데이터구조입문

리스트 II

김영민

2019. 5. 30

목차

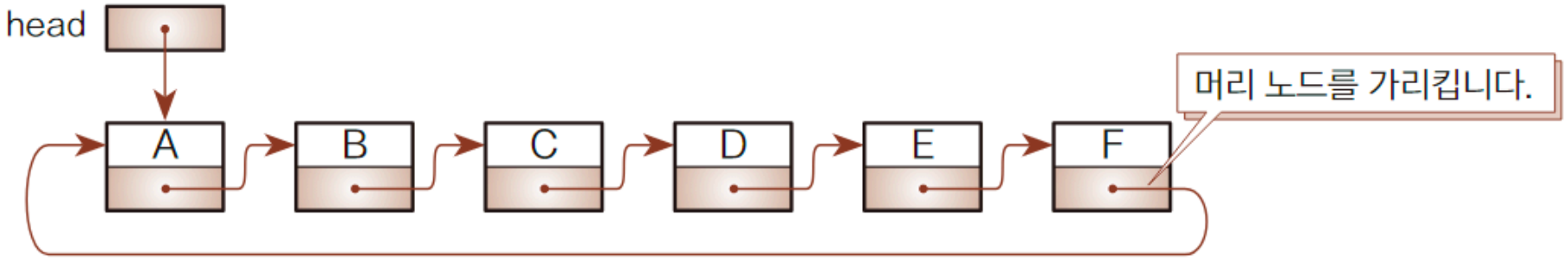
- 원형 이중 연결 리스트
- 트리

원형 이중 연결 리스트

원형 리스트

- 원형 리스트(circular list): 연결 리스트의 꼬리 노드가 머리 노드를 가리키는 경우

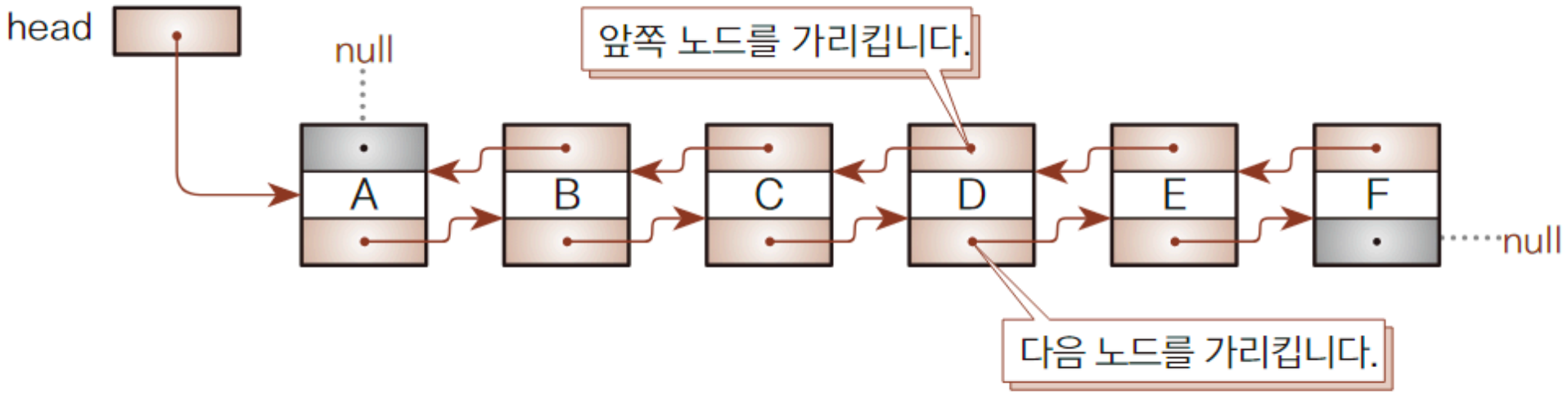
- 꼬리 노드의 다음 노드를 가리키는 포인터가 null이 아닌 머리 노드의 포인터 값



[그림 9-18] 원형 리스트

이중 연결 리스트

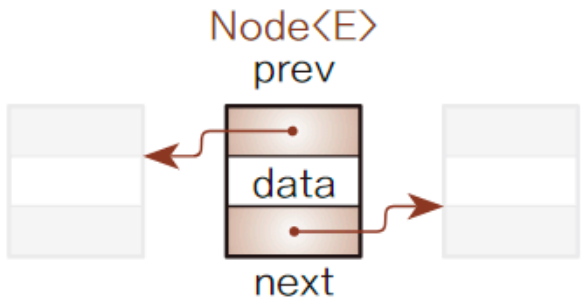
- 연결 리스트에서는 다음 노드는 찾기 쉽지만 앞쪽의 노드는 찾을 수 없다는 점이 가장 큰 단점
- 이중 연결 리스트(doubly linked list): 위의 단점을 개선한 자료구조. 각 노드에는 다음 노드에 대한 포인터와 앞쪽 노드에 대한 포인터가 모두 주어짐



[그림 9-19] 이중 연결 리스트

이중 연결 리스트 노드

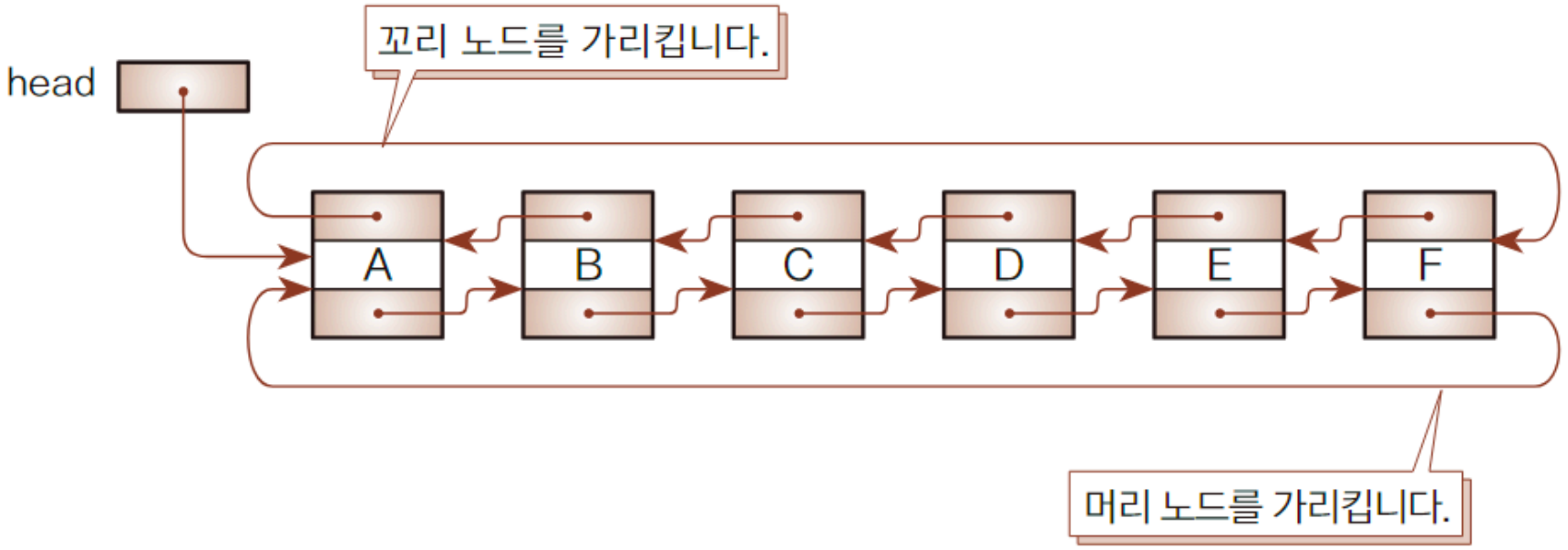

```
class Node<E> {
   E data;              // 데이터
   Node<E> prev;        // 머리 노드를 가리킴
   Node<E> next;        // 꼬리 노드를 가리킴
}
```



[그림 9-20] 이중 연결 리스트의 노드 구성

원형 이중 연결 리스트

- 원형 이중 연결 리스트(circular doubly linked list): 원형 리스트와 이중 리스트가 합쳐진 리스트



[그림 9-21] 원형 이중 연결 리스트

구현


```
01  package chap09;
02  import java.util.Comparator;
03  // 원형 이중 연결 리스트 클래스
04
05  public class DblLinkedList<E> {
06    // 노드
07    class Node<E> {
08      private E data;                  // 데이터
09      private Node<E> prev;            // 앞쪽 포인터 (앞쪽 노드에 대한 참조)
10      private Node<E> next;            // 뒤쪽 포인터 (다음 노드에 대한 참조)
11
12      // 생성자
13      Node() {
14        data = null;
15        prev = next = this;
16      }
17
18      // 생성자
19      Node(E obj, Node<E> prev, Node<E> next) {
20        data = obj;
21        this.prev = prev;
22        this.next = next;
23      }
24    }
25
26    private Node<E> head;              // 머리 노드 (더미 노드)
27    private Node<E> crnt;              // 선택 노드
28
29    // 생성자
30    public DblLinkedList() {
31      head = crnt = new Node<E>();     // 더미 노드를 생성
32    }
33
34    // 리스트가 비어 있는가?
35    public boolean isEmpty() {
36      return head.next == head;
37    }
```

노드 클래스

- Node<E>
  - 필드: data, prev, next
  - 생성자

1. Node()

데이터 data가 널이고 앞쪽 포인터와 뒤쪽 포인터가 모두 this인 노드를 생성합니다. 자기 자신의 노드가 앞쪽 노드이면서 동시에 다음 노드가 됩니다.

