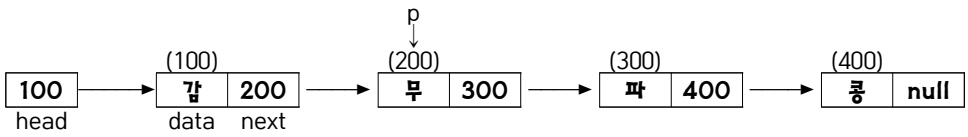
4. 단순연결리스트(singly linked list)

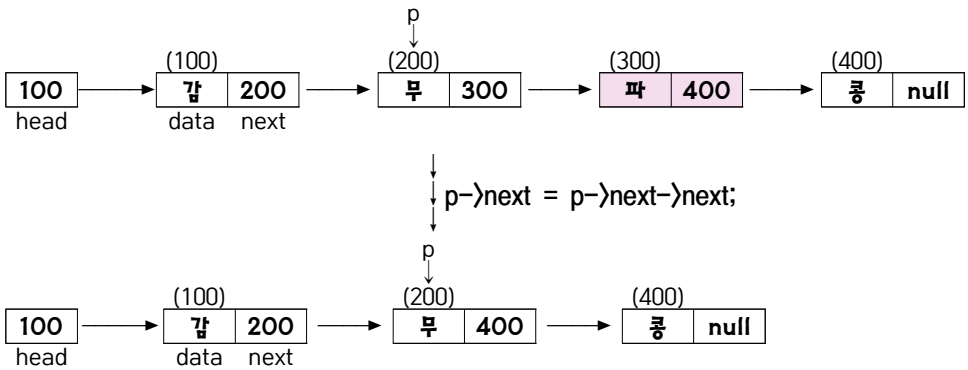
단순연결리스트의 각 노드는 하나의 포인터를 가진다.

```
struct node
{
    char data[10];
    struct node *next;   //next는 다음 노드를 가리키는 포인터
};
```



- 단순연결리스트에서는 거꾸로 가는 방법은 없다.
- 탐색은 첫 번째 노드부터 시작되어야 한다.
- 탐색시간은 포인터를 이용하여 순차적으로 추적해야 하므로 $O(n)$ 이다.

① 포인터 p가 가리키는 다음 노드 삭제



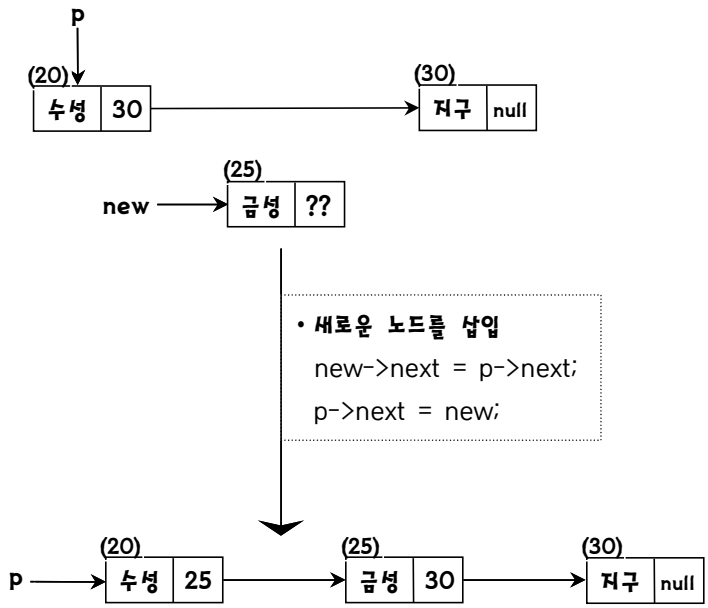
- 포인터 p가 가리키는 다음 노드 삭제 연산시간 : O(1)

[질문] 여기서, 포인터 p가 가리키는 이전 노드 삭제 연산시간은 뭘까?

- 정답은 O(n)이다. 이유는 거꾸로 갈 수 없으므로 첫 번째 노드부터 탐색되어야 한다.
- 만약, 삭제할 노드의 이전 노드 위치를 알 경우는 삭제 연산시간은 O(1)이다.

