Data Structure #3

Linked List

2020년 1학기

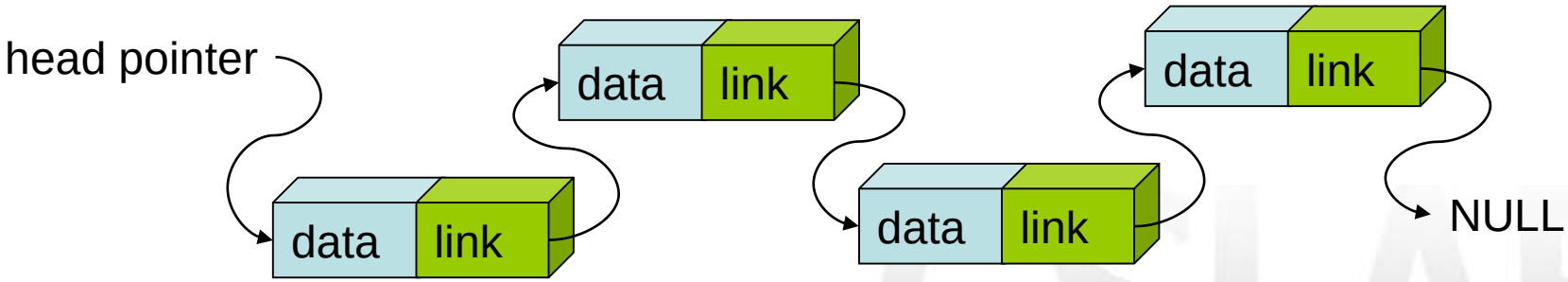
Intro.

- **실습주제 소개**
  - Linked List: insert
  - Linked List : remove
  - Linked List(참고) : create, display, search

- **실습수업 문제**
  - Linked List : insert
  - Linked List : remove

연결 리스트(Linked List)

- **연결 리스트란?**
  - 노드가 하나의 링크 필드에 의해서 다음 노드와 연결되는 구조
  - 입력이 시작되면 노드가 동적으로 생성되어 추가
  - 일반 구조체에 자기 참조 구조체를 추가



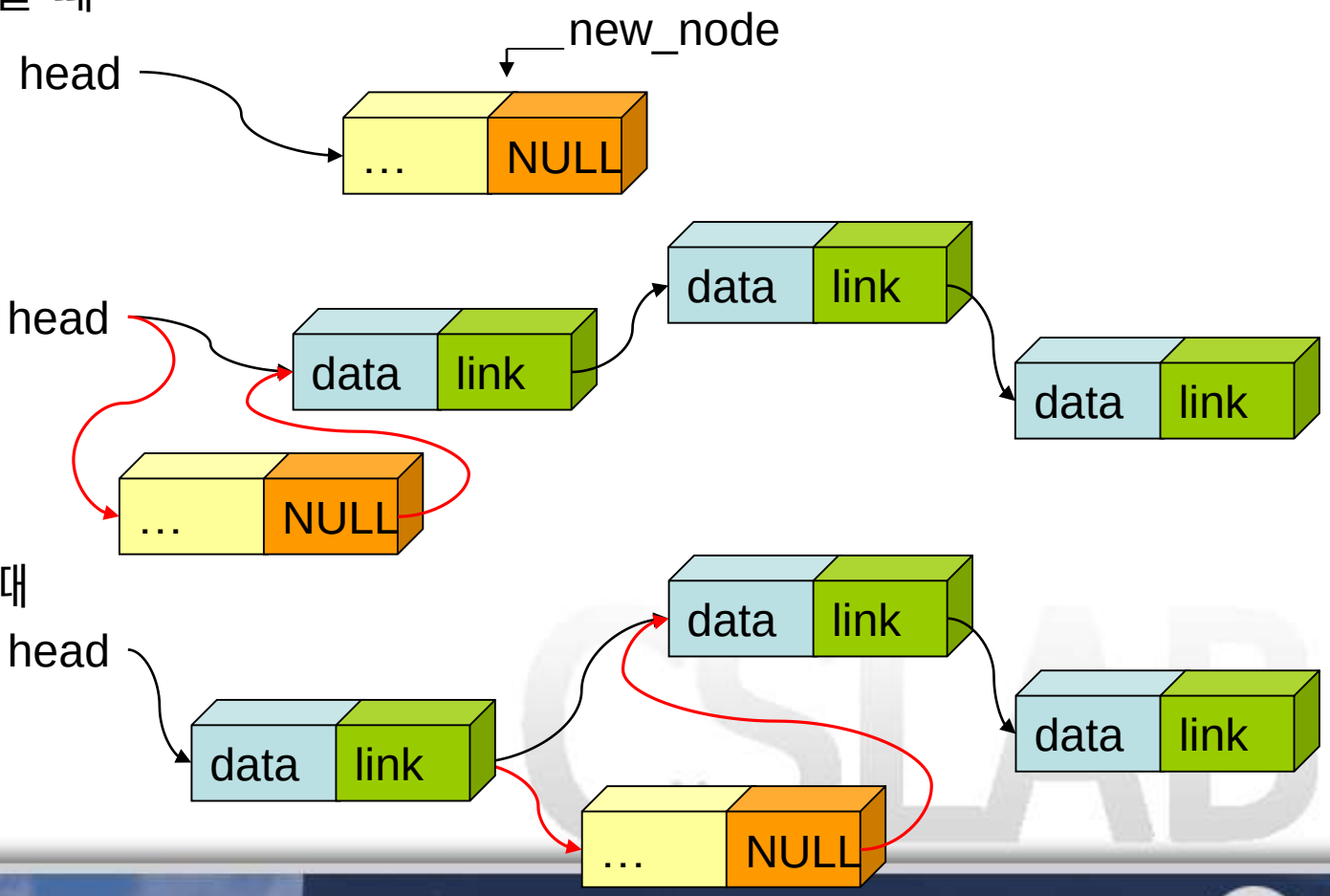
연결 리스트(Linked List)