2. Node(E obj, Node<E> prev, Node<E> next)

데이터 data가 obj이고 앞쪽 포인터가 prev, 뒤쪽 포인터가 next인 노드를 생성합니다.

원형 이중 연결 리스트 클래스

- DblLinkedList<E>
  - 필드: head, crnt

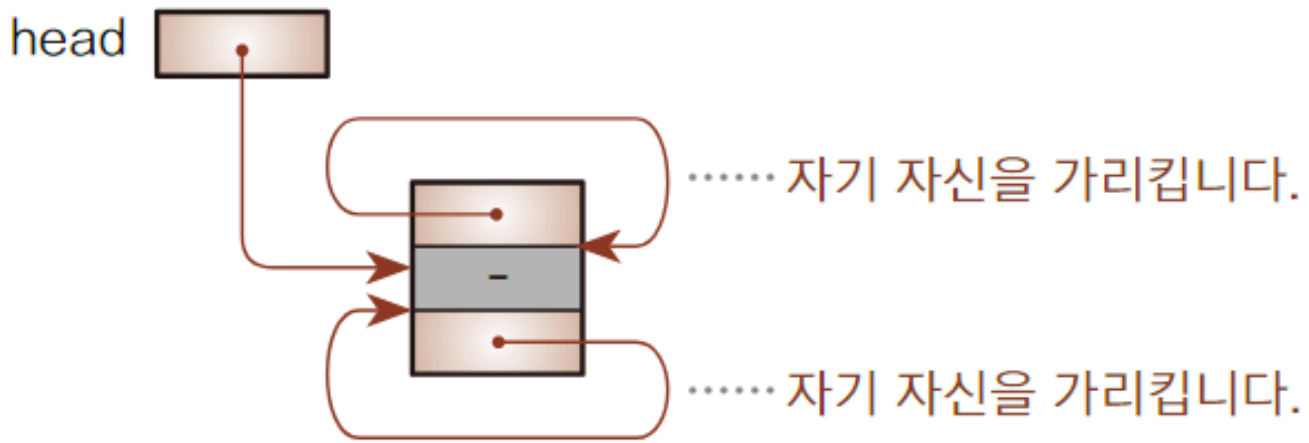
- head … 머리 노드를 가리킵니다.
- crnt … 선택 노드를 가리킵니다.

DblLinkedList 생성자

- 비어 있는 원형 이중 연결 리스트를 생성
- 데이터를 갖지 않는 노드 1개만 생성 : 노드의 삽입과 삭제 처리를 원활하게 하도록 리스트의 머리에 계속 존재하는 더미 노드
- 더미 노드 생성시 new Node<E>()에 의해 생성자를 호출.
- 더미 노드의 prev와 next는 자기 자신을 가리킴
- DblLinkedList 의 head와 crnt가 가리키는 곳은 이때 생성한 더미 노드

DblLinkedList 생성자

더미 노드만 있는 상태입니다.



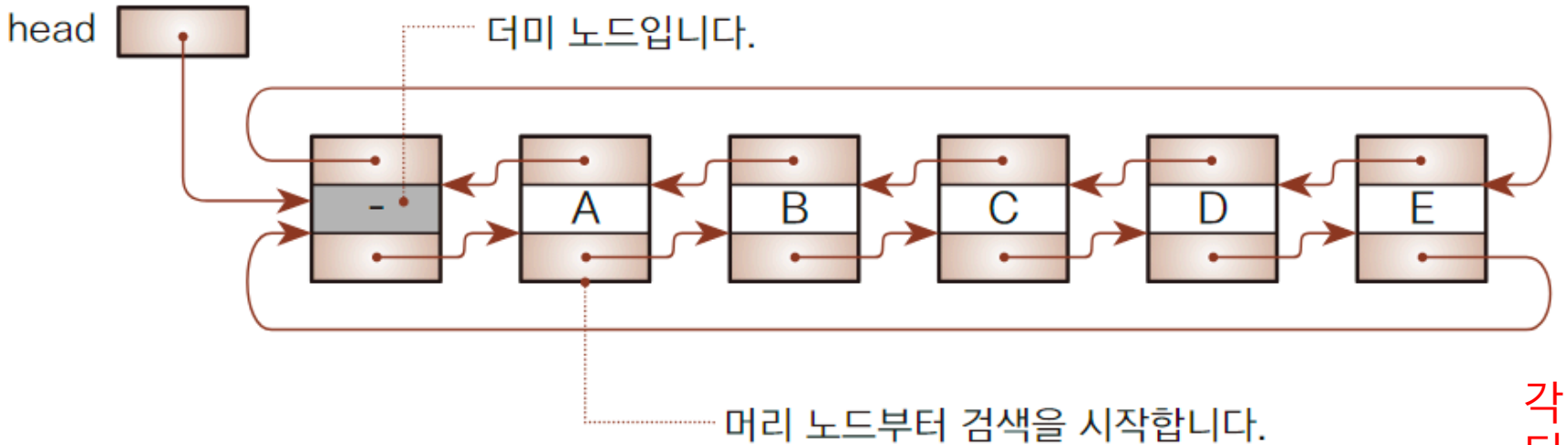
[그림 9-22] 빈 원형 이중 연결 리스트

isEmpty method

- 리스트가 비어 있는지(더미 노드만 있는지)를 조사하는 메서드
- 더미 노드의 뒤쪽 포인터 head.next가 더미 노드인 head를 가리킨다면 리스트는 비어 있는 것
- 리스트가 비어 있으면 true, 그렇지 않으면 false 반환

search method

- 노드를 검색하는 메서드
- LinkedList<E> seach 메서드와 거의 동일하게 머리 노드부터 뒤쪽 포인터를 차례로 찾아가며 스캔
- 다른 점은 검색을 시작하는 노드 → 더미 노드 바로 다음부터 시작



[그림 9-23] 노드 검색의 시작 위치

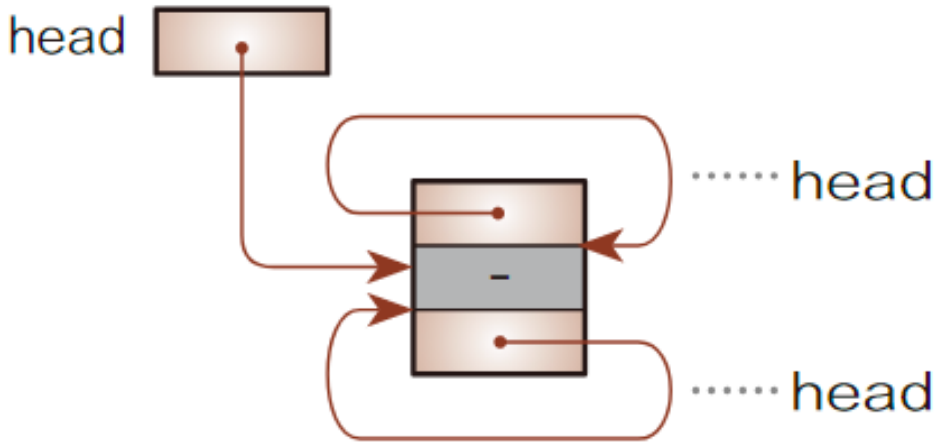
각 노드를 가리키는 포인터
더미 노드: head
머리 노드: head.next
꼬리 노드: head.prev

search method

- While문 스캔 과정에서 comparator c의 compare 메서드로 비교한 결과가 0 이면 검색 성공
- ptr이 가리키는 노드의 데이터 data를 반환
- 이 때 crnt는 찾은 노드 ptr을 가리키도록 설정
- 목적 노드를 찾지 못하고 스캔이 한바퀴 돌아 다시 머리 노드로 돌아올 때 while 문이 끝남
- 현재 ptr이 가리키고 있는 것이 노드 E라면 ptr=ptr.next를 수행한 후 ptr이 가리키는 곳이 더미 노드가 됨.
- 즉, ptr이 가리키는 곳이 head와 같은 때 스캔이 끝남

search method

- 빈 리스트인 경우
- Ptr에 대입하는 head.next는 더미 노드에 대한 참조. 즉, head와 같은 값
- While문이 성립하기 않으므로 null 값 반환



