

프로그래밍 및 연습 1

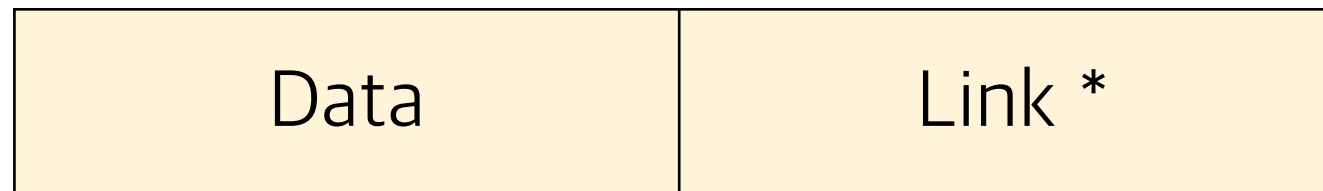
연결리스트 2 / 정렬 알고리즘

목차

- 연결 리스트 2
- 정렬 알고리즘

연결 리스트

- 연결 리스트(Linked List)
 - “각 노드가 데이터와 포인터를 가지고 있으며 한 줄로 연결되어 있는 방식으로 데이터를 저장하는 자료구조”
 - 노드는 데이터 필드, 링크 필드로 되어 있음



- 데이터 필드에는 실질적인 데이터가 들어 있음
- 링크 필드는 다음 노드의 메모리 주소값(포인터)를 갖고 있음

연결 리스트

- 연결 리스트(Linked List)
 - 장점
 - 추가, 삭제가 (배열보다) 쉽다
 - 필요할 때마다 저장 공간을 요구하므로 효율적인 메모리 관리가 가능
 - 단점
 - 구현이 어려움
 - 특정 위치 데이터 검색에는 시간이 오래 걸림

연결 리스트

- 연결 리스트(Linked List)

- 종류

- 단일 연결리스트

- 단방향

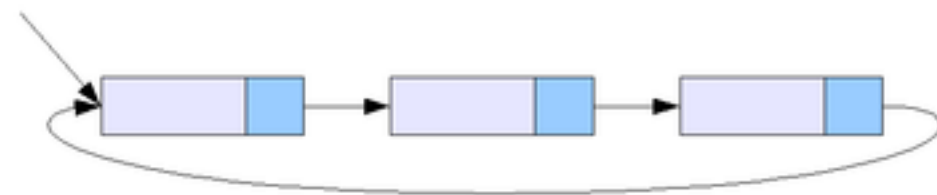
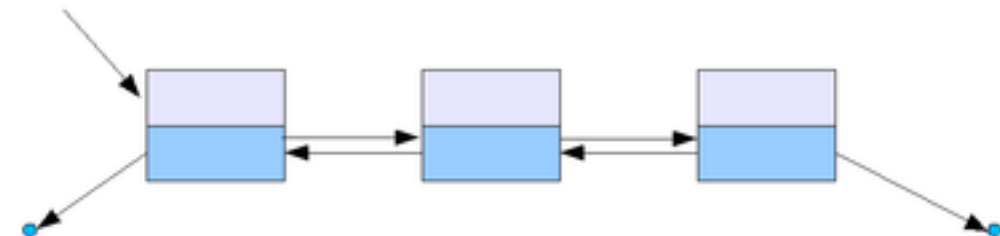
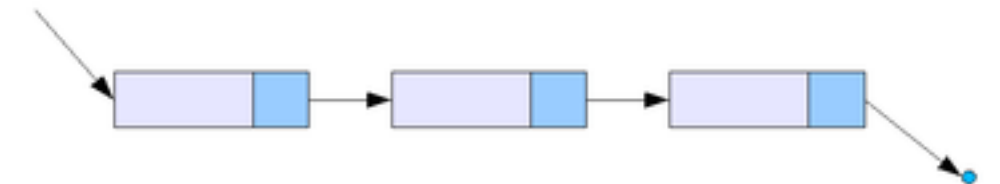
- 이중 연결 리스트

- 양방향

- 원형 연결 리스트

- 마지막 노드에서 첫 노드로 연결

- 환형 큐와 유사하다고 보면 됨

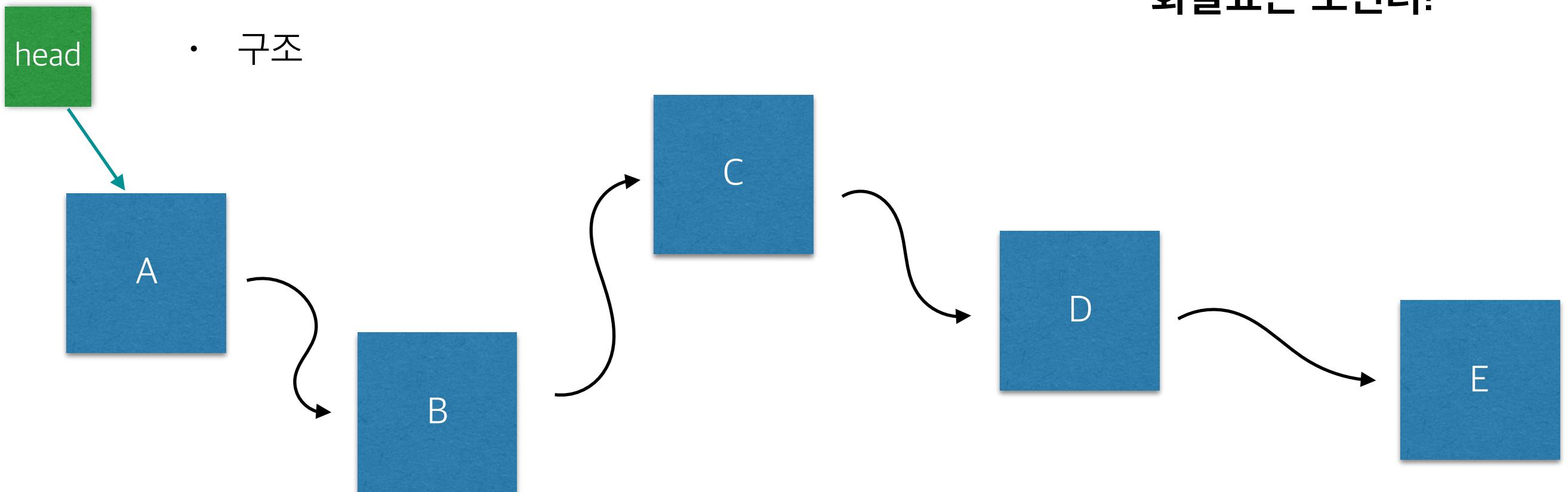


연결 리스트

헤드 포인터(Head Pointer)
첫 노드를 가리키는 포인터!

- 연결 리스트(Linked List)
- 구조

화살표는 포인터!



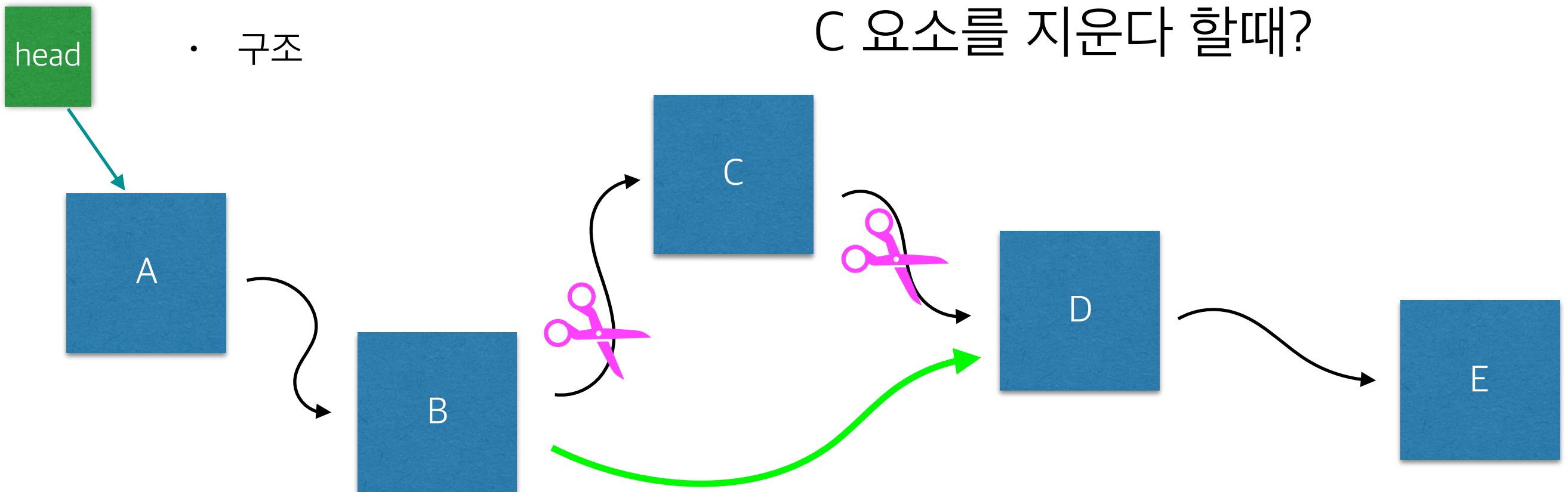
- 위의 구조를 배열로 나타낸다면?

A	B	C	D	E
---	---	---	---	---

연결 리스트

헤드 포인터(Head Pointer)
첫 노드를 가리키는 포인터!

- 연결 리스트(Linked List)
- 구조



- 위의 구조를 배열로 나타낸다면?