– 연결 리스트의 구조체

```
typedef int element;
typedef struct ListNode {
        element data;
        struct ListNode *link;
} ListNode;
```

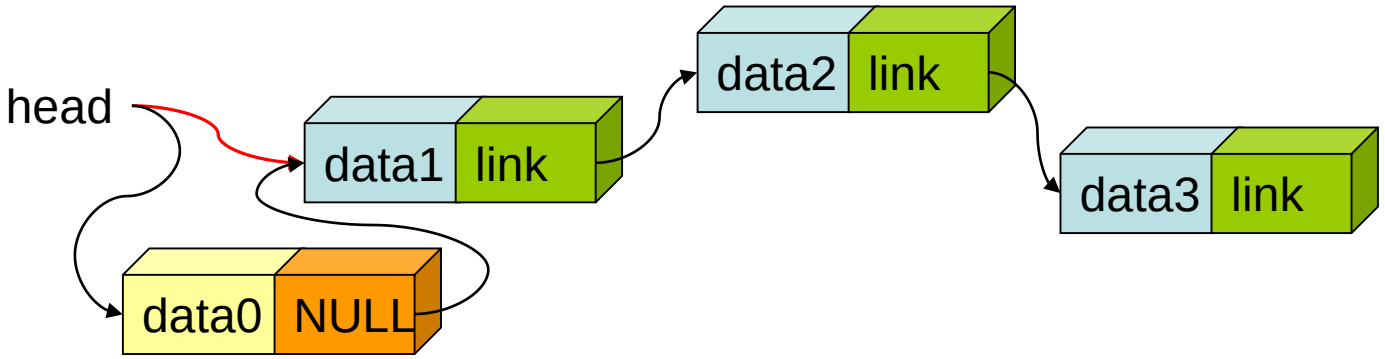
연결 리스트(Linked List)