[그림 9-24] 빈 원형 이중 연결 리스트를 검색하는 경우

검색 구현


```
39      // 노드를 검색
40      public E search(E obj, Comparator<? super E> c) {
41        Node<E> ptr = head.next;          // 현재 스캔 중인 노드
42
43        while (ptr != head) {
44          if (c.compare(obj, ptr.data) == 0) {
45            crnt = ptr;
46            return ptr.data;              // 검색 성공
47          }
48          ptr = ptr.next;                 // 다음 노드로 선택
49        }
50        return null;                      // 검색 실패
51      }
```

Node pointer

더미 노드	head	e.next	d.next.next	a.prev	b.prev.prev
노드 A	a	head.next	e.next.next	b.prev	c.prev.prev
노드 B	b	a.next	head.next.next	c.prev	d.prev.prev
노드 C	c	b.next	a.next.next	d.prev	e.prev.prev
노드 D	d	c.next	b.next.next	e.prev	head.prev.prev
노드 E	e	d.next	c.next.next	head.prev	a.prev.prev

노드 위치 판단

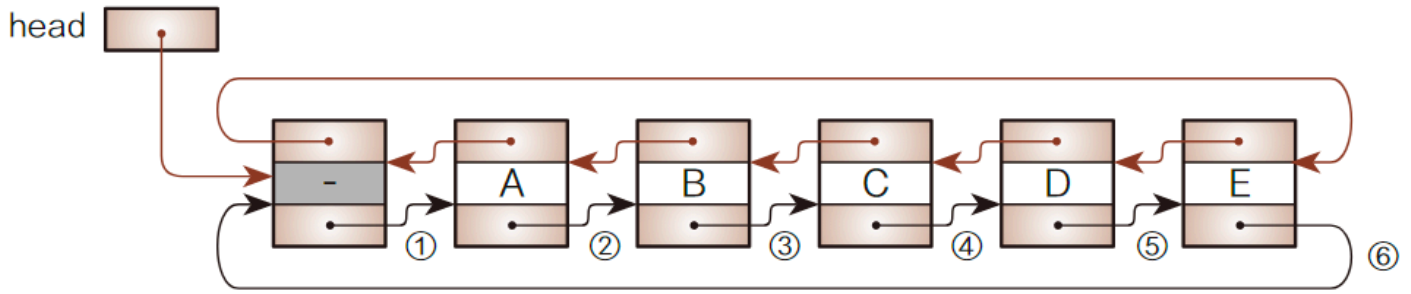

```
p.prev == head          // p가 가리키는 노드가 머리 노드인지 확인합니다(더미 노드는 제외).
p.prev.prev == head     // p가 가리키는 노드가 머리에서부터 2번째 노드인지 확인합니다(더미
                        노드는 제외).
p.next == head          // p가 가리키는 노드가 꼬리 노드인지 확인합니다.
p.next.next == head     // p가 가리키는 노드가 꼬리에서부터 2번째 노드인지 확인합니다.
```

출력 method

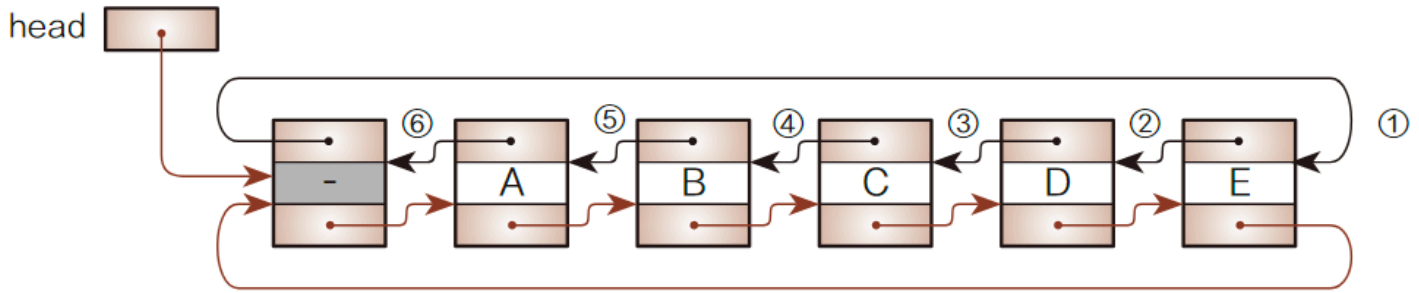
- printCurrentNode method
  - 선택 노드를 출력하는 메서드로, 선택 노드의 crnt.data 출력
- dump method
  - 리스트의 모든 노드를 머리부터 꼬리까지 출력하는 메서드
- dumpReverse method
  - 리스트의 모든 노드를 꼬리부터 거꾸로 출력하는 메서드
  - head.prev부터 스캔을 시작하여 앞쪽 포인터를 찾아가며 각 노드 데이터 출력

노드 스캔 과정

a 머리부터 모든 노드를 스캔합니다.



b 꼬리부터 모든 노드를 스캔합니다.



[그림 9-25] 모든 노드를 스캔하는 과정

출력 구현


```
53    // 선택 노드를 출력
54    public void printCurrentNode() {
55       if (isEmpty())
56          System.out.println("선택 노드가 없습니다.");
57       else
58          System.out.println(crnt.data);
59    }
60
61    // 모든 노드를 출력
62    public void dump() {
63       Node<E> ptr = head.next;        // 더미 노드의 다음 노드
64
65       while (ptr != head) {
66          System.out.println(ptr.data);
67          ptr = ptr.next;
68       }
69    }
```

```
71    // 모든 노드를 거꾸로 출력
72    public void dumpReverse() {
73       Node<E> ptr = head.prev;        // 더미 노드의 앞쪽 노드
74
75       while (ptr != head) {
76          System.out.println(ptr.data);
77          ptr = ptr.prev;
78       }
79    }
```

이동 method

- next method
  - 선택 노드를 하나 뒤쪽의 노드로 옮겨 진행하는 메서드
  - 리스트가 비어 있지 않고 선택 노드 다음 노드가 있을 때 이동

- prev method
  - 선택 노드를 하나 앞쪽의 노드로 이동하는 메서드
  - 리스트가 비어 있지 않고 선택 노드 앞쪽 노드가 있을 때 이동
  - 선택 노드가 이동하면 true, 그렇지 않으면 false 반환

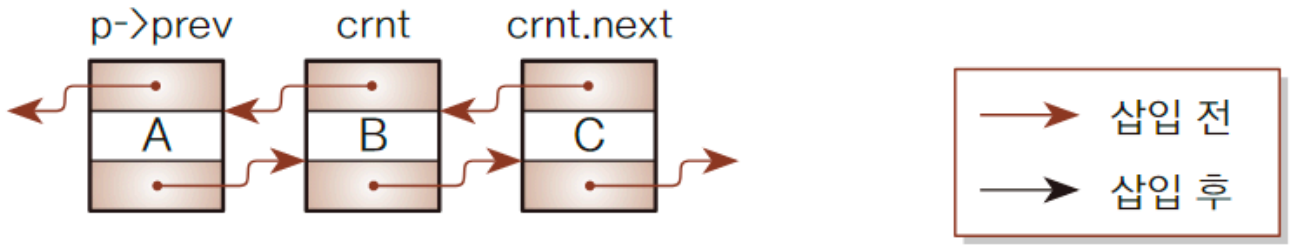
이동 구현


```
81      // 선택 노드를 하나 뒤쪽으로 이동
82      public boolean next() {
83        if (isEmpty() || crnt.next == head)
84          return false;                   // 이동할 수 없음
85        crnt = crnt.next;
86        return true;
87      }
88
89      // 선택 노드를 하나 앞쪽으로 이동
90      public boolean prev() {
91        if (isEmpty() || crnt.prev == head)
92          return false;                   // 이동할 수 없음
93        crnt = crnt.prev;
94        return true;
95      }
```

