Fast campus

컴퓨터공학 All in One

C/C++ 문법, 자료구조 및 심화 프로젝트 (나동빈)
제 20강 - 연결 리스트

학습 목표

연결 리스트

1) 연결 리스트의 필요성과 그 쓰임새에 대해서 학습합니다.
2) C언어를 활용하여 연결 리스트를 구현하는 방법에 대해서 공부합니다.

연결 리스트

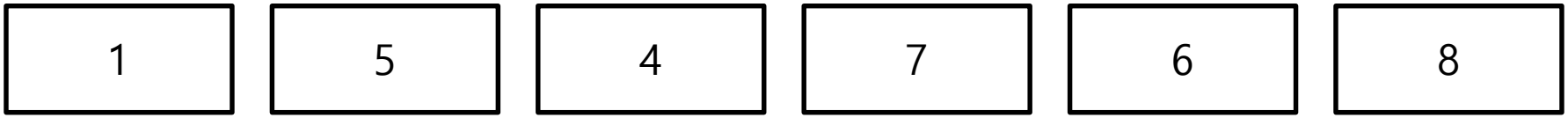
연결 리스트의 필요성

1) 일반적으로 배열을 사용하여 데이터를 순차적으로 저장하고, 나열할 수 있습니다.
2) 배열을 사용하는 경우 메모리 공간이 불필요하게 낭비 될 수 있습니다.
3) 실습을 통해 이를 바르게 이해하는 시간을 가져보도록 하겠습니다.

연결 리스트

배열 기반의 리스트

1) 데이터를 순차적으로 저장하고, 처리할 때는 배열 기반의 리스트를 간단히 이용할 수 있습니다.



Fast campus

연결 리스트

배열 기반의 리스트

```c
#include <stdio.h>
#define INF 10000

int arr[INF];
int count = 0;

void addBack(int data) {
  arr[count] = data;
  count++;
}

void addFirst(int data) {
  for (int i = count; i >= 1; i--) {
    arr[i] = arr[i - 1];
  }
  arr[0] = data;
  count++;
}
```

```c
void show() {
  for (int i = 0; i < count; i++) {
    printf("%d ", arr[i]);
  }
}

int main(void) {
  addFirst(4);
  addFirst(5);
  addFirst(1);
  addBack(7);
  addBack(6);
  addBack(8);
  show();
  system("pause");
  return 0;
}
```

연결 리스트

배열 기반의 리스트

[생각할 거리] 특정한 위치의 원소를 삭제하는 함수는 어떻게 만들 수 있을까요?

```c
#include <stdio.h>
#define INF 10000

int arr[INF];
int count = 0;

void addBack(int data) {
  arr[count] = data;
  count++;
}

void addFirst(int data) {
  for (int i = count; i >= 1; i--) {
    arr[i] = arr[i - 1];
  }
  arr[0] = data;
  count++;
}
```

```c
void show() {
  for (int i = 0; i < count; i++) {
    printf("%d ", arr[i]);
  }
}

int main(void) {
  addFirst(4);
  addFirst(5);
  addFirst(1);
  addBack(7);
  addBack(6);
  addBack(8);
  show();
  system("pause");
  return 0;
}
```

연결 리스트

배열 기반의 리스트

1) 특정한 위치의 원소를 삭제하는 removeAt() 함수는 다음과 같이 구현할 수 있습니다.

```
void removeAt(int index) {
  for (int i = index; i < count - 1; i++) {
    arr[i] = arr[i + 1];
  }
  count--;
}
```

연결 리스트

배열 기반 리스트의 특징

1) 배열로 만들었으므로 특정한 위치의 원소에 즉시 접근할 수 있다는 장점이 있습니다.
2) 데이터가 들어갈 공간을 미리 메모리에 할당해야 한다는 단점이 있습니다.
3) 원하는 위치로의 삽입이나 삭제가 비효율적입니다.

연결 리스트

연결 리스트

1) 일반적으로 연결 리스트는 구조체와 포인터를 함께 사용하여 구현합니다.
2) 연결 리스트는 리스트의 중간 지점에 노드를 추가하거나 삭제할 수 있어야 합니다.
3) 필요할 때마다 메모리 공간을 할당 받습니다.