A	B		D	E
---	---	--	---	---

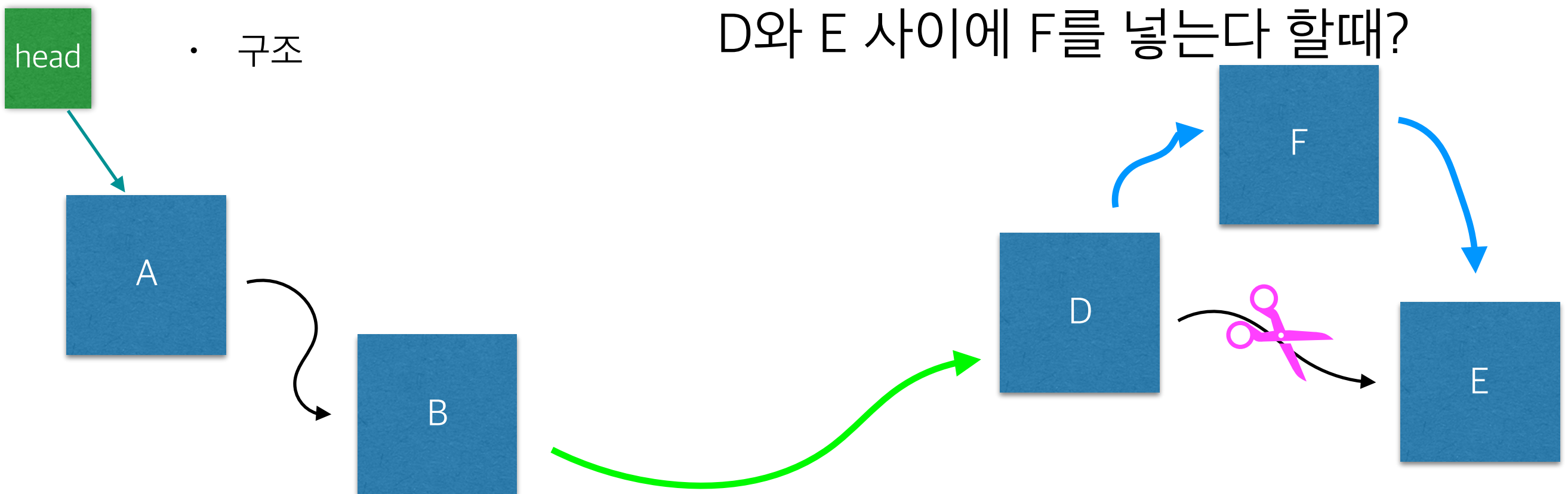
연결 리스트

헤드 포인터(Head Pointer)
첫 노드를 가리키는 포인터!

- 연결 리스트(Linked List)

- 구조

D와 E 사이에 F를 넣는다 할때?



- 위의 구조를 배열로 나타낸다면?

A	B	D	F	E
---	---	---	---	---

연결 리스트

- 연결 리스트(Linked List)
 - 노드 작성
 - typedef을 이용해 노드 구조체 정의 가능
 - struct NODE * link와 같이 같은 타입의 구조체를 가리키는 포인터를 가지는 구조체를 **자기 참조 구조체(Self-referential Structure)**라 함

```
typedef struct NODE{  
    int data;  
    struct NODE * link  
} NODE;
```

Q:
L

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _node{
    int data;
    struct _node *next;
} NODE;

int main(){
    NODE *head= NULL; //헤드 포인터 (더미 노드를 가짐)
    NODE *node1 = NULL; // 첫 번째 노드
    NODE *node2 = NULL; // 두 번째 노드
    NODE * currentNode = NULL; // 노드를 가리키고자 사용하는 노드 포인터.

    head = (NODE *)malloc(sizeof(NODE));
    //머리 노드 생성. 이 노드는 어떠한 데이터를 갖지 않는 더미 노드임.

    node1 = (NODE *)malloc(sizeof(NODE));
    head->next = node1;
    node1->data = 20;
    // 헤드 포인터(더미 노드)가 가리키는 첫 노드는 node1이 됨.
    // 첫 노드의 데이터는 20을 가짐.

    node2 = (NODE *)malloc(sizeof(NODE));
    node1->next = node2;
    node2->data = 30;
    //node1의 노드의 다음 노드는 node2.
    //두 번째 노드의 데이터는 30을 가짐.

    node2->next = NULL;
    //node2를 마지막으로 다음 노드는 존재하지 않음.

    currentNode = head->next;
    //현재의 노드를 나타내는 노드 포인터. 첫 노드를 가리키고 있음.

    while(currentNode != NULL){
        printf("%d\n", currentNode->data);
        currentNode = currentNode->next;
    }

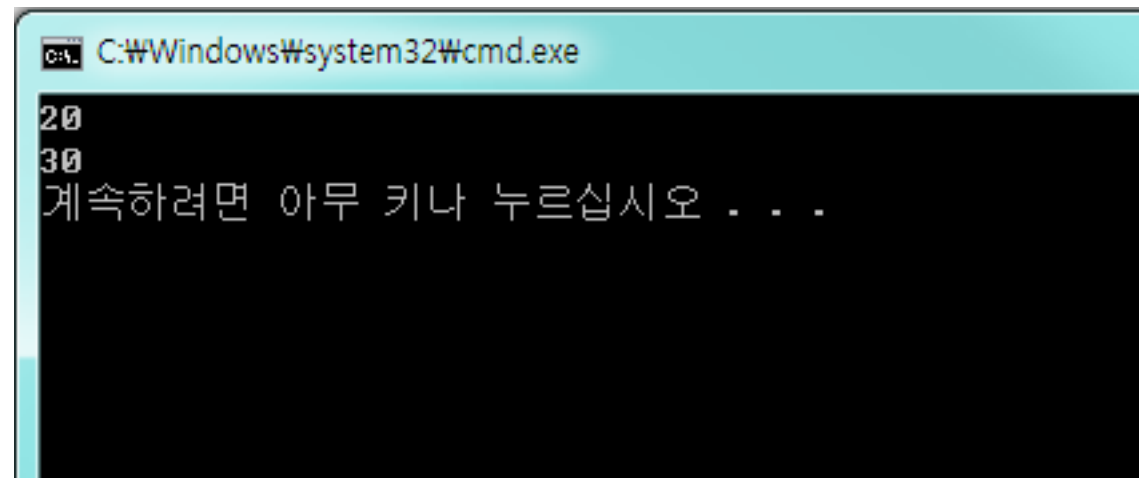
    free(node2);
    free(node1);
    free(head);
    return 0;
}
```

- 연결 리스트(Linked List)

- 간단 예제 코드

연결 리스트

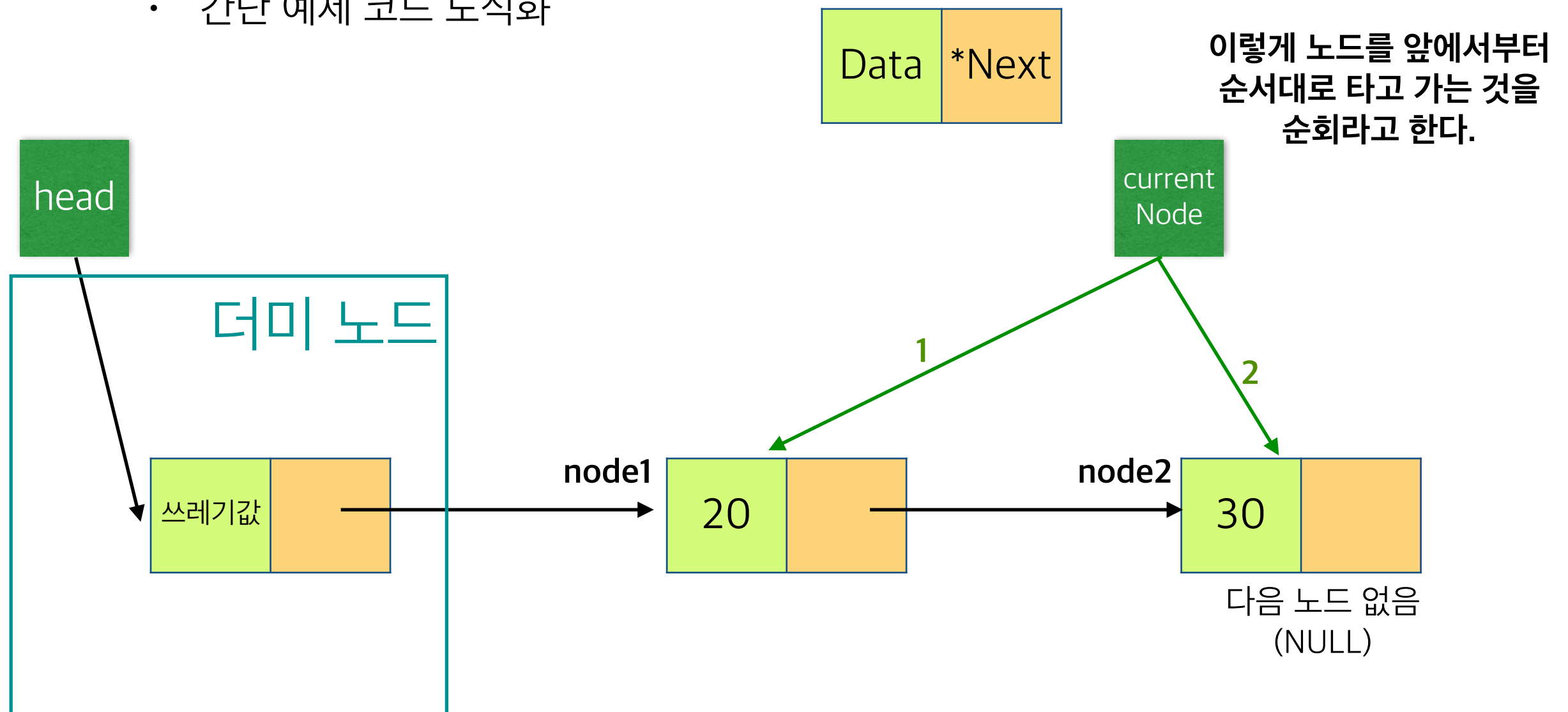
- 연결 리스트(Linked List)
 - 간단 예제 코드 결과



```
C:\Windows\system32\cmd.exe
20
30
계속하려면 아무 키나 누르십시오 . . .
```

연결 리스트

- 연결 리스트(Linked List)
- 간단 예제 코드 도식화



더미 노드란? 의미없는 노드. 구현을 위해 넣은 노드.

연결

- 연결 리스트(Linked List)
 - 노드 생성 예제 코드 및 결과
 - insertNodeAtFirst
 - 맨 앞에 노드 추가!

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _node{
    int data;
    struct _node *next;
} NODE;

void insertNodeAtFirst(NODE * node, int data){
    NODE * newNode = (NODE *)malloc(sizeof(NODE));
    newNode->next = node->next;
    newNode->data = data;
    node->next = newNode;
}

int main(){
    NODE * head = (NODE *)malloc(sizeof(NODE)); //헤드 포인터
    NODE * currentNode; //현재 노드를 가리키는 노드 포인터
    NODE * tempNext; //다음 노드 주소값을 갖기 위한 임시 노드 포인터.

    head->next = NULL;

    insertNodeAtFirst(head, 10); // 맨 앞에 10의 데이터를 가지는 노드 저장
    insertNodeAtFirst(head, 20); // 맨 앞에 20의 데이터를 가지는 노드 저장
    insertNodeAtFirst(head, 30); // 맨 앞에 30의 데이터를 가지는 노드 저장

    currentNode = head->next;

    //전체 노드 출력하기 위한
    while(currentNode != NULL){
        printf("%d\n", currentNode->data);
        currentNode = currentNode->next;
    }

    currentNode = head->next;
    //삽입한 노드 메모리들을 앞에서부터 반납하기 위한 반복문.
    while(currentNode != NULL){
        tempNext = currentNode->next; // 다음노드 주소값을 임시 저장
        free(currentNode); //insertNodeAtFirst에서 할당받은 동적할당메모리 반납
        currentNode = tempNext; //임시 저장한 다음 노드 주소값 저장.
    }

    free(head);
    return 0;
}
```

C:\Windows\system32\cmd.exe

30
20
10

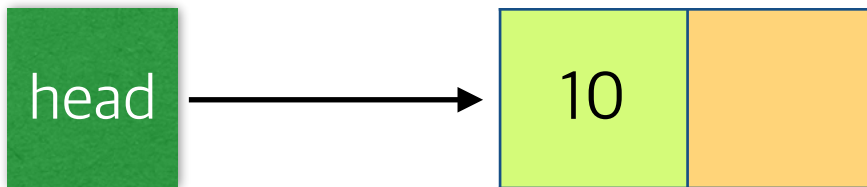
계속하려면 아무 키나 누르십시오 . . .