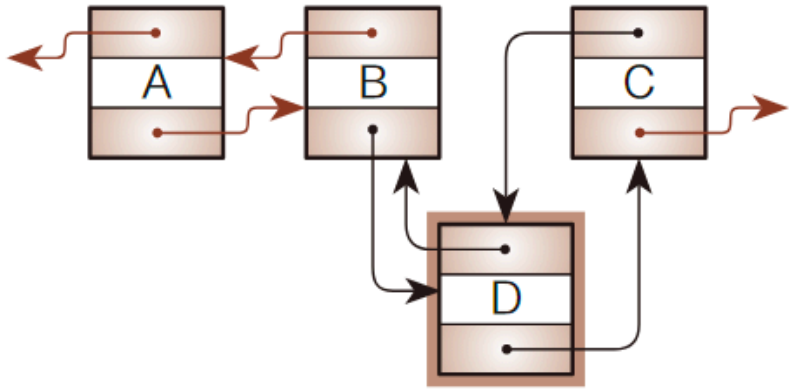
add method

- 선택 노드의 바로 뒤에 노드를 삽입하는 메서드

a 삽입 전



b 삽입 후



[그림 9-26] 원형 이중 연결 리스트에 노드를 삽입하는 과정

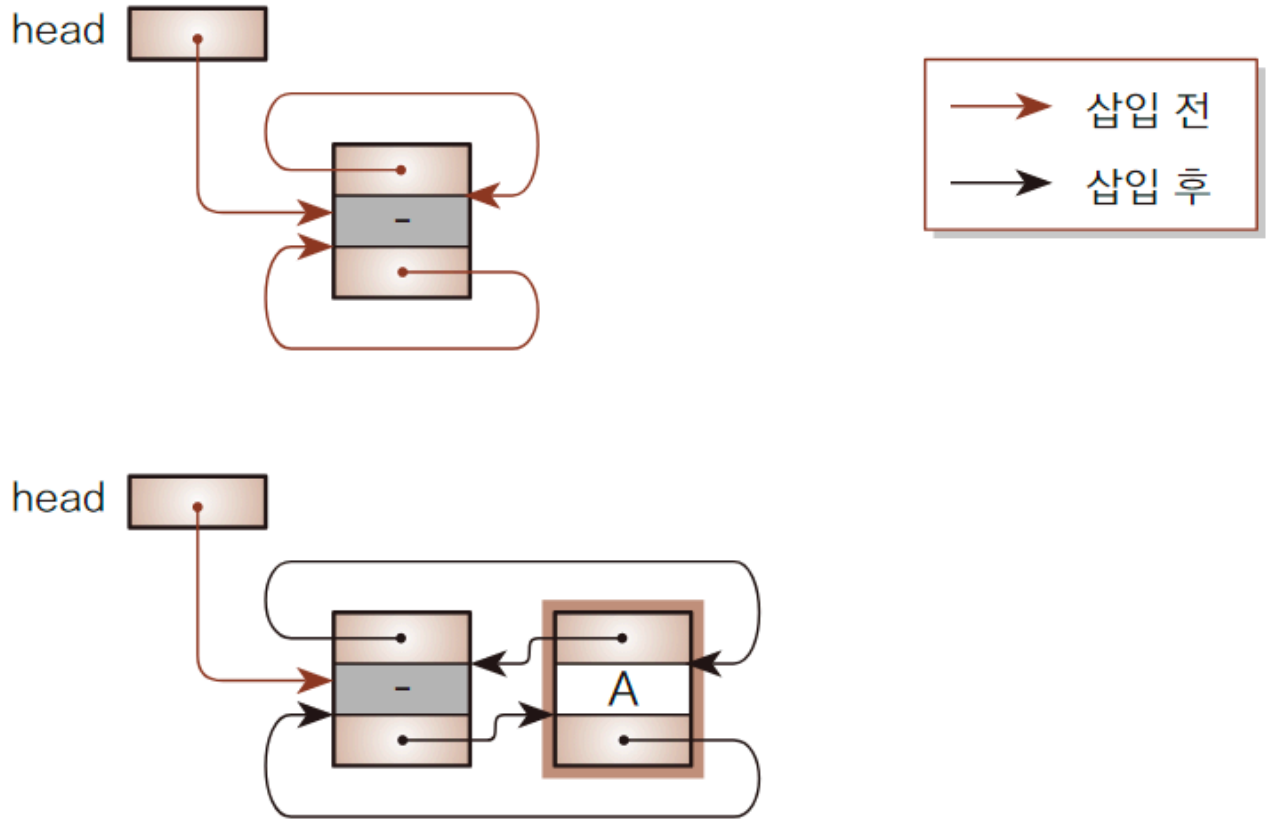
add method

1. 새로 삽입할 노드를 new Node<E>(obj, crnt, crnt.next)에 의해 생성합니다. 생성한 노드의 데이터는 obj, 앞쪽 포인터가 가리키는 곳은 노드 B, 뒤쪽 포인터가 가리키는 곳은 노드 C가 됩니다.
2. 노드 B의 뒤쪽 포인터 crnt.next와 노드 C의 앞쪽 포인터 crnt.next.prev 둘 다 새로 삽입한 노드를 가리키도록 업데이트합니다.
3. 선택 노드가 삽입한 노드를 가리키도록 업데이트합니다.

- 더미 노드가 있으므로 비어있는 리스트에 삽입하거나 리스트의 머리에 삽입하는 것을 특별히 따로 다룰 필요는 없음

add method

a 삽입 전



[그림 9-27] 빈 원형 이중 연결 리스트에 노드를 삽입하는 과정

addFirst, addLast

- addFirst method
  - 리스트의 머리에 노드를 삽입하는 메서드
  - 더미 노드 바로 뒤에 노드를 삽입하므로 선택 노드 crnt가 가리키는 곳을 head로 업데이트 후 add 호출

- addLast method
  - 리스트의 꼬리에 노드를 삽입하는 메서드
  - 꼬리 노드 바로 뒤에 있는 더미 노드 바로 앞에 노드를 삽입하므로 선택 노드 crnt가 가리키는 곳을 head.prev로 업데이한 후에 add 호출

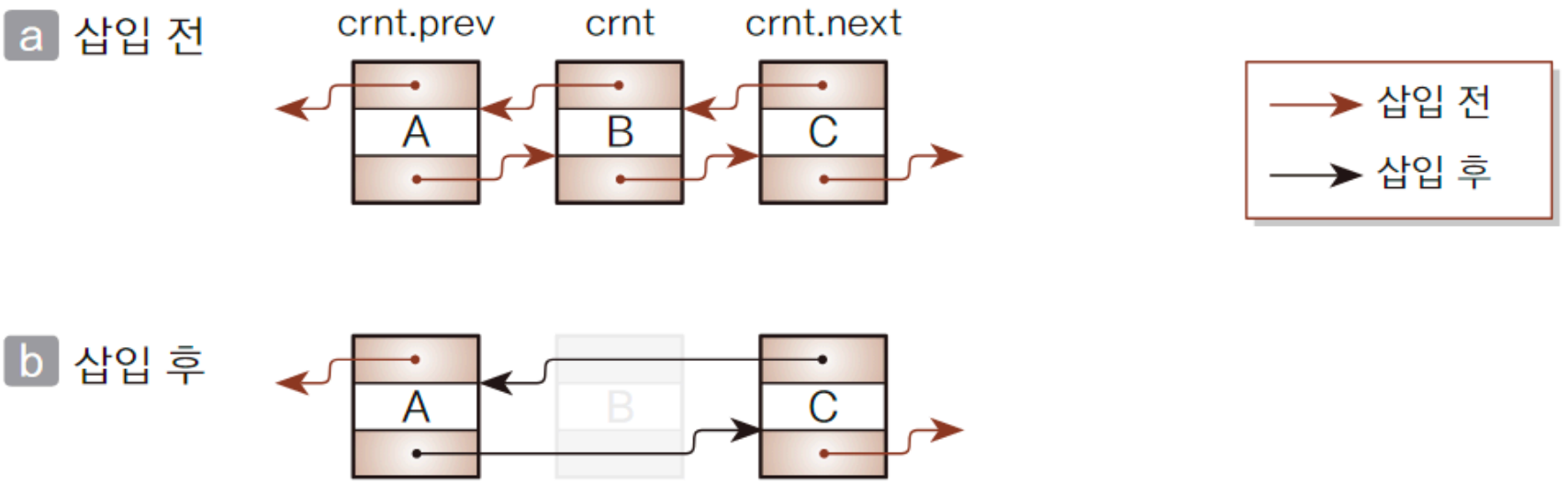
add 구현


```
97    // 선택 노드의 바로 뒤에 노드를 삽입
98    public void add(E obj) {
99       Node<E> node = new Node<E>(obj, crnt, crnt.next);   1
100      crnt.next = crnt.next.prev = node;                  2
101      crnt = node;                                        3
102   }
103
```

```
104   // 머리에 노드를 삽입
105   public void addFirst(E obj) {
106      crnt = head;              // 더미 노드 head의 바로 뒤에 삽입
107      add(obj);
108   }
109
110   // 꼬리에 노드를 삽입
111   public void addLast(E obj) {
112      crnt = head.prev;         // 꼬리 노드 head.prev의 바로 뒤에 삽입
113      add(obj);
114   }
```