② 단순연결리스트에서 포인터 p가 가리키는 노드 다음에 새로운 노드 삽입

```
struct node
{
    char data[10];
    struct node *next;                                  //next는 다음 노드를 가리키는 포인터
};
new = (struct node *)malloc(sizeof(struct node));   //동적 메모리 할당 (힙 할당)
new->next = p->next;
p->next = new;
```



• 포인터 p가 가리키는 노드 다음에 새로운 노드 삽입 연산시간 : O(1)

// 탐구 – 단순연결리스트 구현

```
#define IRUM 10
typedef struct node
{
    char irum[IRUM];
    struct node *next;
}node;
node *head = NULL;

void front(){                                   //연결리스트 가장 앞에 자료 추가
    node *new;
    new = (node *)malloc(sizeof(node));
    printf("\nIrum....."); gets(new->irum);
    new->next = head;   head = new;
}
void back(){                                    //연결리스트 마지막에 자료 추가
    node *new, *temp;
    new = (node *)malloc(sizeof(node));
    printf("\nIrum....."); gets(new->irum);
    new->next = NULL;
    temp = head;
    while(1){ temp = temp->next;
              if(temp->next == NULL) { temp->next = new; break; }
    }
}
void insert(char *base, char *ir){              //연결리스트 중간에 자료 삽입
    node *new, *temp;
    new = (node *)malloc(sizeof(node));
    strcpy(new->irum,ir);
    temp = head;
    while(1){
      temp = temp->next;
      if(strcmp(temp->irum,base) == 0){ new->next=temp->next; temp->next=new; break; }
    }
}
void scan_list(){                               //연결리스트 내용 모두 출력
    node *scan = head;
    printf("<< 리스트 출력 >>\n");
    while(scan) { printf("%10s\n", scan->irum); scan = scan->next; }
}

void main(){
   char k = '1';
   while(1){
     switch(k){
       case '1': front(); break;
       case '2': back();  break;
       case '3': insert("사과", "수박"); break;   //자료 "사과" 다음에 "수박"을 삽입
       default : printf("종료...\n"); exit(0);
     }
     scan_list();
     printf("\nFront:1, Back:2, Insert:3 => "); k = getche();
   }
}
```

// 탐구 - 단순연결리스트 구현(python)

```
class Node:                                 # 클래스 노드(Node) 정의
    def __init__(self, data):               # 초기화(객체 생성 시 자동호출), self는 this와 같음
        self.data = data                    # 데이터 필드
        self.next = None                    # 링크 필드 (다음 노드를 가리킴)

    def __str__(self):                      # 메소드 str은 연결리스트 출력 형식을 지정
        return str(self.data)               # 파이썬에서 지원하는 특수 메소드는 앞뒤에 __

class SingleLinkedList:                     # 단순연결리스트 클래스 생성
    def __init__(self, data):               # 생성자 정의(공백 연결리스트에 새로운 노드 추가)
        new = Node(data)                    # 새로운 노드 생성
        self.head = new                     # 헤드가 새로 생성된 노드를 가리킴

    def __str__(self):                      # 연결리스트를 차례로 출력하기 위한 것(문자열)
        print_list = '[ '                   # str은 문자열 객체를 생성하는 클래스이다.
        node = self.head                    # str은 메소드처럼 보이는데 실제는 클래스이다.
        while True:
            print_list += str(node)
            if node.next == None:           # None은 Null과 같은 것
                break
            node = node.next
            print_list += ', '
        print_list += ' ]'
        return print_list

    def insertFirst(self, data):            # 연결리스트 처음에 새로운 노드 추가
        new = Node(data)                    # 새로운 노드 생성
        new.next = self.head                # 새로운 노드의 링크에 기존 헤드값 대입
        self.head = new                     # 헤드가 새로운 노드를 가리킴

    def insertLast(self, data):             # 연결리스트 마지막에 새로운 노드 추가
        new = Node(data)                    # 새로운 노드 생성
        temp = self.head                    # 임시 노드 temp 생성
        while True:
            if temp.next == None:           # temp.next == None이면(마지막 노드)
                break                       # 반복문 탈출
            temp = temp.next                # 연결리스트 추적
        temp.next = new                     # 연결리스트 마지막에 새로운 노드 추가

if __name__ == "__main__":
    L_list = SingleLinkedList(300)          # 연결리스트에 새로운 노드 300 추가
    print(L_list)                           # 출력 [ 300 ] : __str__에 정의된 형식 출력
    L_list.insertFirst(200)                 # 연결리스트에 처음에 새로운 노드 200 추가
    L_list.insertFirst(100)                 # 연결리스트에 처음에 새로운 노드 100 추가
    print(L_list)                           # 출력 [ 100, 200, 300 ]
    L_list.insertLast(400)                  # 연결리스트에 마지막에 새로운 노드 200 추가
    L_list.insertLast(500)                  # 연결리스트에 마지막에 새로운 노드 200 추가)
    print(L_list)                           # 출력 [ 100, 200, 300, 400, 500 ]
```