연결 리스트

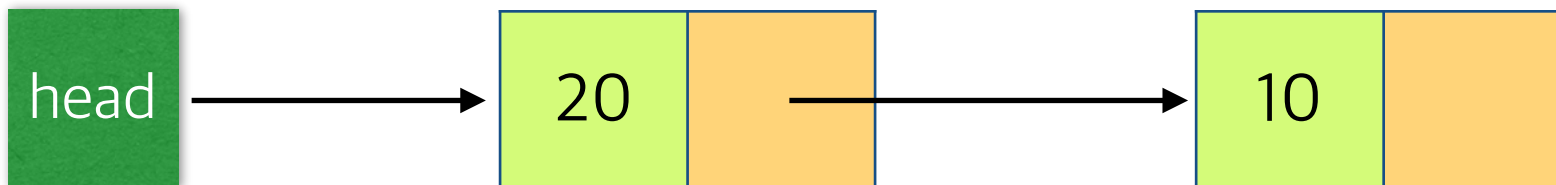
- 연결 리스트(Linked List)
 - 노드 생성 예제 도식화(더미 노드 생략)

head

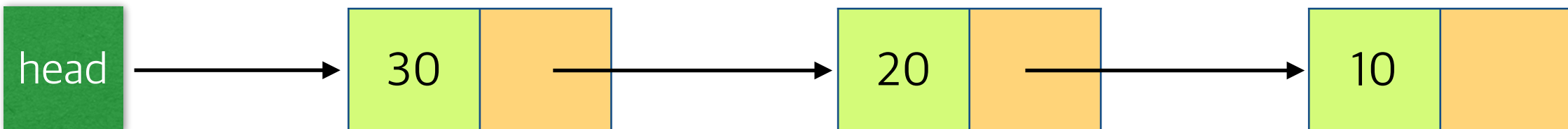
insertNodeAtFirst 함수 실행 전



insertNodeAtFirst(head,10) 이후



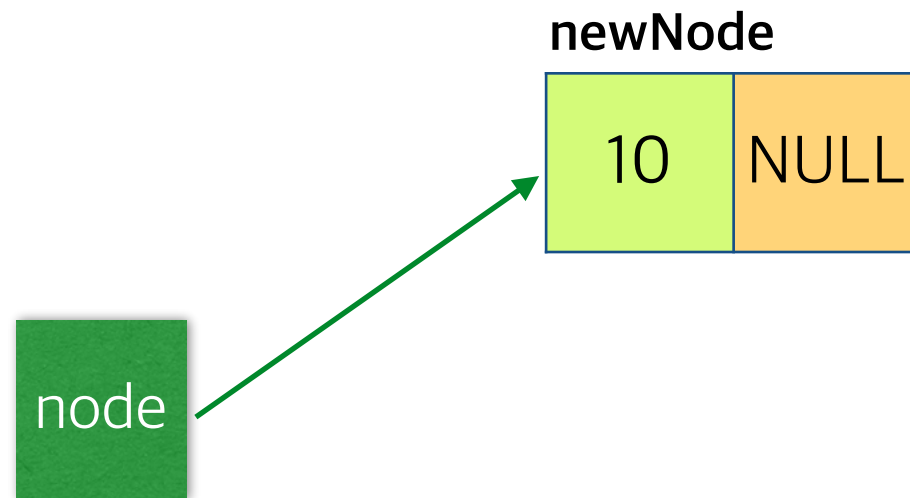
insertNodeAtFirst(head,20) 이후



insertNodeAtFirst(head,30)이후

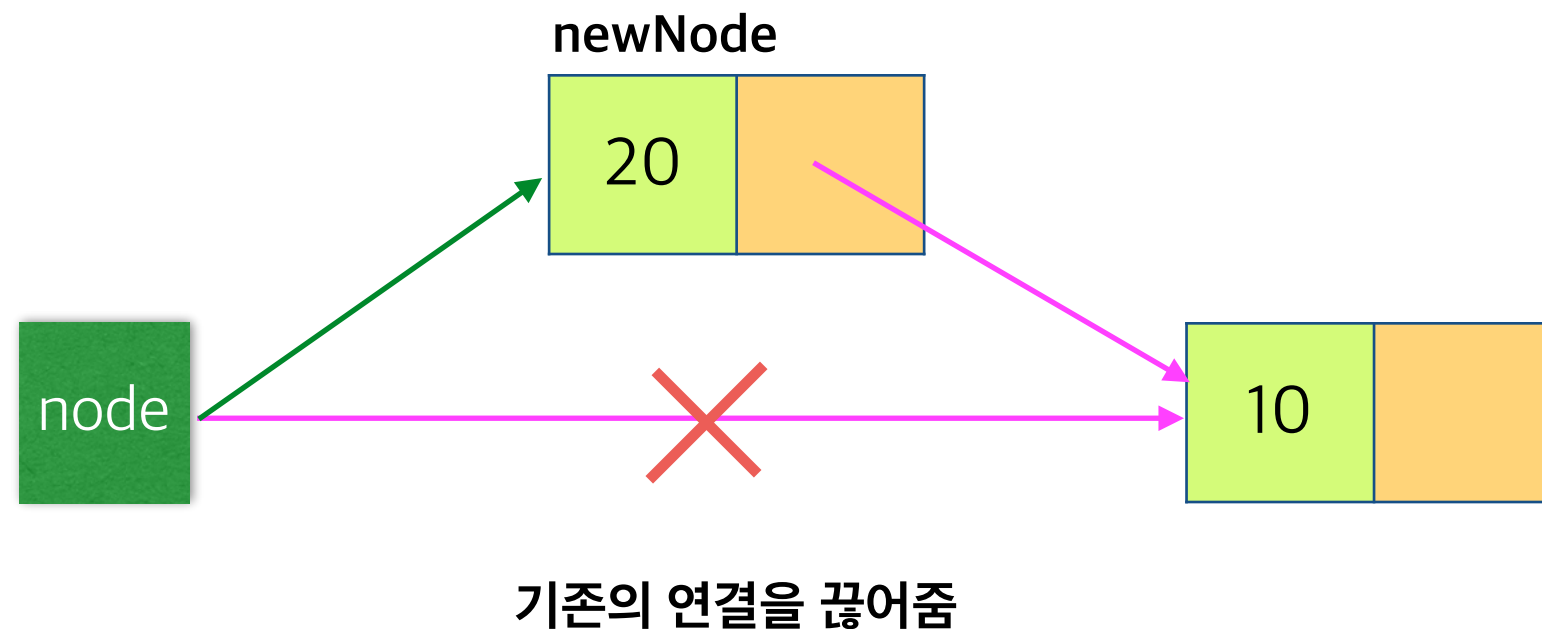
연결 리스트

- 연결 리스트(Linked List)
 - 노드 생성 예제 insertNodeAtFirst 함수 도식화(더미 노드 생략)
 - insertNodeAtFirst(head,10) 인 경우



연결 리스트

- 연결 리스트(Linked List)
 - 노드 생성 예제 insertNodeAtFirst 함수 도식화(더미 노드 생략)
 - insertNodeAtFirst(head,20) 인 경우



연결 리스트

- 연결 리스트(Linked List)
 - 노드 삭제 예제 코드 및 결과
 - removeNodeAtFirst
 - 맨 앞 노드 삭제!

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _node{
    int data;
    struct _node *next;
} NODE;

void insertNodeAtFirst(NODE * node, int data){
    NODE * newNode = (NODE *)malloc(sizeof(NODE));
    newNode->next = node->next;
    newNode->data = data;
    node->next = newNode;
}

void removeNodeAtFirst(NODE *node){
    NODE * removeNode = node->next;
    node->next = removeNode->next;

    free(removeNode); // 노드 메모리 반납
}
```

연결

- 연결 리스트(Linked List)
 - 노드 삭제 예제 코드 및 결과
 - removeNodeAtFirst
 - 맨 앞 노드 삭제!

```
int main(){
    NODE * head = (NODE *)malloc(sizeof(NODE)); //헤드 포인터
    NODE * currentNode; //현재 노드를 가리키는 노드 포인터
    NODE * tempNext; //다음 노드 주소값을 갖기 위한 임시 노드 포인터.

    head->next = NULL;

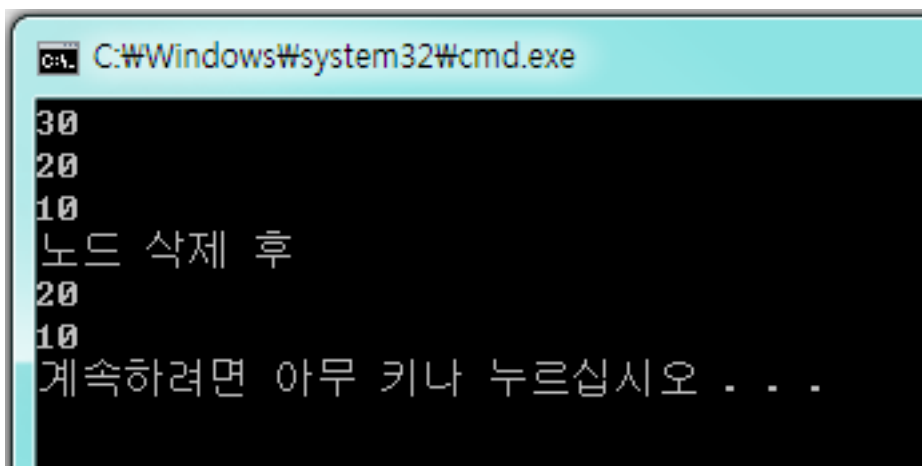
    insertNodeAtFirst(head, 10); // 맨 앞에 10의 데이터를 가지는 노드 저장
    insertNodeAtFirst(head, 20); // 맨 앞에 20의 데이터를 가지는 노드 저장
    insertNodeAtFirst(head, 30); // 맨 앞에 30의 데이터를 가지는 노드 저장

    currentNode = head->next;