removeCurrentNode method

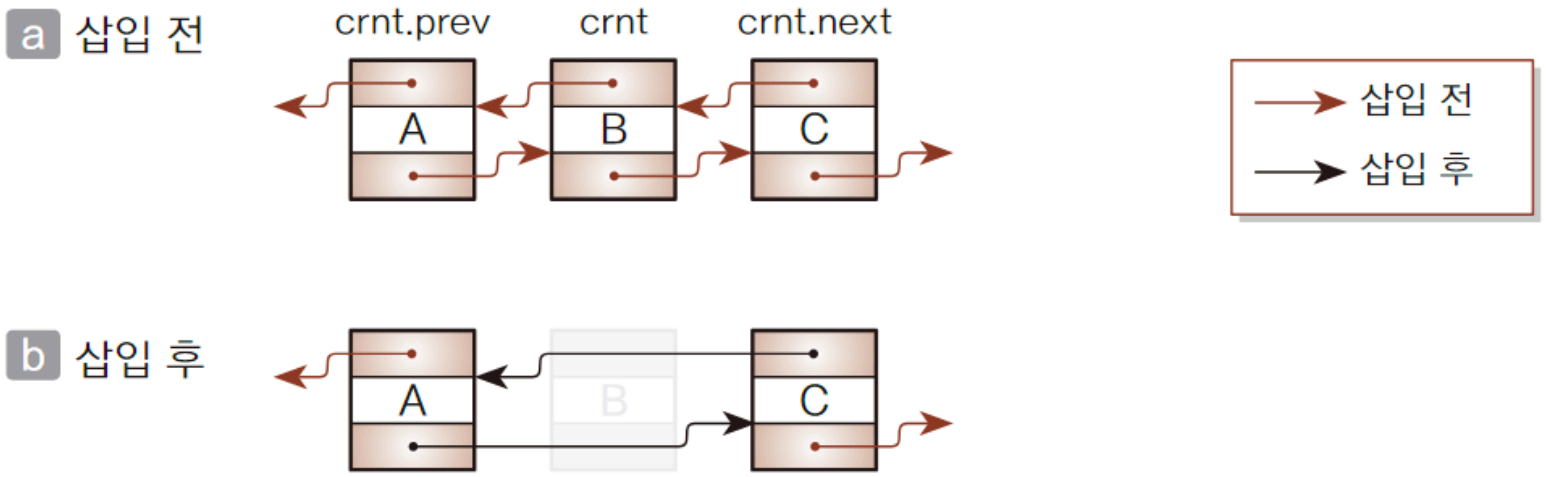
- 선택 노드를 삭제하는 메서드
- 더미 노드는 삭제할 수 없으므로 먼저 리스트가 비어있는 지 확인



[그림 9-28] 원형 이중 연결 리스트에서 노드를 삭제하는 과정

removeCurrentNode method

1. 노드 A의 뒤쪽 포인터 crnt.prev.next가 가리키는 곳이 노드 C crnt.next가 되도록 업데이트합니다.
2. 노드 C의 앞쪽 포인터 crnt.next.prev가 가리키는 곳이 노드 A crnt.prev가 되도록 업데이트합니다. 노드 B는 어디에서도 가리키는 곳이 없게 되어 삭제 처리를 끝냅니다.
3. 선택 노드가 삭제한 노드의 앞쪽 노드 A를 가리키도록 업데이트합니다.



[그림 9-28] 원형 이중 연결 리스트에서 노드를 삭제하는 과정

삭제 method

- remove method
  - 임의의 노드를 삭제하는 메서드
  - P가 참조하는 노드를 삭제하는 메서드
  - 리스트가 비어있지 않고 인수가 가리키는 노드 p가 있을 때에만 삭제
  - While 문으로 모든 노드를 스캔하다가 노드 p를 찾으면 crnt가 가리키는 곳을 p로 업데이트 하고 removeCurrentNode 메서드를 호출

- removeFirst method
  - 머리 노드를 삭제하는 메서드
  - 선택 노드 crnt가 가리키는 곳을 머리 노드인 head.next로 업데이트 하고 removeCurrentNode 메서드를 호출

삭제 method

- removeLast
  - 꼬리 노드를 삭제하는 메서드
  - 선택 노드 crnt가 가리키는 곳을 꼬리 노드 head.prev로 업데이트 하고 removeCurrentNode 메서드를 호출

- clear
  - 더미 노드를 제외한 모든 노드를 삭제하는 메서드
  - 리스트가 빌 때까지 removeFirst에 의해 머리 노드의 삭제를 반복
  - 그 결과 선택 노드의 포인터 crnt가 가리키는 곳은 더미 노드 head로 업데이트됨

```
116   // 선택 노드를 삭제
117   public void removeCurrentNode() {
118     if (!isEmpty()) {
119       crnt.prev.next = crnt.next;  ①
120       crnt.next.prev = crnt.prev;  ②
121       crnt = crnt.prev;            ③
122       if (crnt == head) crnt = head.next;
123     }
124   }
125
126   // 노드 p를 삭제
127   public void remove(Node p) {
128     Node<E> ptr = head.next;
129
130     while (ptr != head) {
131       if (ptr == p) {            // p를 찾음
132         crnt = p;
133         removeCurrentNode();
134         break;
135       }
136       ptr = ptr.next;
137     }
138   }
```

```
140   // 머리 노드를 삭제
141   public void removeFirst() {
142     crnt = head.next;            // 머리 노드 head.next를 삭제
143     removeCurrentNode();
144   }
145
146   // 꼬리 노드를 삭제
147   public void removeLast() {
148     crnt = head.prev;            // 꼬리 노드 head.prev를 삭제
149     removeCurrentNode();
150   }
151
152   // 모든 노드를 삭제
153   public void clear() {
154     while (!isEmpty())           // 텅 빌 때까지
155       removeFirst();             // 머리 노드를 삭제
156   }
157 }
```

원형 이중 연결 리스트를 사용한 프로그램 구현


```
01  package chap09;
02  import java.util.Scanner;
03  import java.util.Comparator;
04  // 원형 이중 연결 리스트 클래스 DblLinkedList<E>의 사용 예
05
06  class DblLinkedListTester {
07    static Scanner stdIn = new Scanner(System.in);
08
09    // 데이터 (회원번호 + 이름)
10    static class Data {
11      static final int NO   = 1;   // 번호를 입력 받습니까?
12      static final int NAME = 2;   // 이름을 입력 받습니까?
13
14      private Integer no;          // 회원번호
15      private String  name;        // 이름
16
17      // 문자열을 반환합니다.
18      public String toString() {
19        return "(" + no + ") " + name;
20      }
21
22      // 데이터를 입력합니다.
23      void scanData(String guide, int sw) {
24        System.out.println(guide + "할 데이터를 입력하세요.");
25
26        if ((sw & NO) == NO) {
27          System.out.print("번호 : ");
28          no = stdIn.nextInt();
29        }
30        if ((sw & NAME) == NAME) {
31          System.out.print("이름 : ");
32          name = stdIn.next();
33        }
34      }
```

구현


```
36      // 회원번호로 순서를 매기는 comparator
37      public static final Comparator<Data> NO_ORDER =
38                              new NoOrderComparator();
39
40      private static class NoOrderComparator implements Comparator<Data> {
41        public int compare(Data d1, Data d2) {
42          return (d1.no > d2.no) ? 1 : (d1.no < d2.no) ? -1 : 0;
43        }
44      }
45
46      // 이름으로 순서를 매기는 comparator
47      public static final Comparator<Data> NAME_ORDER =
48                              new NameOrderComparator();
49
50      private static class NameOrderComparator implements Comparator<Data> {
51        public int compare(Data d1, Data d2) {
52          return d1.name.compareTo(d2.name);
53        }
54      }
55    }
```

구현


```
57    // 메뉴 열거형
58    enum Menu {
59       ADD_FIRST(  "머리에 노드를 삽입"),
60       ADD_LAST(   "꼬리에 노드를 삽입"),
61       ADD(        "선택 노드의 바로 뒤에 삽입"),
62       RMV_FIRST(  "머리 노드를 삭제"),
63       RMV_LAST(   "꼬리 노드를 삭제"),
64       RMV_CRNT(   "선택 노드를 삭제"),
65       CLEAR(      "모든 노드를 삭제"),
66       SEARCH_NO(  "번호로 검색"),
67       SEARCH_NAME("이름으로 검색"),
68       NEXT(       "선택 노드를 뒤쪽으로"),
69       PREV(       "선택 노드를 앞쪽으로"),
70       PRINT_CRNT( "선택 노드를 출력"),
71       DUMP(       "모든 노드를 출력"),
72       TERMINATE(  "종료");
73
74       private final String message;       // 출력할 문자열
75
76       static Menu MenuAt(int idx) {       // 서수가 idx인 열거를 반환
77          for (Menu m : Menu.values())
78             if (m.ordinal() == idx)
79                return m;
80          return null;
81       }
82
83       Menu(String string) {               // 생성자
84          message = string;
85       }
86
87       String getMessage() {               // 출력할 문자열을 반환
88          return message;
89       }
90    }
```