기출문제 분석

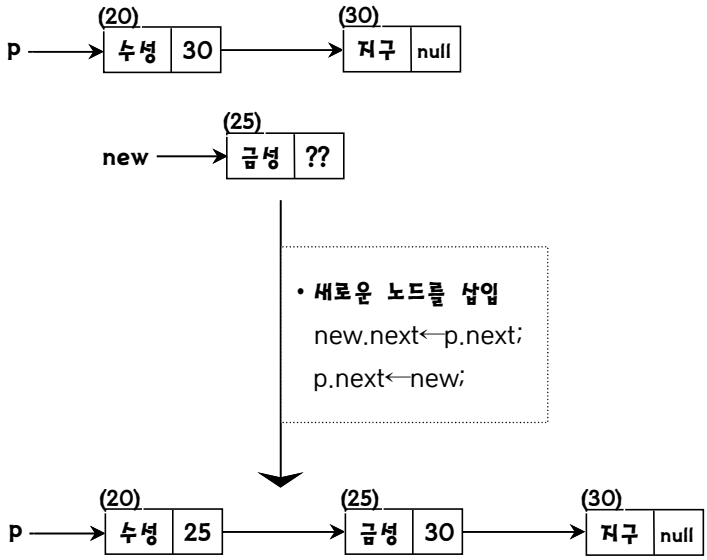
1. 단순연결리스트(singly linked list) L에서, 특정 노드 p 바로 뒤에 새로운 노드 new를 삽입하기 위한 연산 insertAfter의 의사코드가 다음과 같다. ㉠에 들어갈 내용으로 옳은 것은? [2011년 국가 7급]

```
Algorithm insertAfter(L, p, new)
  if L = NULL
    then L ← new;
    else [    ㉠    ]
  end if
  return;
```

① p.next←new; new.next←p.next;
② new.next←p.next; p.next←new;
③ new.next←p; p.next←new;
④ p.next←new; new.next←p;

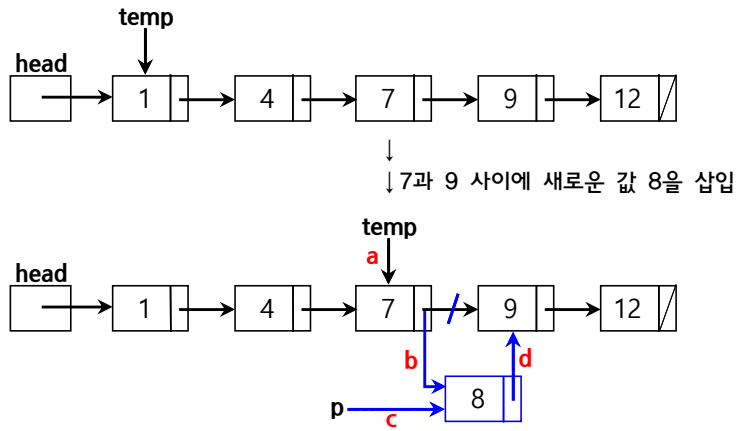
♤ 단순연결리스트에서 새로운 노드 new 삽입

• 이런 유형의 문제는 다음처럼 그림을 그려서 풀면 쉽게 답을 찾을 수 있다.



정답 : ②

2. 다음 연결리스트에서 7과 9 사이에 새로운 값 8을 삽입할 경우, 아래 a, b, c, d 연산들의 순서로 가장 옳은 것은? [2022년 군무원 7급]



---

① a → b → c → d
② a → c → b → d
③ c → a → d → b
④ c → a → b → d

♤ 연결리스트

```
// 7과 9 사이에 새로운 값 8을 삽입하는 과정을 다음과 같다.
 struct node
 {
    int data;
    struct node *next;                    //next는 다음 노드를 가리키는 포인터
 };
          ↓
          ↓노드 구조가 위와 같을 때
          ↓삽입 과정은 다음처럼 기술할 수 있다.(문제 기준)
          ↓
c : p = (struct node *)malloc(sizeof(struct node));      //삽입할 노드 메모리 할당
a : temp = 노드 7을 가리킴;      //탐색 결과 노드 7을 가리키도록 함
d : p->next = temp->next;    //삽입할 노드 8이 노드 9를 가리킴
b : temp->next = p;       //삽입할 노드 8을 가리킴
```