연결 리스트

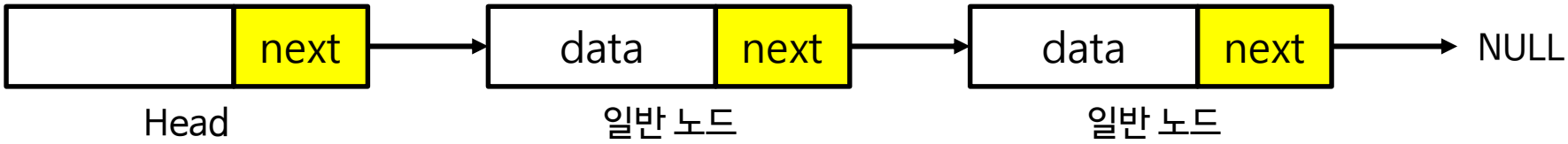
연결 리스트

1) 단일 연결 리스트는 다음과 같은 형태로 나타낼 수 있습니다.
2) 포인터를 이용해 단방향적으로 다음 노드를 가리킵니다.

연결 리스트

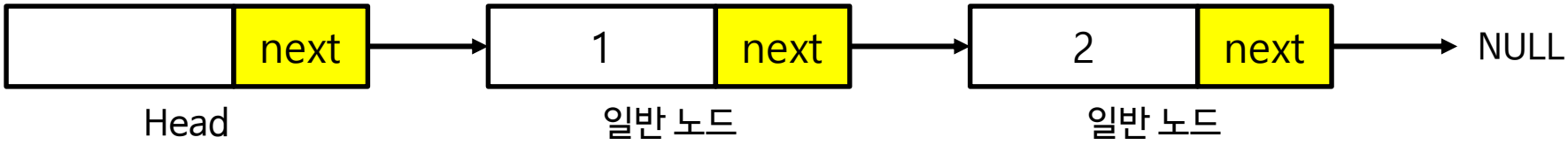
연결 리스트

1) 단일 연결 리스트는 다음과 같은 형태로 나타낼 수 있습니다.
2) 포인터를 이용해 단방향적으로 다음 노드를 가리킵니다.
3) 일반적으로 연결 리스트의 시작 노드를 헤드(Head)라고 하며 별도로 관리합니다.
4) 다음 노드가 없는 끝 노드의 다음 위치 값으로는 NULL을 넣습니다.



연결 리스트

연결 리스트 구조체 만들기



연결 리스트

연결 리스트 구조체 만들기

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

typedef struct {
  int data;
  struct Node *next;
} Node;

Node *head;
```

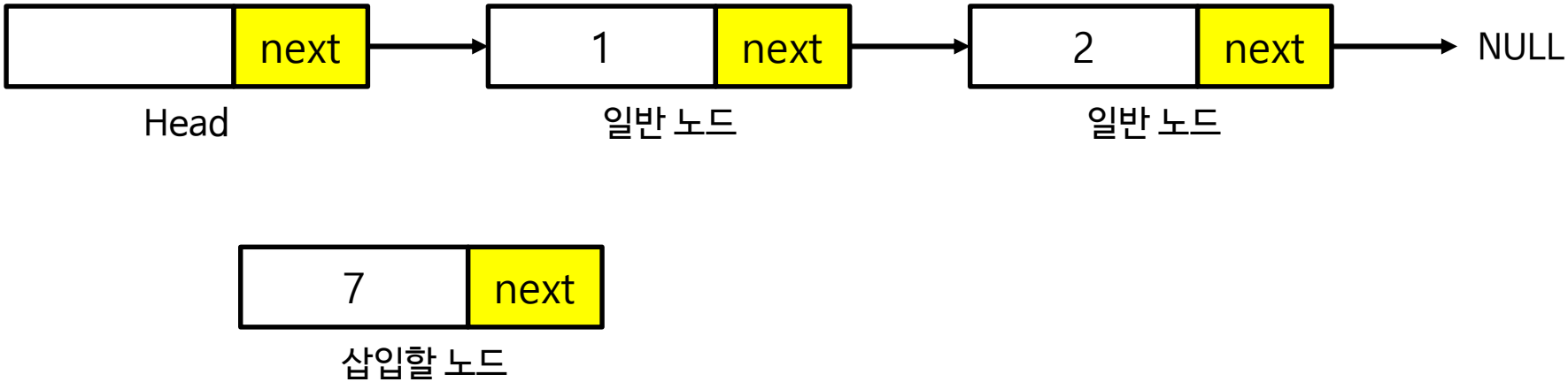
연결 리스트

연결 리스트 구조체 사용해보기