구현

```
92    // 메뉴 선택
93    static Menu SelectMenu() {
94       int key;
95       do {
96          for (Menu m : Menu.values()) {
97             System.out.printf("(%d) %s  ", m.ordinal(), m.getMessage());
98             if ((m.ordinal() % 3) == 2 &&
99                   m.ordinal() != Menu.TERMINATE.ordinal())
100               System.out.println();
101         }
102         System.out.print(" : ");

103         key = stdIn.nextInt();
104      } while (key < Menu.ADD_FIRST.ordinal() ||
105                              key > Menu.TERMINATE.ordinal());
106      return Menu.MenuAt(key);
107   }
```

구현


```
109   public static void main(String[] args) {
110     Menu menu;                          // 메뉴
111     Data data;                          // 추가용 데이터 참조
112     Data ptr;                           // 검색용 데이터 참조
113     Data temp = new Data();             // 입력용 데이터
114
115     DblLinkedList<Data> list = new DblLinkedList<Data>(); // 리스트를
116
117     do {
118       switch (menu = SelectMenu()) {
119
120         case ADD_FIRST :                // 머리에 노드를 삽입
121             data = new Data();
122             data.scanData("머리에 삽입", Data.NO | Data.NAME);
123             list.addFirst(data);
124             break;
125
126         case ADD_LAST :                 // 꼬리에 노드를 삽입
127             data = new Data();
128             data.scanData("꼬리에 삽입", Data.NO | Data.NAME);
129             list.addLast(data);
130             break;
```

```
132         case ADD :                      // 선택 노드의 바로 뒤에 노드를 삽입
133             data = new Data();
134             data.scanData("선택 노드의 바로 뒤에 삽입", Data.NO | Data.NAME);
135             list.add(data);
136             break;
137
138         case RMV_FIRST :                // 머리 노드를 삭제
139             list.removeFirst();
140             break;
141
142         case RMV_LAST :                 // 꼬리 노드를 삭제
143             list.removeLast();
144             break;
145
146         case RMV_CRNT :                 // 선택 노드를 삭제
147             list.removeCurrentNode();
148             break;
149
```

```
150         case SEARCH_NO :                // 회원번호로 검색
151             temp.scanData("검색", Data.NO);
152             ptr = list.search(temp, Data.NO_ORDER);
153             if (ptr == null)
154               System.out.println("그 번호의 데이터가 없습니다.");
155             else
156               System.out.println("검색 성공 : " + ptr);
157             break;
158
159         case SEARCH_NAME :              // 이름으로 검색
160             temp.scanData("검색", Data.NAME);
161             ptr = list.search(temp, Data.NAME_ORDER);
162             if (ptr == null)
163               System.out.println("그 이름의 데이터가 없습니다.");
164             else
165               System.out.println("검색 성공 : " + ptr);
166             break;
```

```
168         case NEXT :                     // 선택 노드를 뒤쪽으로 진행
169             list.next();
170             break;
171
172         case PREV :                     // 선택 노드를 앞쪽으로 진행
173             list.prev();
174             break;
175
176         case PRINT_CRNT :               // 선택 노드의 데이터를 출력
177             list.printCurrentNode();
178             break;
179
180         case DUMP :                     // 모든 데이터를 리스트 순서로
181             list.dump();
182             break;
184         case CLEAR :                    // 모든 노드를 삭제
185             list.clear();
186             break;
187       }
188     } while (menu != Menu.TERMINATE);
189   }
190 }
```

실행 결과

실행 결과

(0) 머리에 노드를 삽입 (1) 꼬리에 노드를 삽입 (2) 선택 노드의 바로 뒤에 삽입
(3) 머리 노드를 삭제 (4) 꼬리 노드를 삭제 (5) 선택 노드를 삭제
(6) 모든 노드를 삭제 (7) 번호로 검색 (8) 이름으로 검색
(9) 선택 노드를 뒤쪽으로 (10) 선택 노드를 앞쪽으로 (11) 선택 노드를 출력
(12) 모든 노드를 출력 (13) 종료 : 0
머리에 삽입할 데이터를 입력하세요.
번호 : 1 ..... {1, 모모}를 머리에 삽입
이름 : 모모

(0) 머리에 노드를 삽입 (1) 꼬리에 노드를 삽입 (2) 선택 노드의 바로 뒤에 삽입
(3) 머리 노드를 삭제 (4) 꼬리 노드를 삭제 (5) 선택 노드를 삭제
(6) 모든 노드를 삭제 (7) 번호로 검색 (8) 이름으로 검색
(9) 선택 노드를 뒤쪽으로 (10) 선택 노드를 앞쪽으로 (11) 선택 노드를 출력
(12) 모든 노드를 출력 (13) 종료 : 1
꼬리에 삽입할 데이터를 입력하세요.
번호 : 5 ..... {5, 나연}을 꼬리에 삽입
이름 : 나연

(0) 머리에 노드를 삽입 (1) 꼬리에 노드를 삽입 (2) 선택 노드의 바로 뒤에 삽입
(3) 머리 노드를 삭제 (4) 꼬리 노드를 삭제 (5) 선택 노드를 삭제
(6) 모든 노드를 삭제 (7) 번호로 검색 (8) 이름으로 검색
(9) 선택 노드를 뒤쪽으로 (10) 선택 노드를 앞쪽으로 (11) 선택 노드를 출력
(12) 모든 노드를 출력 (13) 종료 : 0
머리에 삽입할 데이터를 입력하세요.
번호 : 10 ..... {10, 정연}을 머리에 삽입
이름 : 정연

(0) 머리에 노드를 삽입 (1) 꼬리에 노드를 삽입 (2) 선택 노드의 바로 뒤에 삽입
(3) 머리 노드를 삭제 (4) 꼬리 노드를 삭제 (5) 선택 노드를 삭제
(6) 모든 노드를 삭제 (7) 번호로 검색 (8) 이름으로 검색
(9) 선택 노드를 뒤쪽으로 (10) 선택 노드를 앞쪽으로 (11) 선택 노드를 출력
(12) 모든 노드를 출력 (13) 종료 : 1
꼬리에 삽입할 데이터를 입력하세요.
번호 : 12 ..... {12, 사나}를 꼬리에 삽입
이름 : 사나

(0) 머리에 노드를 삽입 (1) 꼬리에 노드를 삽입 (2) 선택 노드의 바로 뒤에 삽입
(3) 머리 노드를 삭제 (4) 꼬리 노드를 삭제 (5) 선택 노드를 삭제
(6) 모든 노드를 삭제 (7) 번호로 검색 (8) 이름으로 검색
(9) 선택 노드를 뒤쪽으로 (10) 선택 노드를 앞쪽으로 (11) 선택 노드를 출력
(12) 모든 노드를 출력 (13) 종료 : 0
머리에 삽입할 데이터를 입력하세요.
번호 : 14 ..... {14, 지효}를 머리에 삽입
이름 : 지효

(0) 머리에 노드를 삽입 (1) 꼬리에 노드를 삽입 (2) 선택 노드의 바로 뒤에 삽입
(3) 머리 노드를 삭제 (4) 꼬리 노드를 삭제 (5) 선택 노드를 삭제
(6) 모든 노드를 삭제 (7) 번호로 검색 (8) 이름으로 검색
(9) 선택 노드를 뒤쪽으로 (10) 선택 노드를 앞쪽으로 (11) 선택 노드를 출력
(12) 모든 노드를 출력 (13) 종료 : 4 ..... 꼬리의 {12, 사나}를 삭제

(0) 머리에 노드를 삽입 (1) 꼬리에 노드를 삽입 (2) 선택 노드의 바로 뒤에 삽입
(3) 머리 노드를 삭제 (4) 꼬리 노드를 삭제 (5) 선택 노드를 삭제
(6) 모든 노드를 삭제 (7) 번호로 검색 (8) 이름으로 검색
(9) 선택 노드를 뒤쪽으로 (10) 선택 노드를 앞쪽으로 (11) 선택 노드를 출력
(12) 모든 노드를 출력 (13) 종료 : 8
검색할 데이터를 입력하세요.
이름 : 사나 ..... {사나} 검색 실패
그 이름의 데이터가 없습니다.