정답 : ③

3. n개의 데이터로 구성된 선형리스트(linear list)를 단순 연결리스트(singly linked list)로 표현하고자 한다. 다음 중 시간복잡도가 가장 낮은 연산은? [2009년 국가 7급]

① 포인터가 가리키는 노드의 다음 노드를 리스트에서 삭제
② 리스트의 길이를 출력
③ 포인터 값이 주어진 임의의 노드 앞에 새로운 노드 추가
④ 마지막 노드의 데이터를 출력

♤ **시간복잡도**

① 포인터가 가리키는 노드의 다음 노드를 리스트에서 삭제 → O(1)
② ③ ④는 O(n)이다.

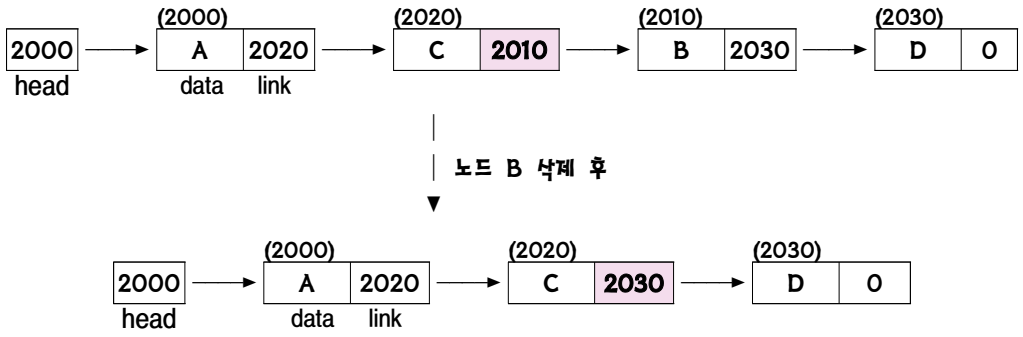
정답 : ①

4. 자료들이 단순 연결리스트(singly linked list)에 다음과 같이 구성되어 있을 때, 자료 B를 삭제한 후 변경된 내용으로 옳은 것은? [2009년 국가 7급]

메모리 주소	data	link
2000	A	2020
2010	B	2030
2020	C	2010
2030	D	0

① A의 link→2030 ② B의 link→2010
③ B의 link→2020 ④ C의 link→2030

♤ **단순 연결리스트에서 자료 삭제(그림 참조)**



정답 : ④

5. 다음의 표는 단순연결리스트(singly linked list)를 표현한 것으로 각 행은 각 노드의 주소, 데이터 및 다음 노드 주소로 구성되어 있다. 〈다음 노드 주소 값〉은 주소가 102인 노드를 삭제한 후 다음 노드 주소의 값을 설명한 것이다. ㉠~㉣에 들어갈 값을 바르게 연결한 것은? (단, 특정 노드의 다음 노드가 없을 때 그 노드의 다음 노드 주소 값은 NULL이다) [2020년 국가 7급]