```
int main(void) {
  head = (Node*) malloc(sizeof(Node));
  Node *node1 = (Node*) malloc(sizeof(Node));
  node1->data = 1;
  Node *node2 = (Node*) malloc(sizeof(Node));
  node2->data = 2;
  head->next = node1;
  node1->next = node2;
  node2->next = NULL;
  Node *cur = head->next;
  while (cur != NULL) {
  printf("%d ", cur->data);
    cur = cur->next;
  }
  system("pause");
  return 0;
}
```

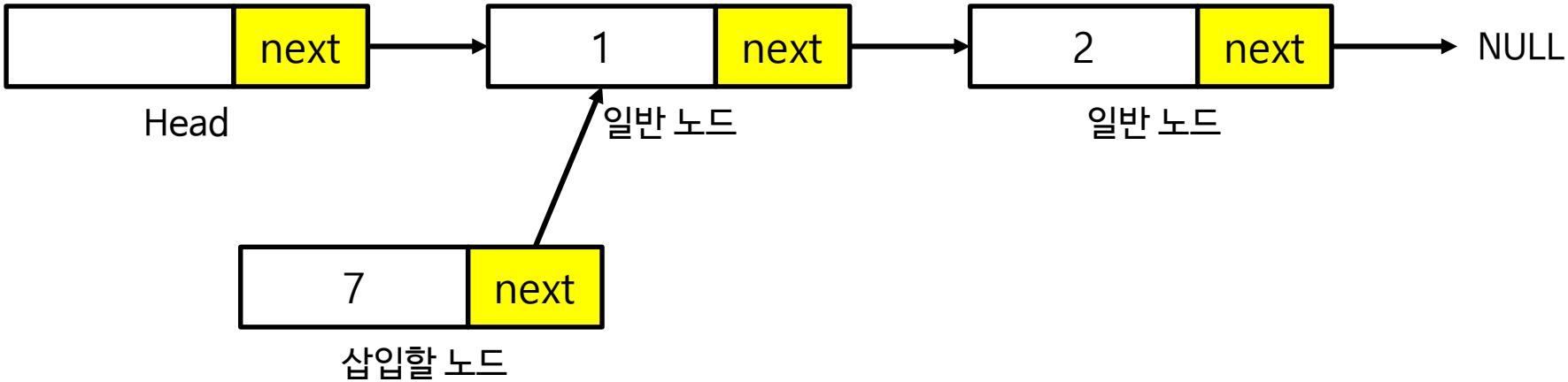
연결 리스트

연결 리스트 삽입 과정 1



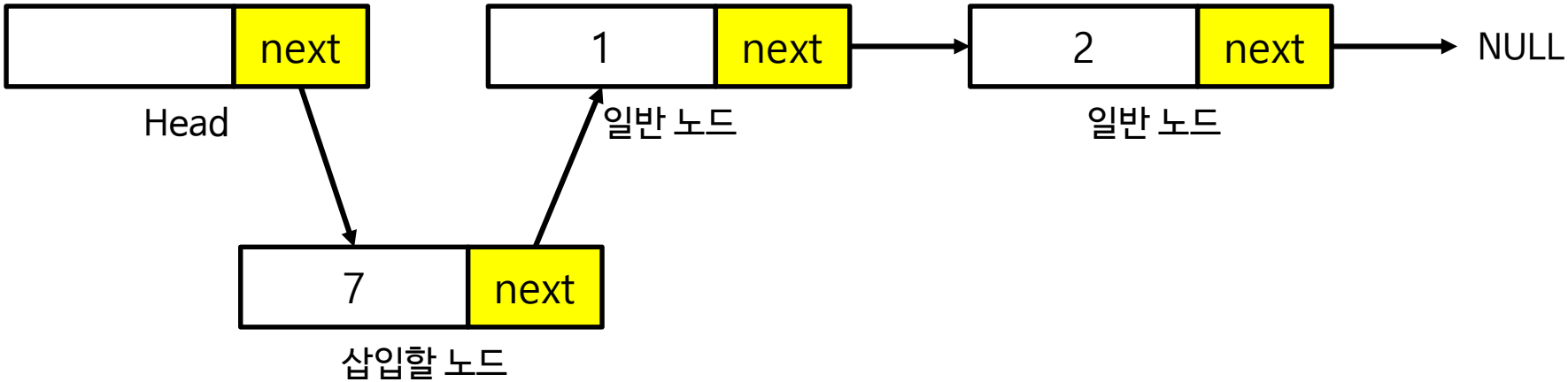
연결 리스트

연결 리스트 삽입 과정 2



연결 리스트

연결 리스트 삽입 과정 3



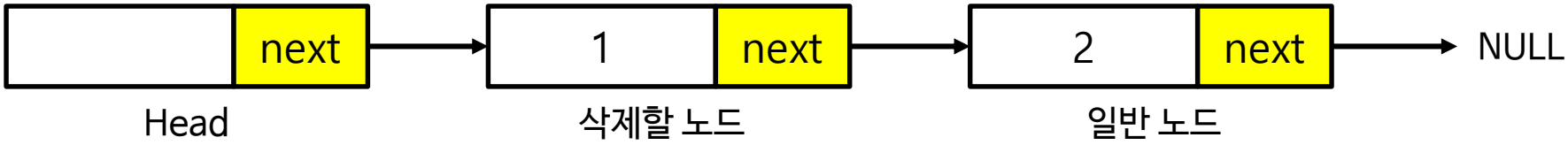
연결 리스트

연결 리스트 삽입 함수