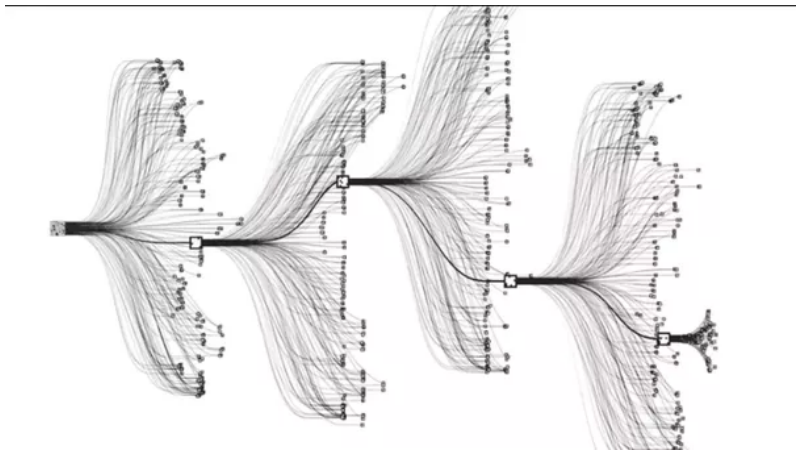
(0) 머리에 노드를 삽입 (1) 꼬리에 노드를 삽입 (2) 선택 노드의 바로 뒤에 삽입
(3) 머리 노드를 삭제 (4) 꼬리 노드를 삭제 (5) 선택 노드를 삭제
(6) 모든 노드를 삭제 (7) 번호로 검색 (8) 이름으로 검색
(9) 선택 노드를 뒤쪽으로 (10) 선택 노드를 앞쪽으로 (11) 선택 노드를 출력
(12) 모든 노드를 출력 (13) 종료 : 7
검색할 데이터를 입력하세요.
번호 : 10 ..... {10} 검색 성공
검색 성공 : (10) 정연

(0) 머리에 노드를 삽입 (1) 꼬리에 노드를 삽입 (2) 선택 노드의 바로 뒤에 삽입
(3) 머리 노드를 삭제 (4) 꼬리 노드를 삭제 (5) 선택 노드를 삭제
(6) 모든 노드를 삭제 (7) 번호로 검색 (8) 이름으로 검색
(9) 선택 노드를 뒤쪽으로 (10) 선택 노드를 앞쪽으로 (11) 선택 노드를 출력
(12) 모든 노드를 출력 (13) 종료 : 11
(10) 정연 ..... 선택한 노드는 {10, 정연}

트리

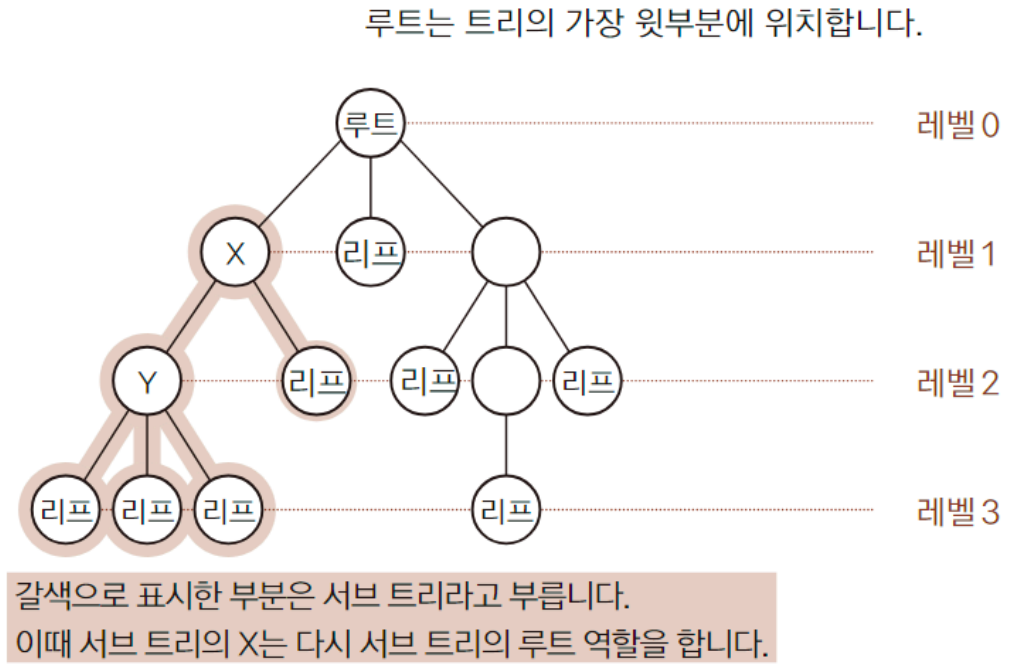
트리

- 나무
- 데이터 사이의 계층 관계를 나타내는 자료 구조
- 트리는 부모-자식 관계의 노드들로 이루어진다.
- 응용분야:
  - 계층적인 조직 표현
  - 파일 시스템
  - 인공지능에서의 의사결정나무



트리 관련 용어

- 노드(node) : 트리의 구성 요소
- 가지(edge) : 노드 사이를 연결하는 선



[그림 10-1] 트리

트리 관련 용어

- 루트(root) : 트리의 가장 윗부분에 위치하는 노드. 하나의 트리에는 하나의 루트가 있음
- 리프(leaf) : 트리의 가장 아랫부분에 위치하는 노드 (= terminal node = external node)
- 안쪽 노드(internal node) : 루트를 포함하여 리프를 제외한 노드
- 자식(child) : 현재 노드로부터 가지로 연결된 아래쪽 놈. 노드는 자식을 여러 개 가질 수 있음.

트리 관련 용어

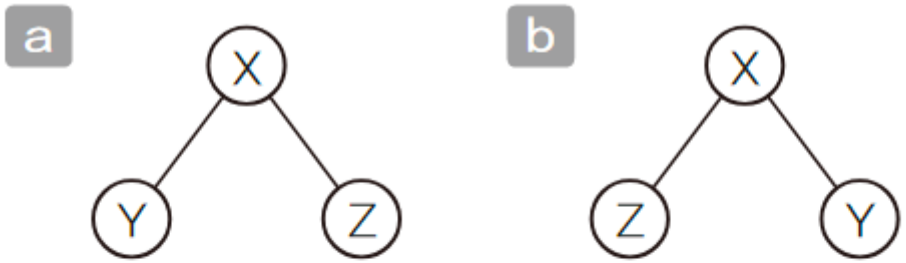
- 부모(parent) : 현재 노드에서 가지로 연결된 위쪽 노드. 노드는 1개의 부모 노드를 가짐
- 형제(sibling) : 같은 부모를 가지는 노드
- 조상(ancestor) : 현재 노드에서 가지로 연결된 위쪽 노드 모두
- 자손(descendant) : 현재 노드에서 가지로 연결된 아래쪽 노드 모두
- 레벨(level) : 루트로부터 얼마나 떨어져 있는지에 대한 값. 루트의 레벨은 0

트리 관련 용어

- 차수(degree) : 노드가 갖는 자식의 수. 예에서 X의 차수는 2, Y의 차수는 3. 모든 노드의 차수가 n 이하인 트리를 n진 트리라고 함. 예시의 트리는 3진 트리.
- 높이(height) : 루트로부터 가장 멀리 떨어진 리프까지의 거리 = 리프 레벨의 최대값
- 서브 트리(subtree) : 트리 내부에서 특정 노드를 선택할 시 그것을 루트로 하고 그 자손들로 이루어진 트리
- 널 트리(null tree) : 노드나 가지가 없는 트리

순서 트리와 무순서 트리

- 순서 트리(ordered tree) : 형제 노드의 순서를 따지는 경우
- 무순서 트리(unordered tree) : 형제 노드의 순서를 따지지 않는 경우

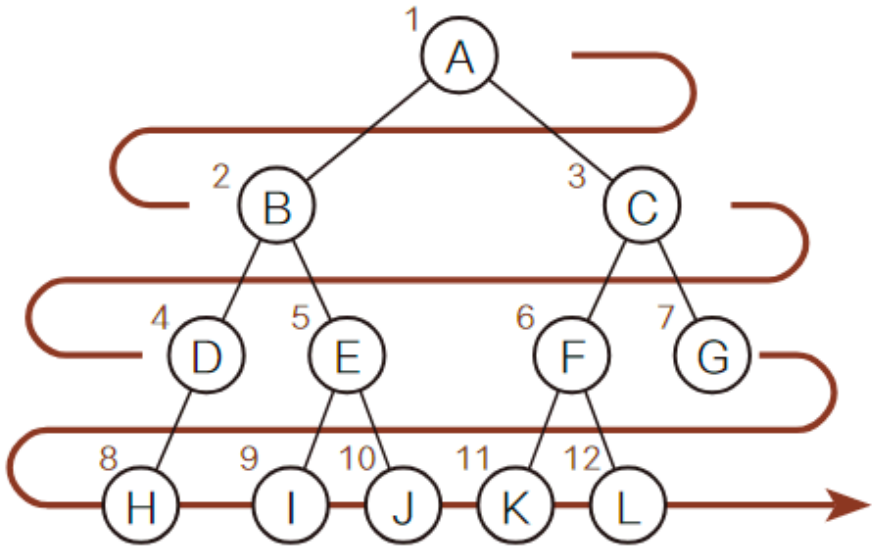


두 트리는 다른 순서 트리이면서
같은 무순서 트리라고 할 수 있습니다.