    //전체 노드 출력하기 위한
    while(currentNode != NULL){
        printf("%d\n", currentNode->data);
        currentNode = currentNode->next;
    }

    removeNodeAtFirst(head);
    printf("노드 삭제 후\n");
    //전체 노드 출력하기 위한
    currentNode = head->next;
    while(currentNode != NULL){
        printf("%d\n", currentNode->data);
        currentNode = currentNode->next;
    }

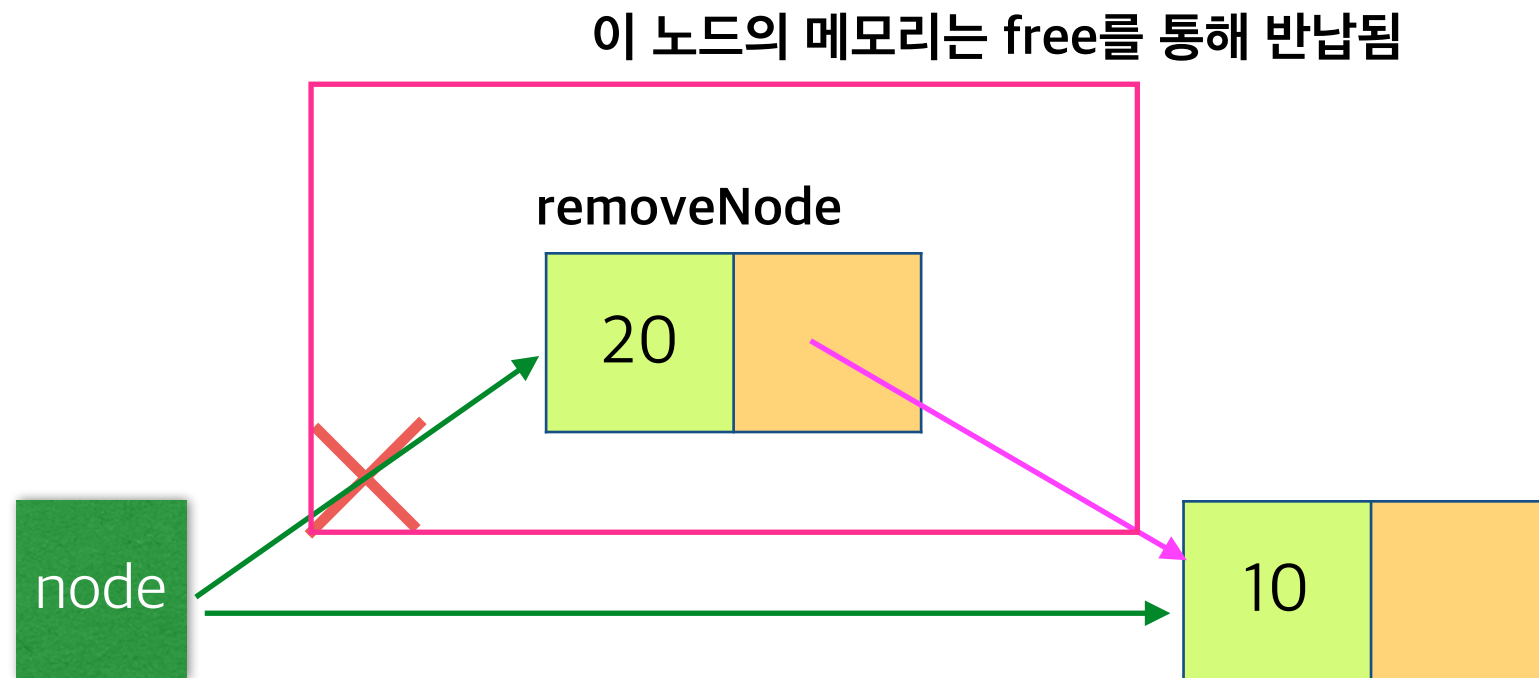
    currentNode = head->next;
    //삽입한 노드 메모리들을 앞에서부터 반납하기 위한 반복문.
    while(currentNode != NULL){
        tempNext = currentNode->next; // 다음노드 주소값을 임시 저장
        free(currentNode); //insertNodeAtFirst에서 할당받은 동적할당메모리 반납
        currentNode = tempNext; //임시 저장한 다음 노드 주소값 저장.
    }
    free(head);
    return 0;
}
```



```
C:\Windows\system32\cmd.exe
30
20
10
노드 삭제 후
20
10
계속하려면 아무 키나 누르십시오 . . .
```

연결 리스트

- 연결 리스트(Linked List)
 - 노드 삭제 코드 도식화 removeNodeAtFirst



연결 리스트

- 연결 리스트(Linked List)
 - 노드 검색 예제 코드 및 결과

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _node{
    int data;
    struct _node *next;
} NODE;

void insertNodeAtFirst(NODE * node, int data){
    NODE * newNode = (NODE *)malloc(sizeof(NODE));
    newNode->next = node->next;
    newNode->data = data;
    node->next = newNode;
}

void removeNodeAtFirst(NODE *node){
    NODE * removeNode = node->next;
    node->next = removeNode->next;

    free(removeNode); // 노드 메모리 반납
}

NODE * findNode(NODE * node, int data){
    NODE * currentNode = node->next;
    if (node == NULL){ //노드가 없으므로 NULL 값 반납
        return NULL;
    }

    while(currentNode!=NULL){
        if(currentNode->data == data){
            return currentNode;
        }
        currentNode = currentNode->next;
    }
    return NULL;
}
```



연결 리스트

- 연결 리스트(Linked List)
 - 노드 검색 예제 코드 및 결과

```
int main(){
    NODE * head = (NODE *)malloc(sizeof(NODE)); //헤드 포인터
    NODE * currentNode; //현재 노드를 가리키는 노드 포인터
    NODE * tempNext; //다음 노드 주소값을 갖기 위한 임시 노드 포인터.
    NODE * foundNode = NULL;

    head->next = NULL;

    insertNodeAtFirst(head, 10); // 맨 앞에 10의 데이터를 가지는 노드 저장
    insertNodeAtFirst(head, 20); // 맨 앞에 20의 데이터를 가지는 노드 저장
    insertNodeAtFirst(head, 30); // 맨 앞에 30의 데이터를 가지는 노드 저장

    currentNode = head->next;

    //전체 노드 출력하기 위함
    while(currentNode!= NULL){
        printf("%d\\n", currentNode->data);
        currentNode = currentNode->next;
    }
    foundNode = findNode(head, 20);

    if(foundNode == NULL){
        printf("찾고자 하는 데이터를 가진 노드 없음");
    }else{
        printf("찾은 데이터 값 : %d\\n", foundNode->data);
    }

    currentNode = head->next;
    //삽입한 노드 메모리들을 앞에서부터 반납하기 위한 반복문.
    while(currentNode!= NULL){
        tempNext = currentNode->next; // 다음노드 주소값을 임시 저장
        free(currentNode); //insertNodeAtFirst에서 할당받은 동적할당메모리 반납
        currentNode = tempNext; //임시 저장한 다음 노드 주소값 저장.
    }
    free(head);
    return 0;
}
```

연결 리스트

- 연결 리스트(Linked List)
 - 노드 검색 예제 코드 및 결과

데이터가 있는 경우

```
C:\Windows\system32\cmd.exe
30
20
10
찾은 데이터 값 : 20
계속하려면 아무 키나 누르십시오 . . .
```

데이터가 없는 경우

```
currentNode = currentNode->next;
}
foundNode = findNode(head, 25);

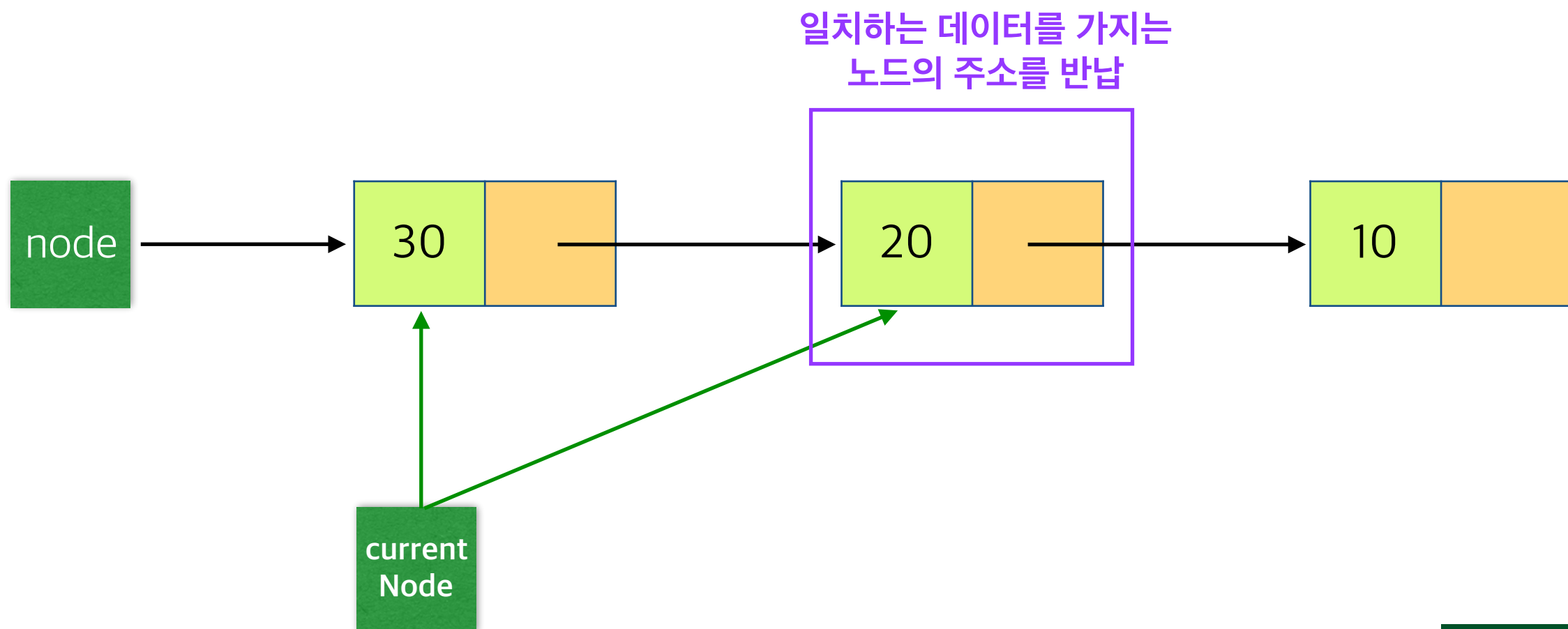
if(foundNode == NULL){
    printf("찾고자 하는 데이터를 가진 노드 없음\n");
}else{
    printf("찾은 데이터 값 : %d\n", foundNode->data);
}

currentNode = head->next;
//삽입한 노드 메모리들을 앞에서부터 반납하기
while(currentNode != NULL){
    tempNext = currentNode->next; // 다음 노드
```

```
C:\Windows\system32\cmd.exe
30
20
10
찾고자 하는 데이터를 가진 노드 없음
계속하려면 아무 키나 누르십시오 . . .
```