주소	데이터	다음 노드 주소
100	LEE	NULL
101	PARK	104
102	KIM	101
103	JUNG	102
104	SEO	100

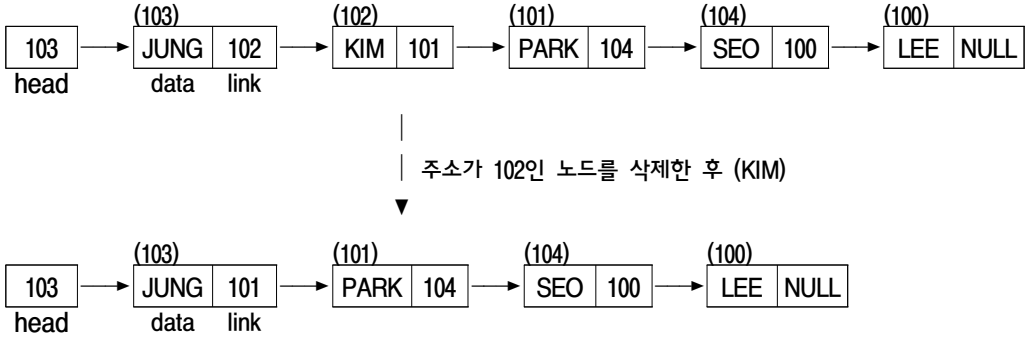
----<다음 노드 주소 값>----

○ 주소가 100인 노드의 다음 노드 주소는 (㉠)이다.
○ 주소가 101인 노드의 다음 노드 주소는 (㉡)이다.
○ 주소가 103인 노드의 다음 노드 주소는 (㉢)이다.
○ 주소가 104인 노드의 다음 노드 주소는 (㉣)이다.

	㉠	㉡	㉢	㉣
①	NULL	104	101	100
②	NULL	103	101	104
③	101	103	102	104
④	101	104	102	100

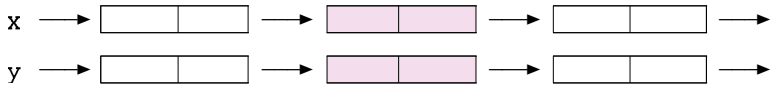
♤ 단순연결리스트에서 삭제

• 다음과 같은 단순연결리스트이다.(head의 주소는 **다음 노드 주소에 없는 값**이다)



정답 : ①

6. 구조체 list를 이용한 두 개의 단순연결리스트 x, y의 후위노드(음영부분)들을 서로 스왑(swap)하기 위한 코드는? (단, 후위노드들의 next는 NULL이 아니다) [2012년 국가 7급]




```
struct list { int data; struct list *next; } *x, *y, *tmp;
```

①
```
tmp = x->next;
x->next->next = y->next->next;
y->next->next = tmp;
tmp = y->next;
x->next = y->next;
y->next = tmp;
```

②
```
tmp = x->next->next;
x->next->next = y->next->next;
y->next->next = tmp;
tmp = x->next;
x->next = y->next;
y->next = tmp;
```

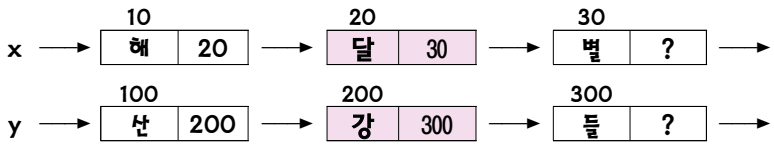
③
```
tmp = x->next;
x->next = y->next;
y->next = tmp;
tmp = x->next;
x->next->next = y->next->next;
y->next->next = tmp;
```

④
```
tmp = x->next->next;
y->next->next = tmp;
x->next->next = y->next->next;
tmp = x->next;
y->next = tmp;
x->next = y->next;
```

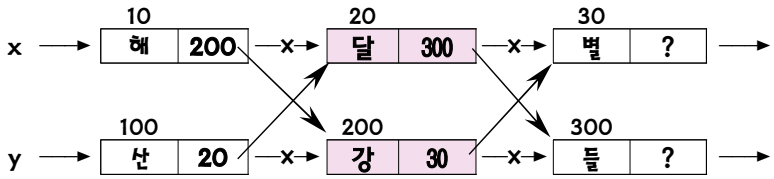
♤ **자기 참조 구조체**

• 실제 모습으로 설명한다. 먼저, 다음 포인터를 이해하면 매우 쉬운 문제이다.
x->next->next = 30, y->next->next = 300

• **스왑(swap)하기 전**



• **스왑(swap)한 결과**



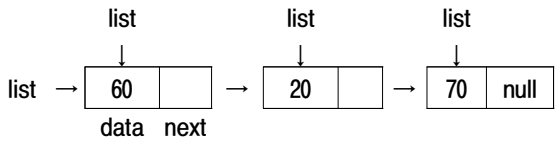
정답 : ②

7. 다음은 연결리스트의 노드 정의와 연결리스트의 노드들 중 data 필드 값이 50보다 큰 노드 수를 세는 함수를 C 언어로 표현하고 있다. 포인터 list가 연결리스트의 첫 노드를 가리킬 때, 괄호 [ ] 안에 들어갈 알맞은 표현은? [2002년 국가 기술고시]

```
----------------------------------
typedef struct
{
  int  data;
  node *next;
}node;
int count_node(node *list)
{
  int count = 0;
  for( [                    ] )
     if(list->data > 50) count++;
  return count;
}
----------------------------------
```