[그림 10-2] 순서 트리와 무순서 트리

순서 트리 탐색

- 너비 우선 탐색(breadth-first Search)
  - 낮은 레벨에서 시작해 왼쪽에서 오른쪽 방향으로 검색하고 한 레벨에서의 검색이 끝나면 다음 레벨로 내려가는 탐색

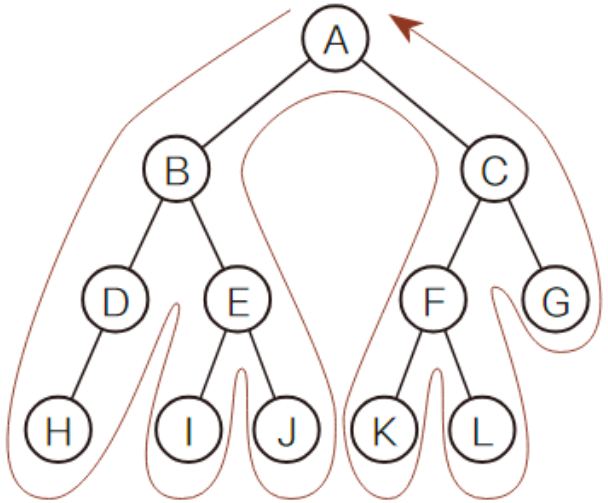


[그림 10-3] 너비 우선 탐색

A → B → C → D → E → F → G → H → I → J → K → L

순서 트리 탐색

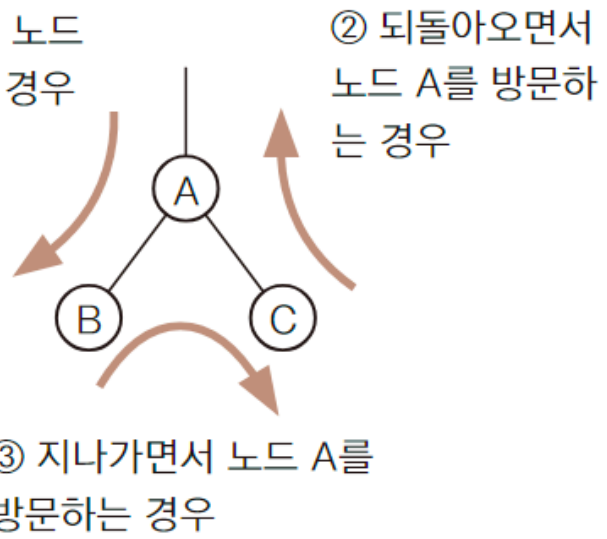
- 깊이 우선 탐색(depth-first Search)
  - 리프까지 내려가면서 검색하는 것을 우선 순위로 하는 탐색 방법
  - 리프에 도달해 더 이상 검색을 진행할 곳이 없는 경우 부모에게 돌아감
  - 이후 자식 노드로 내려감



[그림 10-4] 깊이 우선 탐색

노드 순회 방법

- 노드 방문



[그림 10-5] 깊이 우선 탐색에서 가능한 방문의 종류

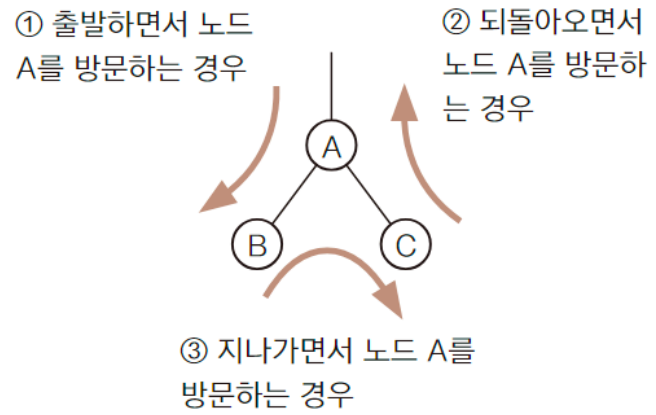
1. A에서 B로 내려가며 A를 지나갑니다.
2. B에서 C로 지나가며 A를 지나갑니다.
3. C에서 A로 되돌아오면서 A를 지나갑니다.

언제 노드를 방문할 것인가?

전위 순회

- 전위 순회(Preorder)
  노드 방문 → 왼쪽 자식 → 오른쪽 자식

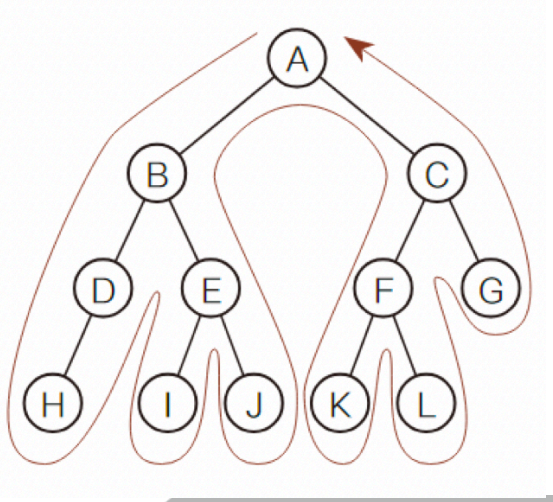
A 방문 → B로 이동 → C로 이동



[그림 10-5] 깊이 우선 탐색에서 가능한 방문의 종류

전위 순회로 깊이 우선 탐색 진행한다면

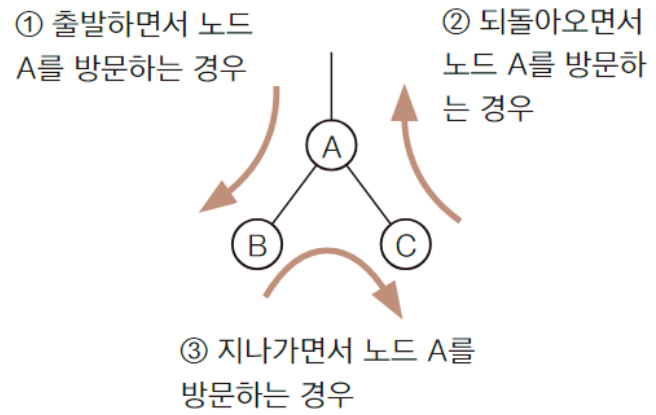
A → B → D → H → E → I → J → C → F → K → L → G



중위 순회

- 중위 순회(Inorder)
  왼쪽 자식 → 노드 방문 → 오른쪽 자식

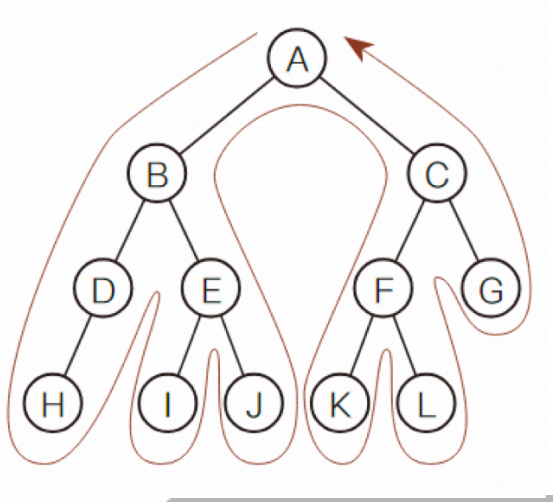
B로 이동 → A 방문 → C로 이동



[그림 10-5] 깊이 우선 탐색에서 가능한 방문의 종류

중위 순회로 깊이 우선 탐색 진행한다면

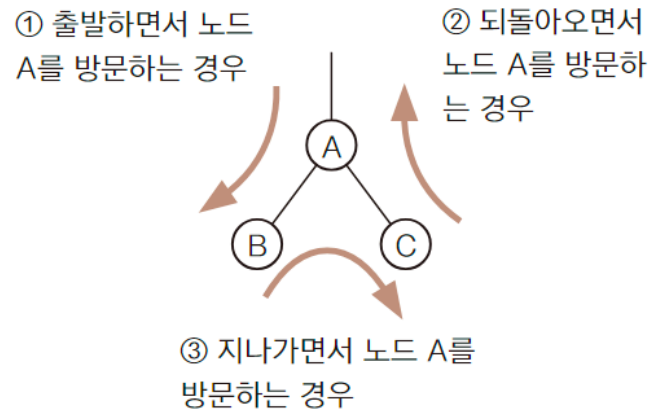
H → D → B → I → E → J → A → K → F → L → C → G



후위 순회

- 후위 순회(Postorder)
  왼쪽 자식 → 오른쪽 자식 → (돌아와) 노드 방문

B로 이동 → C로 이동 → A 방문



[그림 10-5] 깊이 우선 탐색에서 가능한 방문의 종류

후위 순회로 깊이 우선 탐색 진행한다면

H → D → I → J → E → B → K → L → F → G → C → A