2주차 발표_민서연

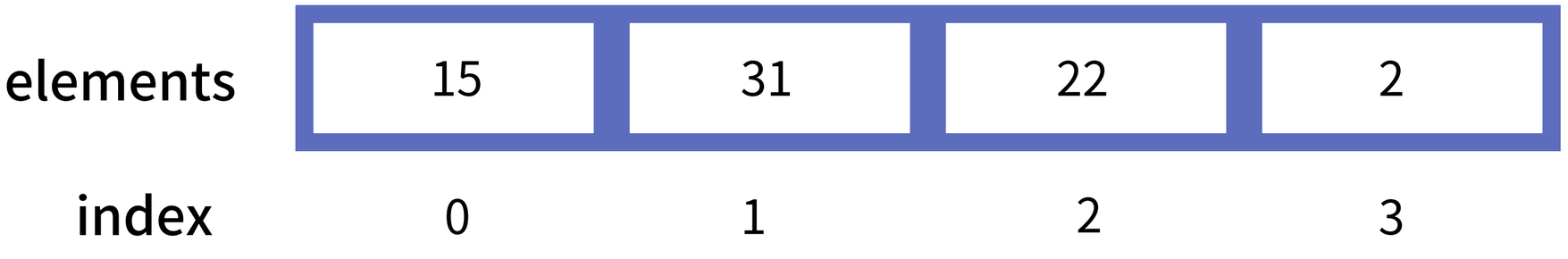
자료구조
Array와 Linked List

출처: https://code-lab1.tistory.com/2, https://yoongrammer.tistory.com/43

01 Array 배열

배열이란,

연속된 메모리 공간에 순차적으로 저장된 데이터 모음



elements
: 배열을 구성하는 각각의 값

index
: 배열에서의 위치

01 Array 배열

장점

- 인덱스를 이용한 접근이 가능하기 때문에 모든 요소에 빠르게 접근 가능
- 기록 밀도가 1이기 때문에 공간 낭비가 적음
- 간단하고 사용 쉬움

단점

- 할당된 정적 메모리에 크기를 변경할 수 없음
- 데이터를 삭제나 삽입할 경우 모든 요소들을 이동 시켜야하기 때문에 비용이 많이 듦
- 배열의 크기를 미리 지정하기가 어려움

01 Array 배열

선언

```java
int arr[] = new int[10];
```

특징

- 같은 타입의 데이터를 여러개 나열한 선형 자료구조
- 연속적인 메모리 공간에 순차적으로 데이터를 저장
- 배열은 선언할 때 크기를 지정하는 순간 그 크기로 고정
- 시간 복잡도는 O(1)로 모두 동일
- index가 존재하기 때문에 indexing, slicing 이 가능

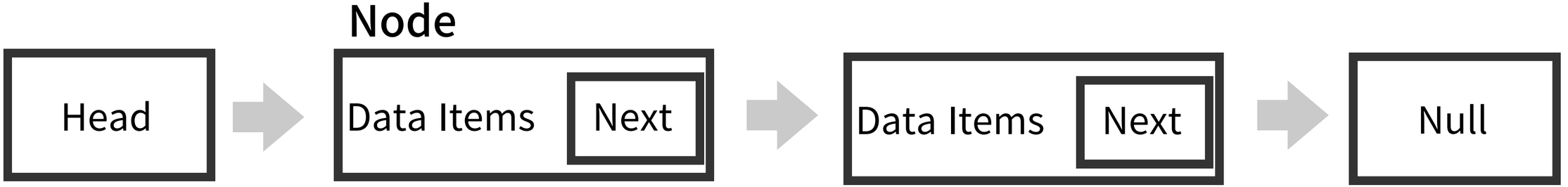
02 Linked List

연결리스트란,

데이터와 포인터로 구성된 노드간의 연결을 이용해서 리스트를 구현한 구조
배열과는 달리 연속적인 저장공간에 저장된 것이 아니기 때문에 노드간의 연결 필요