① ; list=list->next; list!=NULL
② ; list!=NULL; list=list->next
③ list=list->next; list->next!=NULL; list=list->next
④ list=list->next; list->next!=NULL;
⑤ list=list->next

♤ 연결리스트




```
int count_node(node *list)
{
  int count = 0;
  for( ; list!=NULL; list=list->next)    //리스트 추적
     if(list->data > 50) count++;        //data 필드 값이 50보다 큰 노드 수를 계산
  return count;
}
```

정답 : ②

8. 다음은 단순연결리스트(singly linked list)에서 특정 값을 가진 노드의 개수를 반환하는 C 함수이다. ㉠, ㉡에 들어갈 내용으로 옳은 것은? [2019년 국가 7급]

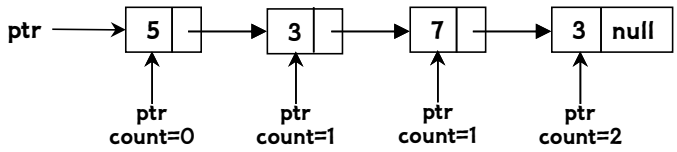
```
struct node {
  int data;
  struct node *link;
};
int countNode(struct node *ptr, int value) {
  int count = 0;
  while (          ㉠          ) {
    if ( ptr->data == value ) count++;
              ㉡
  }
  return count;
}
```

	㉠	㉡
①	ptr == NULL	ptr->link = ptr;
②	ptr == NULL	ptr = ptr->link;
③	ptr != NULL	ptr->link = ptr;
④	ptr != NULL	ptr = ptr->link;

♤ 단순연결리스트

```
while (㉠ ptr != NULL) {              //공백리스트 또는 ptr이 NULL이 아닌 경우
  if ( ptr->data == value ) count++;  //특정 값을 가진 노드 카운터
    ㉡ ptr = ptr->link;               //리스트 추적
}
```

// 특정 값(3)을 가진 노드 카운터



- 그림은 특정 값 3을 가진 노드를 카운터하는 것을 보여준다.
- ptr = ptr->link; //리스트 추적
- ptr = NULL이 되면, 조건식은 거짓이 되어 프로그램은 종료된다.

정답 : ④

9. 다음은 단순연결리스트(singly linked list)에서 특정 값을 가진 노드의 개수를 반환하는 C 함수이다. ㉠, ㉡에 들어갈 내용으로 옳은 것은? [2020년 군무 7급]

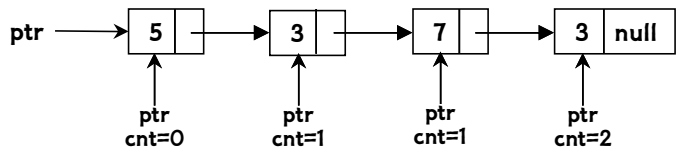
```
-----------------------------------------
struct node {
  int data;
  struct node *link;
};
int count(struct node *ptr, int value) {
  int cnt = 0;
  while (_____㉠_____) {
    if ( ptr->data == value ) cnt++;
       _____㉡_____
  }
  return cnt;
}
-----------------------------------------
```

	㉠	㉡
①	ptr == null	ptr->link = ptr;
②	ptr == null	ptr = ptr->link;
③	ptr	ptr->link = ptr;
④	ptr	ptr = ptr->link;

♤ 단순연결리스트

```
while (㉠ ptr) {                    //공백리스트 또는 마지막 노드가 아닌 경우
  if ( ptr->data == value ) cnt++;  //특정 값을 가진 노드 카운터
    ㉡ ptr = ptr->link;             //리스트 추적
}
```

◈ 특정 값(3)을 가진 노드 카운터

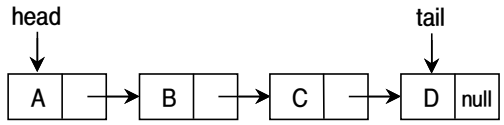


- 그림은 특정 값 3을 가진 노드를 카운터하는 것을 보여준다.
- ptr = ptr->link; //리스트 추적
- ptr = null이 되면, 조건식은 거짓이 되어 프로그램은 종료된다.