연결 리스트

- 연결 리스트(Linked List)
 - 노드 탐색 코드 도식화 findNode



연결 리스트

- 연결 리스트(Linked List)
 - 노드 삽입 예제 코드 (오름차순 삽입)
 - 예를 들어
 - 25, 35, 15, 55, 5 삽입시 노드는 5, 15, 25, 35, 55 순으로 연결됨

연결

- 연결 리스트(Linked List)
 - 노드 삽입 예제 코드
(오름차순 삽입)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _node{
    int data;
    struct _node *next;
} NODE;

void insertNode(NODE * node, int data){
    NODE * newNode = (NODE *)malloc(sizeof(NODE));
    NODE * currentNode = node;
    if(node->next == NULL){
        newNode->next = node->next;
        newNode->data = data;
        node->next = newNode;
        return;
    }
    while(currentNode != NULL){
        if(currentNode->data <= data){
            if(currentNode->next == NULL){
                newNode->next = NULL;
                newNode->data = data;
                currentNode->next = newNode;
                break;
            }
            else if(currentNode->next->data > data){
                newNode->next = currentNode->next;
                newNode->data = data;
                currentNode->next = newNode;
                break;
            }
        }
        else{
            newNode->next = currentNode->next;
            newNode->data = data;
            currentNode->next = newNode;
            break;
        }
        currentNode = currentNode->next;
    }
}
```

연결 리스트

- 연결 리스트(Linked List)
 - 노드 삽입 예제 코드
(오름차순 삽입)

```
int main(){
    NODE * head = (NODE *)malloc(sizeof(NODE)); //헤드 포인터
    NODE * currentNode; //현재 노드를 가리키는 노드 포인터
    NODE * tempNext; //다음 노드 주소값을 갖기 위한 임시 노드 포인터.
    NODE * foundNode = NULL;

    head->next = NULL;

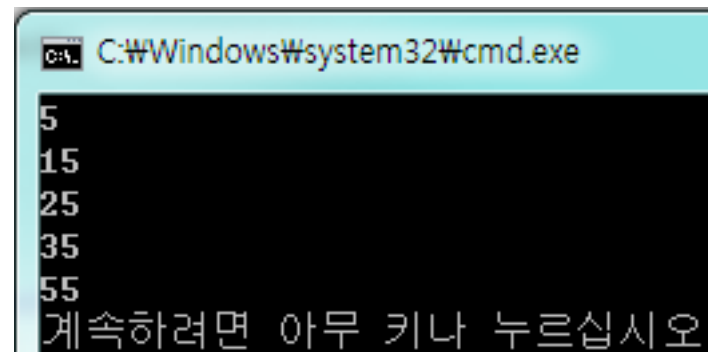
    insertNode(head, 25);
    insertNode(head, 35);
    insertNode(head, 15);
    insertNode(head, 55);
    insertNode(head, 5);
    currentNode = head->next;

    //전체 노드 출력하기 위한
    while(currentNode!= NULL){
        printf("%d\n", currentNode->data);
        currentNode = currentNode->next;
    }

    currentNode = head->next;
    //삽입한 노드 메모리들을 앞에서부터 반납하기 위한 반복문.
    while(currentNode!= NULL){
        tempNext = currentNode->next; // 다음노드 주소값을 임시 저장
        free(currentNode); //insertNodeAtFirst에서 할당받은 동적할당메모리 반납
        currentNode = tempNext; //임시 저장한 다음 노드 주소값 저장.
    }
    free(head);
    return 0;
}
```

연결 리스트

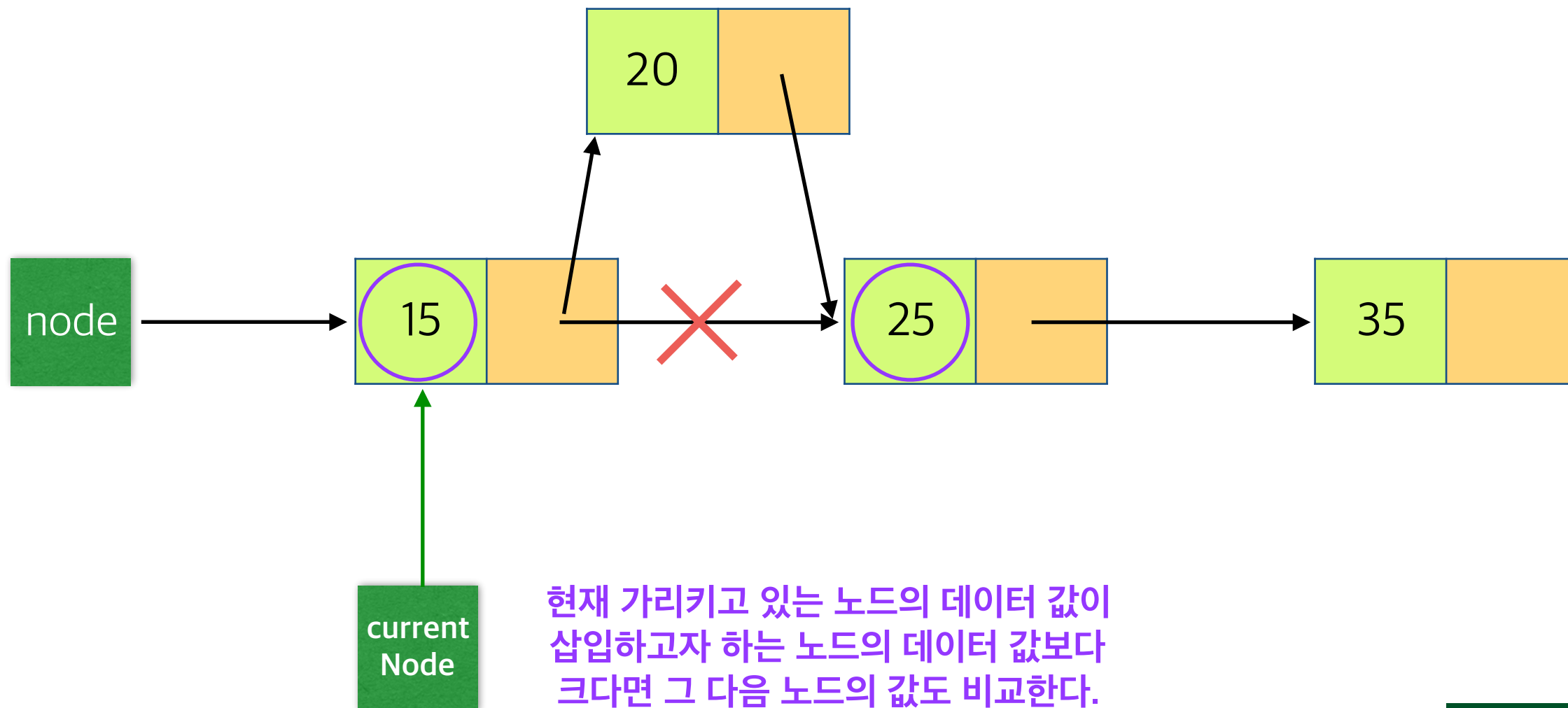
- 연결 리스트(Linked List)
 - 노드 삽입 예제 결과 (오름차순 삽입)