```
void addFront(Node *root, int data) {
  Node *node = (Node*) malloc(sizeof(Node));
  node->data = data;
  node->next = root->next;
  root->next = node;
}
```

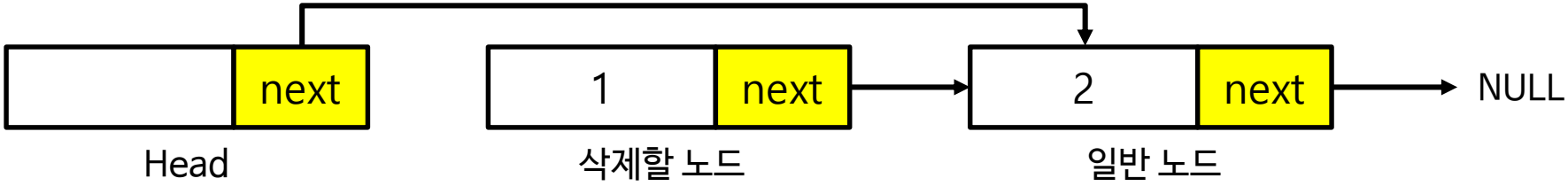
연결 리스트

연결 리스트 삭제 과정 1



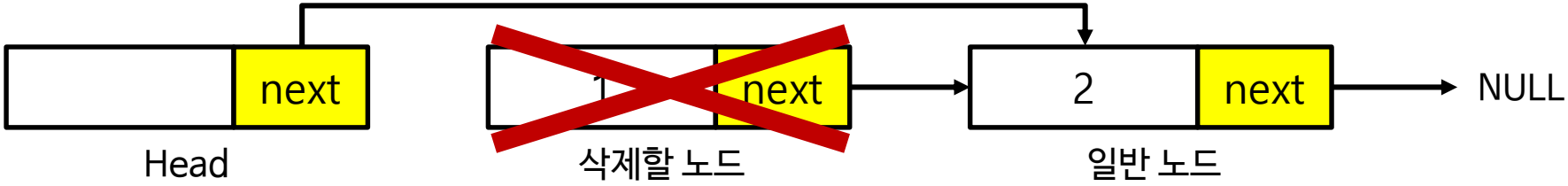
연결 리스트

연결 리스트 삭제 과정 2



연결 리스트

연결 리스트 삭제 과정 3



Fast campus

연결 리스트

연결 리스트 삭제 함수

```
void removeFront(Node *root) {
  Node *front = root->next;
  root->next = front->next;
  free(front);
}
```

연결 리스트

연결 리스트 메모리 해제 함수

```
void freeAll(Node *root) {
  Node *cur = head->next;
  while (cur != NULL) {
    Node *next = cur->next;
    free(cur);
    cur = next;
  }
}
```

Fast campus

연결 리스트

연결 리스트 전체 출력 함수

```
void showAll(Node *root) {
  Node *cur = head->next;
  while (cur != NULL) {
    printf("%d ", cur->data);
    cur = cur->next;
  }
}
```

연결 리스트

완성된 연결 리스트 사용하기

```
int main(void) {
  head = (Node*) malloc(sizeof(Node));
  head->next = NULL;
  addFront(head, 2);
  addFront(head, 1);
  addFront(head, 7);
  addFront(head, 9);
  addFront(head, 8);
  removeFront(head);
  showAll(head);
  freeAll(head);
  system("pause");
  return 0;
}
```

연결 리스트

연결 리스트 구현에 있어서 주의할 점

1) 위 소스코드에 덧붙여서 삽입 및 삭제 기능에서의 예외 사항을 처리할 필요가 있습니다.
2) 삭제할 원소가 없는데 삭제하는 경우, 머리(Head) 노드 자체를 잘못 넣은 경우 등을 체크해야 합니다.

연결 리스트

연결 리스트의 특징

1) 삽입과 삭제가 배열에 비해서 간단하다는 장점이 있습니다.
2) 배열과 다르게 특정 인덱스로 즉시 접근하지 못하며, 원소를 차례대로 검색해야 합니다.
3) 추가적인 포인터 변수가 사용되므로 메모리 공간이 낭비됩니다.

배운 내용 정리하기

연결 리스트

1) 연결 리스트는 데이터를 선형적으로 저장하고 처리하는 한 방법입니다.
2) 기존에 배열을 이용했을 때보다 삽입과 삭제가 많은 경우에서 효율적입니다.
3) 다만 특정한 인덱스에 바로 참조해야 할 때가 많다면 배열을 이용하는 것이 효율적입니다.