정답 : ④

10. 〈보기〉와 같은 형태의 단순연결리스트(singly linked list)에서 add_last 연산과 delete_last 연산의 수행시간에 대한 설명으로 가장 옳은 것은? (단, add_last는 tail 포인터가 가리키는 노드 다음에 새로운 노드를 생성하여 연결하는 연산이고, delete_last는 tail 포인터가 가리키는 노드를 삭제하는 연산이다) [2021년 서울 7급]

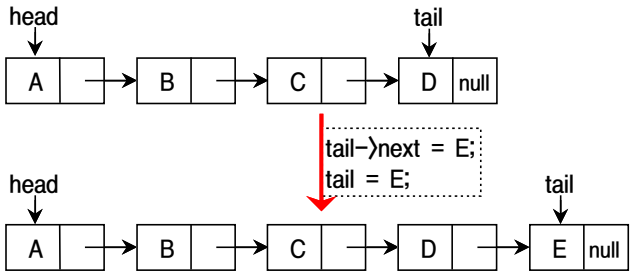
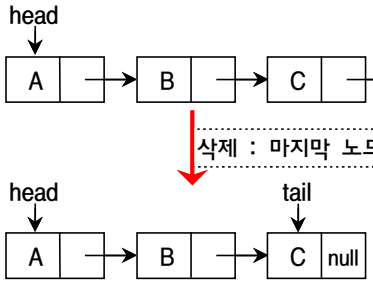
----<보기>----------



---

① 두 연산 모두 O(1) 시간에 수행된다.
② add_last는 O(1) 시간에 수행되나, delete_last는 O(1) 시간에 수행될 수 없다.
③ add_last는 O(1) 시간에 수행될 수 없으나, delete_last는 O(1) 시간에 수행된다.
④ add_last도 O(1) 시간에 수행될 수 없고, delete_last도 O(1) 시간에 수행될 수 없다.

♤ **단순연결리스트**

add_last		수행시간 : O(1)
delete_last	 • **마지막 노드 D**를 삭제하려면 **처음**부터 **추적**해야 한다 • 이유는 **단순연결리스트**는 거꾸로 추적할 수 없으므로 • 해서, **수행시간**은 O(n)이다	수행시간 : O(n)

• add_last는 O(1) 시간에 수행되나, delete_last는 O(1) 시간에 수행될 수 없다.

정답 : ②

11. C 언어에서 배열과 연결리스트에 대한 설명으로 옳지 않은 것은? [2021년 국가 7급]

① 일반적으로 두 개의 정수형 데이터 세트 {1, 2, 3}과 {4, 5, 6, 7, 8}을 합병하여 {1, 2, 3, 4, 5, 6, 7, 8}로 만드는 연산을 수행할 경우, 두 데이터 세트의 자료구조를 연결리스트로 만들면, 정수형 배열로 만드는 것보다 시간복잡도를 낮게 구현할 수 있다.

② 일반적으로 정수형 데이터 세트 {1, 2, 3, 4}를 {4, 1, 2, 3}으로 변경하는 연산을 수행할 경우, 해당 데이터 세트의 자료구조를 연결리스트로 만들면, 정수형 배열로 만드는 것보다 더 빠르게 수행시킬 수 있다.

③ 일반적으로 정수형 데이터 세트 {1, 2, 3, 4}를 {2, 3, 4, 5}로 각 요소에 +1 연산을 수행할 경우, 데이터 세트의 자료구조를 연결리스트로 만들면, 정수형 배열로 만드는 것보다 메모리를 적게 차지한다.

④ 일반적으로 정수형 데이터 세트 {1, 2, 3, 4}를 {0, 1, 2, 3, 4}로 새로운 값을 추가하는 연산을 수행할 경우, 데이터 세트의 자료구조를 연결리스트로 만들면, 정수형 배열로 만드는 것보다 더 빠르게 연산을 수행시킬 수 있다.

♤ 배열과 연결리스트

- 일반적으로 정수형 데이터 세트 {1, 2, 3, 4}를 {2, 3, 4, 5}로 각 요소에 +1 연산을 수행할 경우, 데이터 세트의 **자료구조**를 **연결리스트**로 만들면, **정수형 배열**로 만드는 것보다 메모리를 **적게** 차지한다.**(×)**

  → **자료구조**를 **연결리스트**로 만들면, **정수형 배열**로 만드는 것보다 메모리가 더 **많이** 필요하다.