Head
: 리스트의 처음을 나타냄

Node
: 데이터와 다음 노드를 가리키는 next 포인터로 구성



02 Linked List

초기화(init)

```java
Node* head;

void init(){
  head = NULL;
}
```

삽입(insert)

```java
// 가장 앞에 노드 삽입
void insert(int data){
  Node* ptr;
  Node* newNode = (Node*)malloc(sizeof(Node)); // newNode 할당

  newNode->data = data; // data 할당
  newNode->next = NULL; // next 포인터 초기화

  if(head == NULL) { // empty, 연결 리스트가 비어있을 경우
    head = newNode;
  }else{
    if(head->data > newNode->data){ // not empty, 가장 앞에 노드 추가
      newNode->next = head; // newNode의 next 포인터에 head 노드 가리키게 함
      head = newNode; // head가 newNode 를 가리키게 함
      return;
    }
    for(ptr = head; ptr->next; ptr = ptr->next){ // 중간에 노드 추가
      if(ptr->data < newNode->data && ptr->next->data > newNode->data){
        newNode->next = ptr->next; // 이전 노드가 가리킨 노드를 새노드에 가리킴
        ptr->next = newNode; // 이전 노드 포인터에 새 노드 가리킴
        return;
      }
    }
    ptr->next = newNode; // 마지막에 노드 추가
  }
}
```

02 Linked List

삭제(delete)

```java
int deleteNode(int data){
  Node *cur, *prev;
  cur = prev = head; // cur 노드와 prev 노드를 맨 앞 노드에 설정

  if(head == NULL){ // empty List
    printf("error: list is empty\n");
    return -1;
  }

  if(head->data == data){ // 가장 앞의 노드
    head = cur->next; // head에 삭제할 노드의 포인터 값을 넣음
    cur->next = NULL; // 삭제할 노드 포인터에 null 값을 넣음
    free(cur);
    return 1;
  }
  for(; cur; cur = cur->next){ // 중간 혹은 마지막 노드 삭제
    if(cur->data == data){ // 삭제할 데이터와 일치 할 경우
      prev->next = cur->next; // 삭제할 노드의 앞 노드 포인터에 삭제할 노드의 포인터 값 넣음
      cur->next = NULL; // 삭제할 노드의 포인터 값에는 nulll 값을 넣음
      free(cur);
      return 1;
    }
    prev = cur;
  }

  printf("error : there is no %d!\n", data");
  return -1;
}
```

탐색(search)

```java
int search(int data){
  Node* ptr;
  for(ptr = head; ptr; ptr = ptr->head){
    if(ptr->data = data){ // data 발견
      return 1;
    }
  }
}
```

02 Linked List

특징

- 노드의 next 부분에 다음 노드 위치를 데이터 탐색하기 위해 첫 노드부터 순차적으로 탐색 시작해서 순차 접근만 가능
- 주소만 연결하면 되기 때문에 삽입 삭제가 비교적 빠름
- 불연속적 단위로 저장되기 때문에 조회에 불리하면 포인터로 인해 저장공간 필요

02 Linked List

장점

- 크기가 가변적임
- 삽입 삭제가 쉬움

단점

- 요소에 접근하려면 첫번째 노드부터 순차적으로 접근
- 포인터를 위한 추가 메모리 필요

03 마무리

Array

- 순차적인 데이터를 저장하며 값보다는 순서가 중요할 때
- 다차원 데이터를 다룰 때
- 어딴 특정 요소를 빠르게 읽어야 할 때
- 데이터 사이즈가 자주 바뀌지 않으며 요소가 자주 추가되거나 삭제 되지 않을 때

Linked List

- 정해진 크기가 아닌 동적으로 배열의 사이즈를 늘려야 할 때
- 어떤 요소들을 삽입하고 삭제해야 할 때