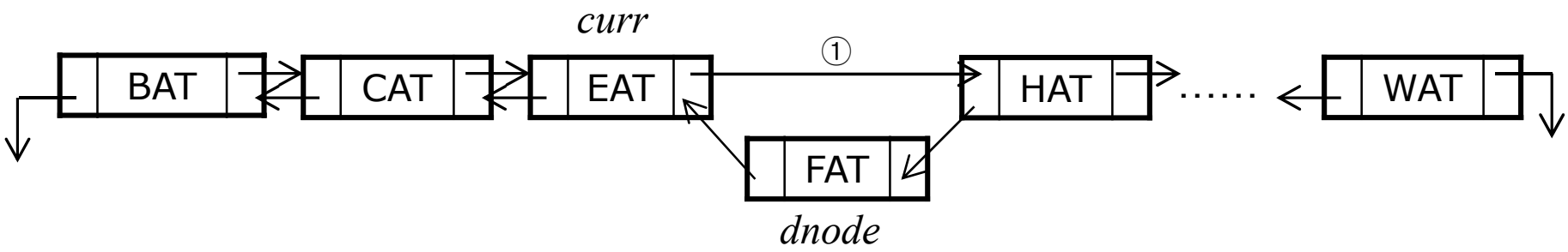
```
void Delete ( nptr first, data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->rlink;
    curr->rlink = dnode->rlink;
}
```



7.5 이중 연결 리스트

(2) 제거

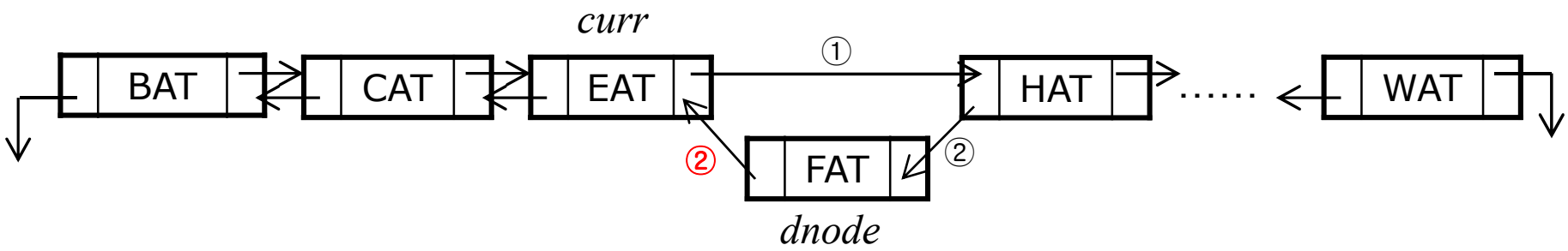
```
void Delete ( nptr first, data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->rlink;
    curr->rlink = dnode->rlink;
}
```



7.5 이중 연결 리스트

(2) 제거

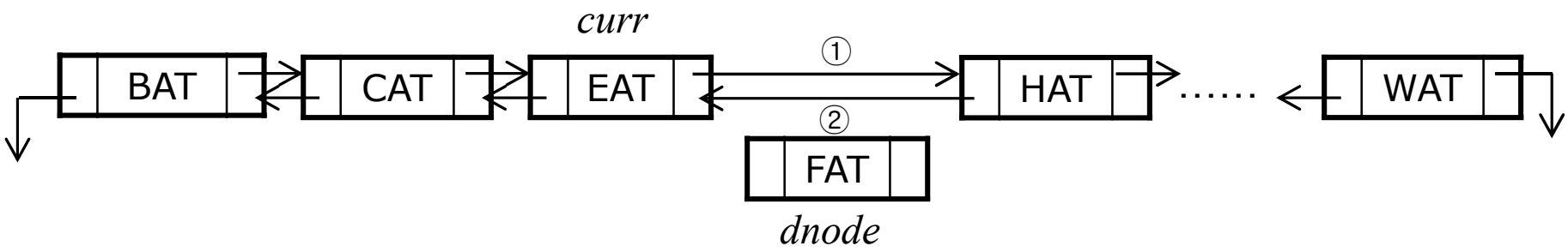
```
void Delete ( nptr first, data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->rlink;
    curr->rlink = dnode->rlink;
    curr->rlink->llink = dnode->llink;
}
```



7.5 이중 연결 리스트

(2) 제거

```
void Delete ( nptr first, data_type item )
{
//  2. 링크를 변경해서 노드를 삭제함
    nptr dnode = curr->rlink;
    curr->rlink = dnode->rlink;
    curr->rlink->llink = dnode->llink;
}
```



연결 리스트와 배열의 비교

- 배열

- 연결 리스트

7장에서 배운 내용

7.1 소개

7.2 리스트 구현?

7.3 연결 리스트 정의

7.4 연결 리스트 구현

7.5 이중 연결 리스트