```
C:\Windows\system32\cmd.exe
5
15
25
35
55
계속하려면 아무 키나 누르십시오
```

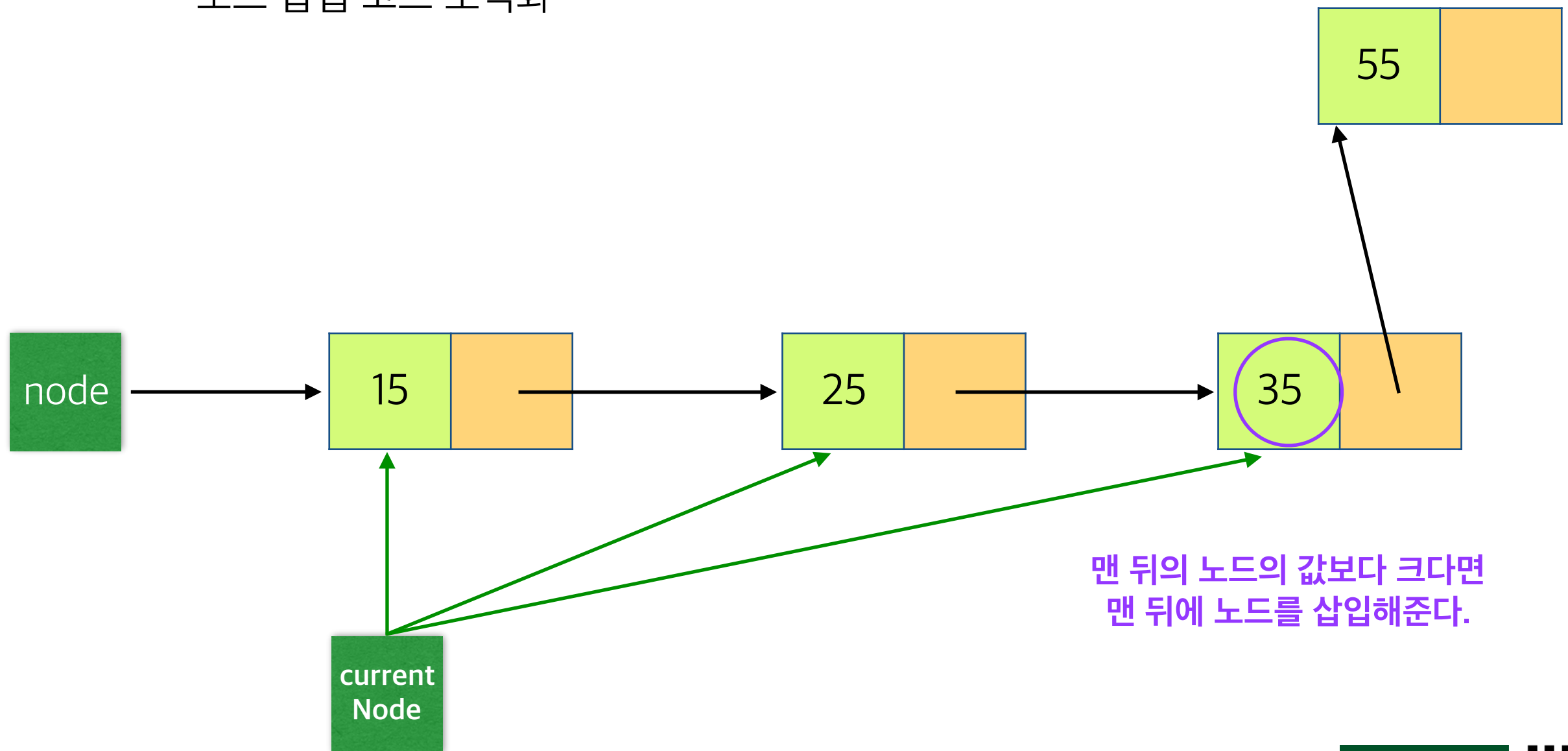
연결 리스트

- 연결 리스트(Linked List)
 - 노드 삽입 코드 도식화



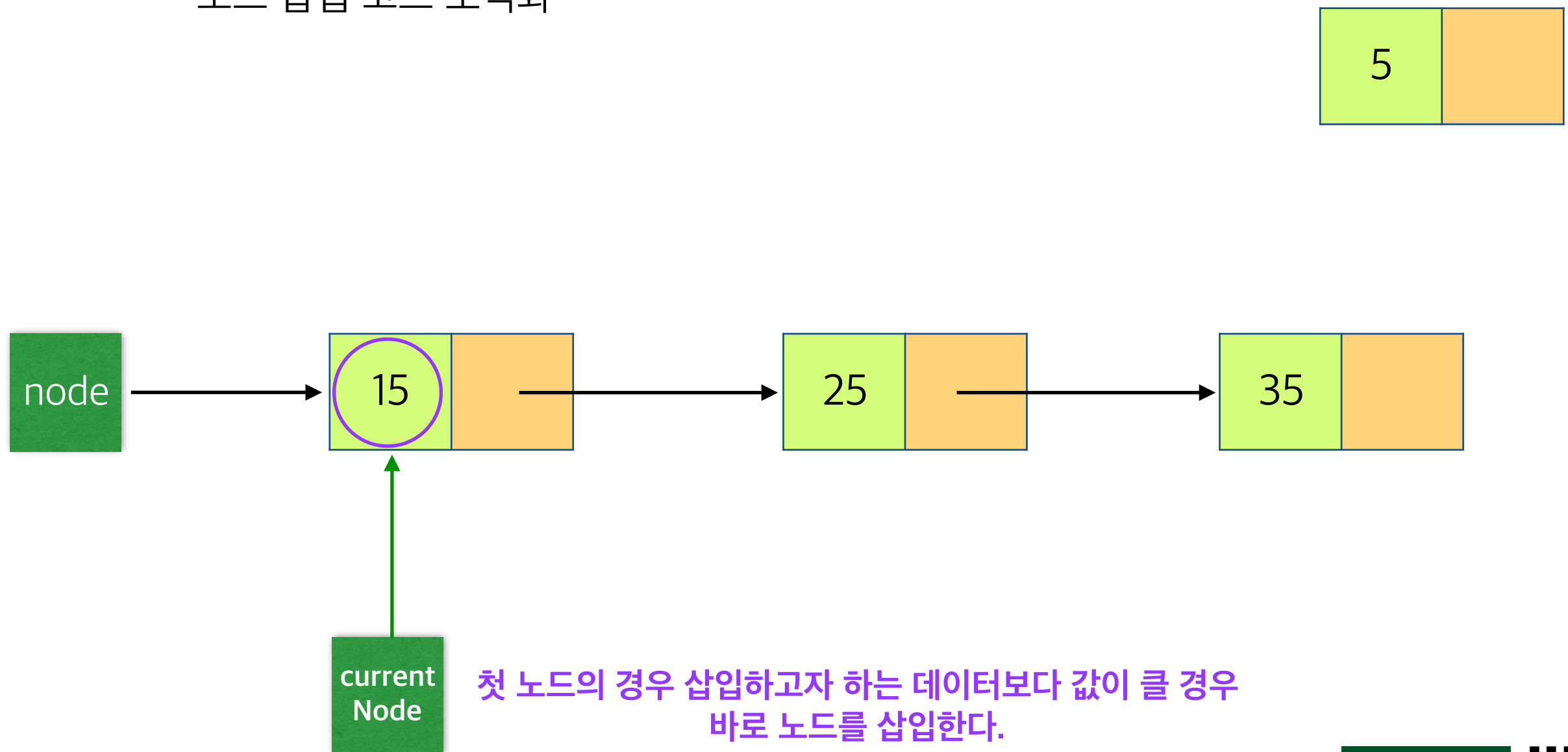
연결 리스트

- 연결 리스트(Linked List)
 - 노드 삽입 코드 도식화



연결 리스트

- 연결 리스트(Linked List)
 - 노드 삽입 코드 도식화



정렬 알고리즘

- 정렬(Sort)이란?
 - 데이터를 하나의 기준을 기반으로 순차적으로 정리해주는 과정
 - 예) 이름(기준) 오름차순, 숫자(기준) 오름차순
- 정렬을 왜 할까?
 - 빠른 탐색을 위해!!!!
 - 예) 1 45 67 25 22 99 1211 5 3 77 3 8 22 34 88 677 21 0 (정렬 X)
 - 가장 작은 수를 찾기 위해서는 처음부터 끝까지 전체적으로 훑어야 함
 - 만약 정렬이 되어 있었다면 맨 앞의 데이터를 쏙
 - 혹은 이진 탐색으로 찾고자 하는 데이터를 잘 찾을 수 있게 됨

정렬 알고리즘

- 오름차순으로 정렬되어 있는 데이터에서 이진 탐색으로 찾고자 하는 데이터를 빠르게 찾을 수 있음
 - 이진 탐색이란?
 - 중간의 값을 임의의 값 선택 후 찾고자 하는 값과 비교하면서 찾는 방식
 - 찾고자하는 값이 10일때

10보다 큰가?

1.

0	1	2	2	5	7	9	10	13	15	19	20	30	58	69	70	99	109	208	222	234	239	250
---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----

10보다 큰가?

2.

0	1	2	2	5	7	9	10	13	15	19
---	---	---	---	---	---	---	----	----	----	----

10보다 큰가?

3.

9	10	13	15	19
---	----	----	----	----

10보다 큰가?

4.

9	10
---	----

정렬 알고리즘

- 정렬하는 알고리즘도 여러가지가 존재함
 - 선택 정렬(Selection Sort)
 - 삽입 정렬(Insertion Sort)
 - 버블 정렬(Bubble Sort)
 - 퀵 정렬(Quick Sort)
 - 합병 정렬(Merge Sort)
 - 힙 정렬(Heap Sort)
 - 기수 정렬(Radix Sort)
 - 등

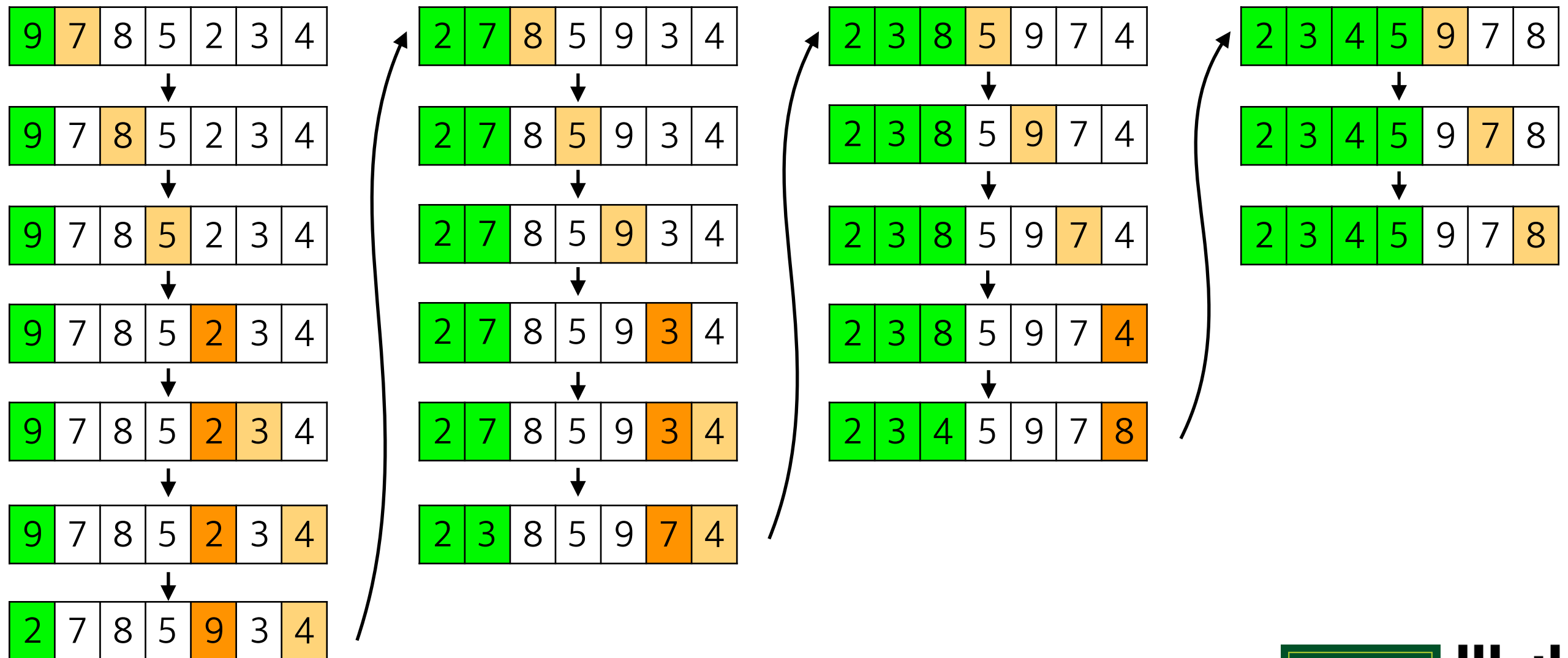
정렬 알고리즘

- 선택 정렬(Selection Sort)
 - 알고리즘 설명
 - 주어진 배열 중에 최솟값을 찾음
 - 그 값을 맨 앞에 위치한 값과 교체
 - 맨 처음 위치를 뺀 나머지 배열을 같은 방법으로 교체

정렬 알고리즘

- 선택 정렬(Selection Sort)

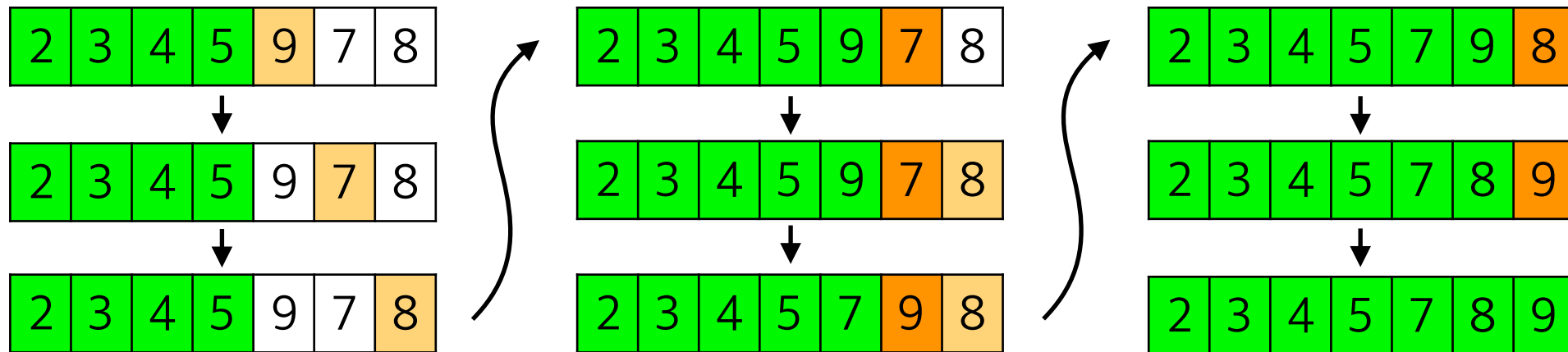
- 도식화



정렬 알고리즘

- 선택 정렬(Selection Sort)

- 도식화



정렬 끝

정렬