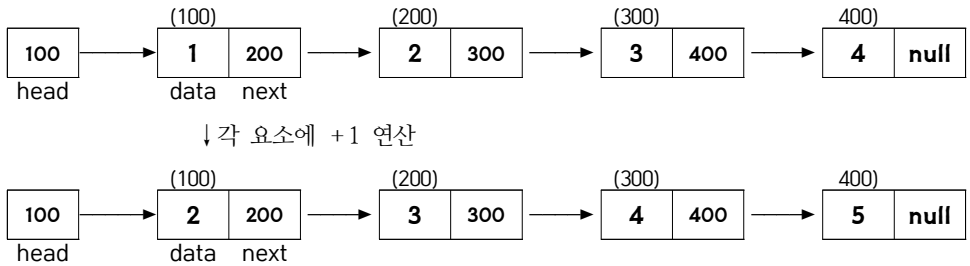
// 배열

정수형 데이터 세트	1	2	3	4

↓각 요소에 +1 연산

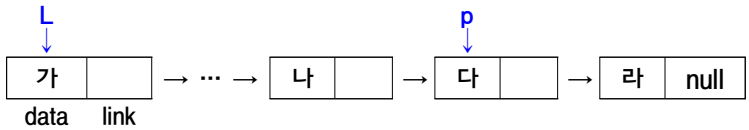
정수형 데이터 세트	2	3	4	5

// 연결리스트 - **포인터 필드가 추가로 필요**



정답 : ③

12. 다음과 같은 단순 연결리스트에 대해, 아래와 같은 C 언어로 작성된 프로그램을 수행한 후 포인터 temp가 가리키는 노드는? [2008년 국가 7급]




```
for(temp = L; temp->link != p; temp = temp->link) ;
temp->link = p->link;
```

---

① 가 ② 나
③ 다 ④ 라

♤ 포인터 p가 가리키는 노드(다)를 삭제하는 알고리즘이다.

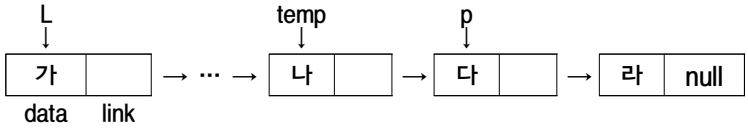
// 문제 분석

`for(temp = L; temp->link != p; temp = temp->link) ;` // 끝에 세미콜론 ;이 있는 것에 주의

↓

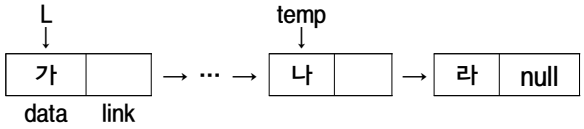
세미콜론 ;이 있으므로 for문만 계속 반복되다가 조건이 거짓이 되면 벗어난다.

`temp->link = p->link;` // 조건이 거짓이 되면 실행



• temp가 노드 '나'를 가리킬 때 조건 temp->link == p는 거짓, 반복문을 벗어난다.
• 반복문을 벗어난 후에 temp->link = p->link;가 수행된다.

• temp->link = p->link;가 수행되면, 포인터 p가 가리키는 노드(다)는 **삭제**된다.



• 하지만, temp가 가리키는 노드 '나'이다.

정답 : ②

13. 〈보기〉와 같이 단순연결리스트(singly linked list)가 메모리에 저장되어 있다. 이 외의 다른 정보는 메모리에 저장되어 있지 않다고 할 때, 리스트에 저장된 원소를 처음부터 링크를 따라 순서대로 바르게 나열한 것은? [2021년 서울 7급]

----<보기>----------

주소	데이터	다음 노드 주소 링크
100	31	NULL
104	41	116
108	59	112
112	26	104
116	53	100

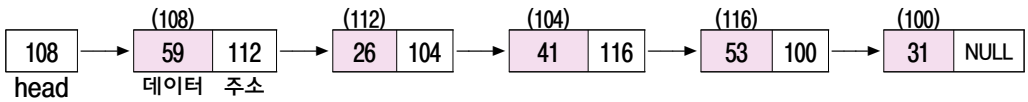
---

① 59-26-41-53-31 ② 31-41-59-26-53

③ 53-26-59-41-31 ④ 31-53-41-26-59

♤ 단순연결리스트

• 다음과 같은 단순연결리스트이다.(head의 주소는 **다음 노드 주소 링크에 없는 값**이다)



정답 : ①