```
#include <stdio.h>
#define SIZE 9

void selectionSort(int *arr, const int n);
void printArray(int *arr, const int n);
int main(){
    int arr[SIZE] = {6, 5, 3, 7, 9, 2, 4, 0, 5};

    selectionSort(arr, SIZE);

    return 0;
}
```

• 선택 정렬(Selection Sort)

• 코드

```
void selectionSort(int * arr, const int n){
    int i;
    int j;
    int indexMin; //최솟값을 갖는 배열 요소의 인덱스 값
    int temp; // 임의 저장 변수

    for(i = 0; i<n-1; i++){
        indexMin = i;
        for(j = i+1; j<n; j++){
            if(arr[j]< arr[indexMin]){
                //주어진 배열값이 초기 설정한 최솟값보다 작은 경우
                //교체가 필요하다.
                indexMin = j;
                temp = arr[indexMin];
                arr[indexMin] = arr[i];
                arr[i] = temp;
                printArray(arr, SIZE);
            }
        }
    }
}

void printArray(int *arr, const int n){
    int i = 0;

    for(i=0; i<n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

정렬 알고리즘

- 선택 정렬(Selection Sort)

- 결과

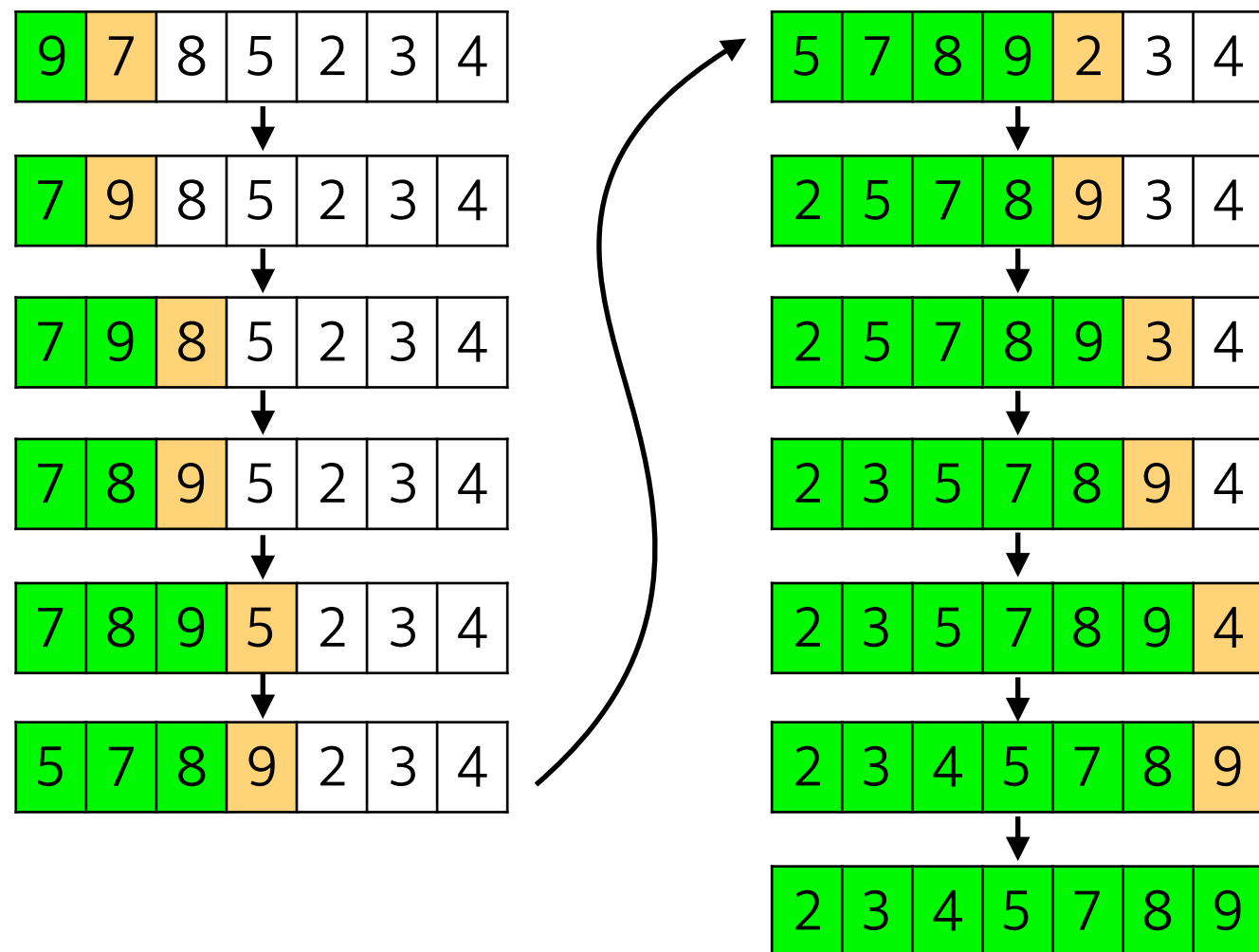
```
C:\Windows\system32\cmd.exe
5 6 3 7 9 2 4 0 5
3 6 5 7 9 2 4 0 5
2 6 5 7 9 3 4 0 5
0 6 5 7 9 3 4 2 5
0 5 6 7 9 3 4 2 5
0 3 6 7 9 5 4 2 5
0 4 6 7 9 5 3 2 5
0 2 6 7 9 5 3 4 5
0 2 5 7 9 6 3 4 5
0 2 3 7 9 6 5 4 5
0 2 4 7 9 6 5 3 5
0 2 4 6 9 7 5 3 5
0 2 4 5 9 7 6 3 5
0 2 4 3 9 7 6 5 5
0 2 4 3 7 9 6 5 5
0 2 4 3 6 9 7 5 5
0 2 4 3 5 9 7 6 5
0 2 4 3 5 9 7 6 5
0 2 4 3 5 7 9 6 5
0 2 4 3 5 6 9 7 5
0 2 4 3 5 5 9 7 6
0 2 4 3 5 5 7 9 6
0 2 4 3 5 5 6 9 7
0 2 4 3 5 5 6 7 9
계속하려면 아무 키나 누르십시오 . . .
```

정렬 알고리즘

- 삽입 정렬(Insertion Sort)
 - 알고리즘
 - 자료 배열의 모든 요소를 앞에서부터 차례대로 이미 정렬된 배열 부분과 비교
 - 자신의 위치를 찾아 삽입함

정렬 알고리즘

- 삽입 정렬(Insertion Sort)
 - 도식화



정렬 끝

정렬 알고리즘

- 삽입 정렬(Insertion Sort)

- 코드

```
#include <stdio.h>
#define SIZE 9

void insertionSort(int *arr, const int n);
void printArray(int *arr, const int n);
int main(){
    int arr[SIZE] = {6, 5, 3, 7, 9, 2, 4, 0, 5};

    insertionSort(arr, SIZE);

    return 0;
}

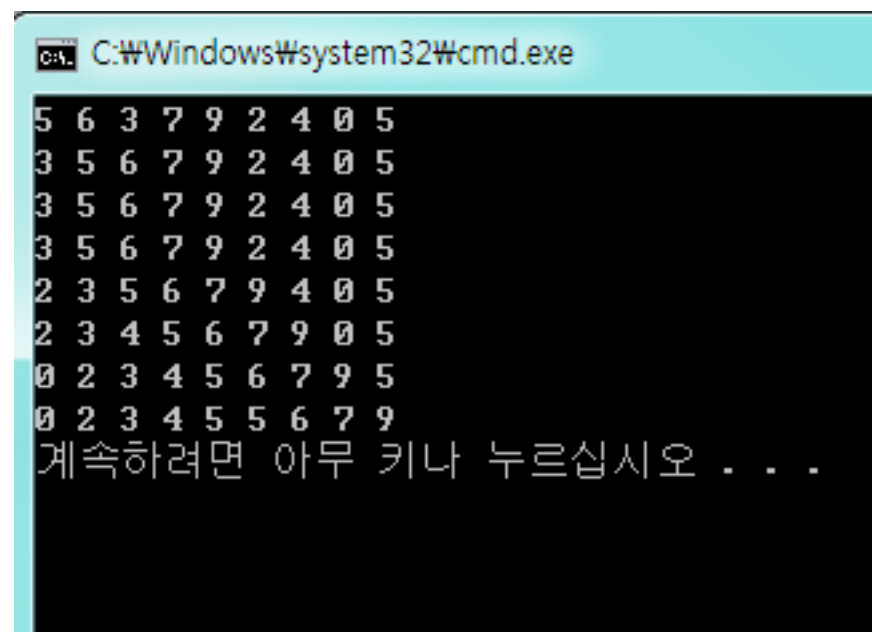
void insertionSort(int *arr, const int n){
    int i, j, temp;
    for(i = 1; i < n; i++){
        j = i;
        temp = arr[i];
        while(--j >= 0 && temp < arr[j]){
            arr[j+1] = arr[j];
        }
        arr[j+1] = temp;
        printArray(arr, SIZE);
    }
}

void printArray(int *arr, const int n){
    int i = 0;

    for(i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

정렬 알고리즘

- 삽입 정렬(Insertion Sort)
 - 결과



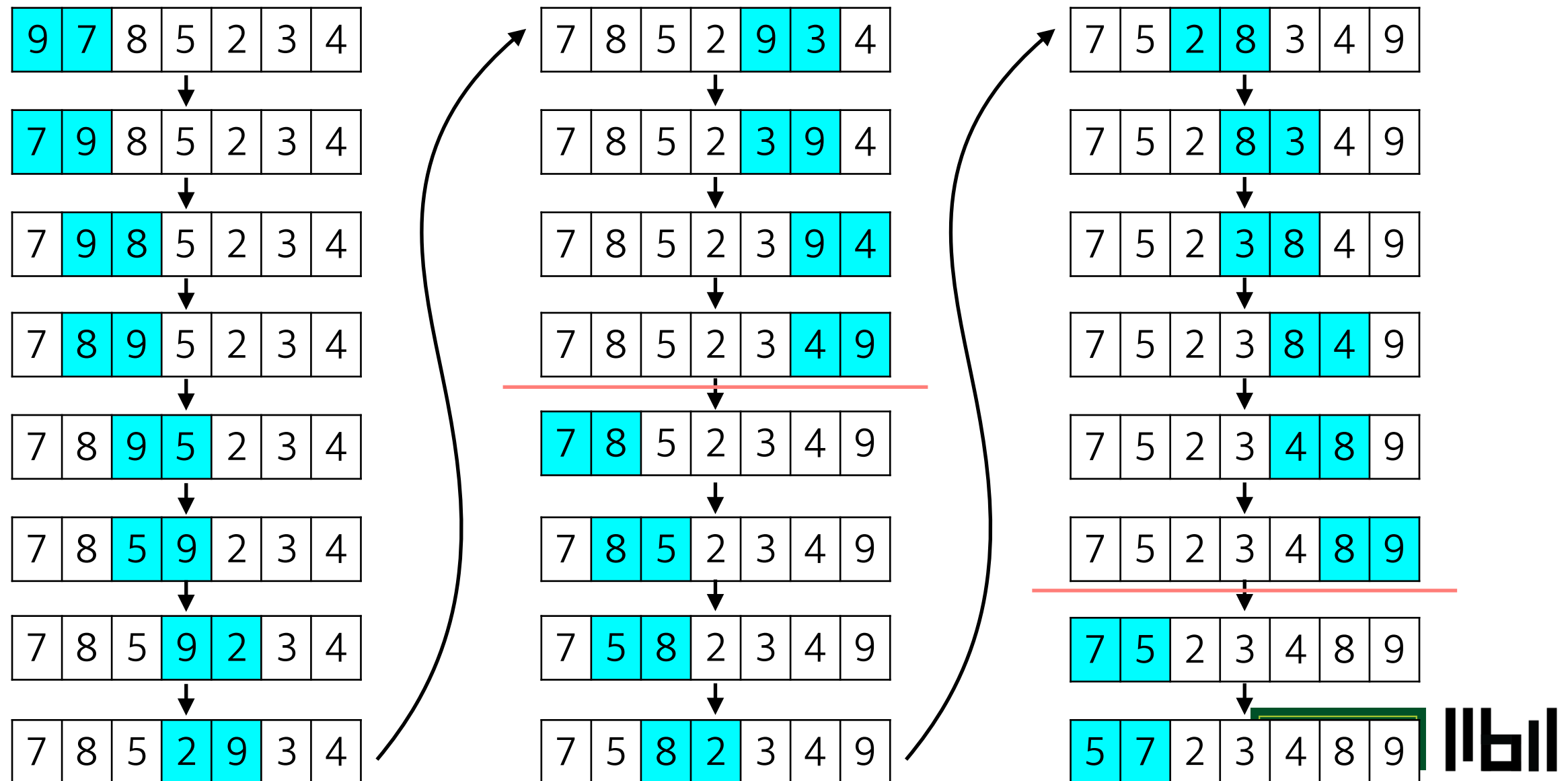
```
C:\Windows\system32\cmd.exe
5 6 3 7 9 2 4 0 5
3 5 6 7 9 2 4 0 5
3 5 6 7 9 2 4 0 5
3 5 6 7 9 2 4 0 5
2 3 5 6 7 9 4 0 5
2 3 4 5 6 7 9 0 5
0 2 3 4 5 6 7 9 5
0 2 3 4 5 5 6 7 9
계속하려면 아무 키나 누르십시오 . . .
```

정렬 알고리즘

- 버블 정렬(Bubble Sort)
 - 알고리즘 설명
 - 배열의 처음에서부터 인접한 요소를 비교
 - 앞에 있는 요소가 바로 뒤에 인접해있는 요소보다 클 경우 교환
 - 최대 배열의 크기 - 1개만큼 스캔하면 정렬 완료

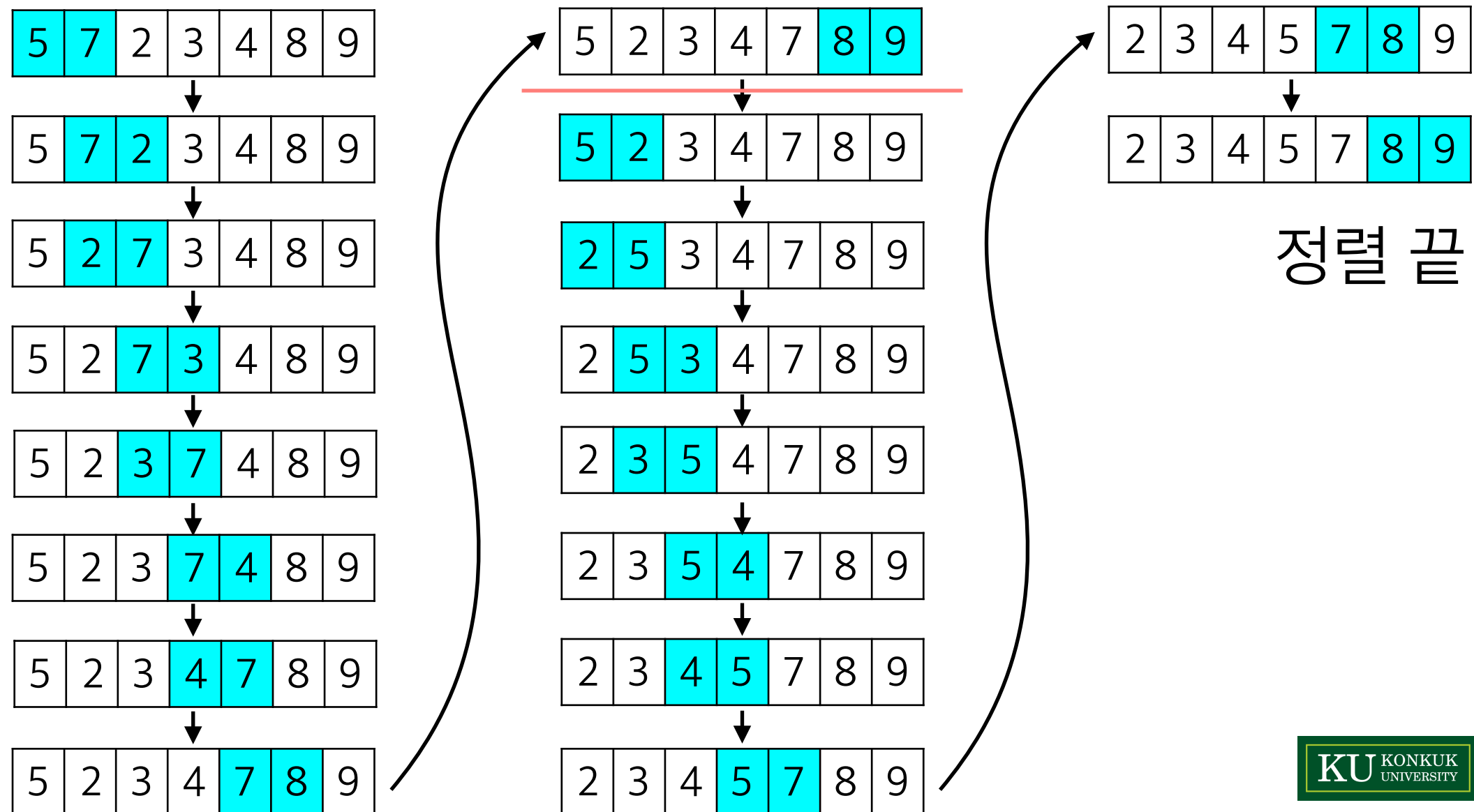
정렬 알고리즘

- 버블 정렬(Bubble Sort)
 - 도식화



정렬 알고리즘

- 버블 정렬(Bubble Sort)
 - 도식화(이어서)



정렬 알고리즘

- 버블 정렬(Bubble Sort)

- 코드

```
#include <stdio.h>
#define SIZE 9

void bubbleSort(int *arr, const int n);
void printArray(int *arr, const int n);
int main(){
    int arr[SIZE] = {6, 5, 3, 7, 9, 2, 4, 0, 5};

    bubbleSort(arr, SIZE);

    return 0;
}

void bubbleSort(int *arr, const int n){
    int temp = 0, loop, i;

    for(loop = 0; loop < n-1; loop++){
        for(i=0; i<n-1; i++){
            if(arr[i] > arr[i+1]){
                temp = arr[i];
                arr[i] = arr[i+1];
                arr[i+1] = temp;
                printArray(arr, SIZE);
            }
        }
    }
}

void printArray(int *arr, const int n){
    int i = 0;

    for(i=0; i<n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

정렬 알고리즘

- 버블 정렬(Bubble Sort)

- 결과

```
C:\Windows\system32\cmd.exe
5 6 3 7 9 2 4 0 5
5 3 6 7 9 2 4 0 5
5 3 6 7 2 9 4 0 5
5 3 6 7 2 4 9 0 5
5 3 6 7 2 4 0 9 5
5 3 6 7 2 4 0 5 9
3 5 6 7 2 4 0 5 9
3 5 6 2 7 4 0 5 9
3 5 6 2 4 7 0 5 9
3 5 6 2 4 0 7 5 9
3 5 6 2 4 0 5 7 9
3 5 2 6 4 0 5 7 9
3 5 2 4 6 0 5 7 9
3 5 2 4 0 6 5 7 9
3 5 2 4 0 5 6 7 9
3 2 5 4 0 5 6 7 9
3 2 4 5 0 5 6 7 9
3 2 4 0 5 5 6 7 9
2 3 4 0 5 5 6 7 9
2 3 0 4 5 5 6 7 9
2 0 3 4 5 5 6 7 9
0 2 3 4 5 5 6 7 9
계속하려면 아무 키나 누르십시오 . . .
```