- **삽입 알고리즘**
  - 헤드가 NULL일 때
  - 첫 노드일 때
  - 중간 노드일 때

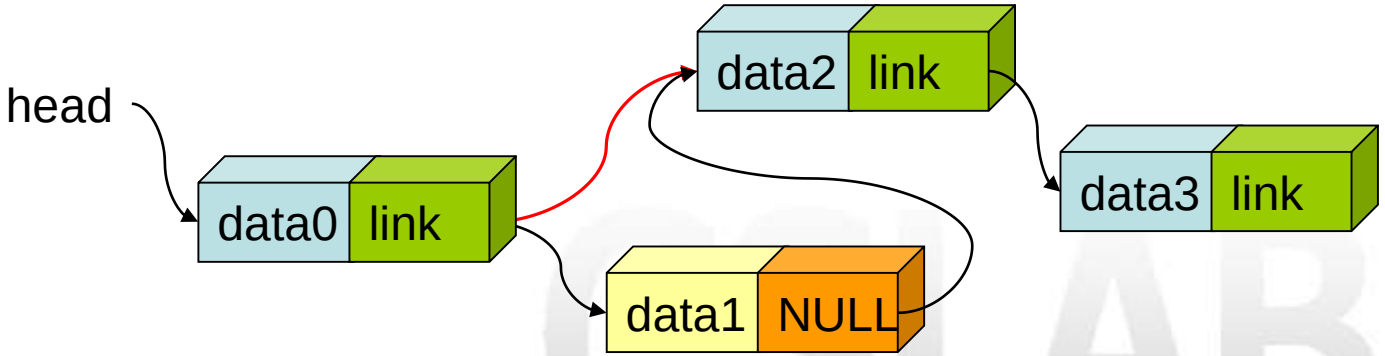


연결 리스트(Linked List)

- **삭제 알고리즘**
  - 데이터0 삭제



  - 데이터1 삭제



연결 리스트(Linked List)

- **연결 리스트의 구현 시 필요한 요소**
  - 새로운 노드 생성
  - 삽입 연산
  - 삭제 연산
  - 리스트 안의 항목을 표시

연결 리스트(Linked List)

- **연결 리스트의 구현**
  - 연결 리스트의 구조체

```
typedef int element;
typedef struct ListNode {
        element data;
        struct ListNode *link;
} ListNode;
```

  - 에러 함수

```
void error(char *message)
{
        fprintf( stderr, “%s\n”, message );
        exit(1);
}
```

연결 리스트(Linked List)

- **연결 리스트의 구현**
  - main 함수

```
void main()
{
        ListNode *list1 = NULL;

        insert_node( &list1, NULL, create_node(10, NULL) );
        insert_node( &list1, NULL, create_node(20, NULL) );
        insert_node( &list1, NULL, create_node(30, NULL) );
        display( list1 );

        remove_node(&list1, NULL, list1);
        display( list1 );
}
```

연결 리스트(Linked List)

- **연결 리스트의 구현**
  - 노드 생성 함수

```
ListNode *create_node( element data, ListNode *link )
{
        ListNode *new_node;
        new_node = (ListNode *)malloc(sizeof(ListNode));
        if( new_node == NULL ) error("malloc err");
        new_node->data = data;
        new_node->link = link;
        return (new_node);
}
```

연결 리스트(Linked List)

- **연결 리스트의 구현**
  - 삽입 함수

```
void insert_node(ListNode **phead, ListNode *p, ListNode *new_node) {
        if( *phead == NULL ) {
                //List is Empty


        }
        else if( p == NULL ){


        }
        else {


        }
}
```

연결 리스트(Linked List)

- **연결 리스트의 구현**
  - 디스플레이 함수

```
void display( ListNode *head )
{
        ListNode *p = head;
        while( p != NULL ){
                printf("%d->", p->data );
                p = p->link;
        }
        printf( “\n” );
}
```

연결 리스트(Linked List)

- **연결 리스트의 구현**
  - 노드값 탐색

```
ListNode *search( ListNode *head, int x )
{
        ListNode *p;
        p = head;
        while( p != NULL ){
                if( p->data == x ) return p;
                p = p->link;
        }
        return p;
}
```

연결 리스트(Linked List)

- **연결 리스트의 구현**
  - 삭제 함수

```
void remove_node(ListNode **phead, ListNode *p, ListNode *removed) {
        if( p == NULL )

        else


        free(removed);   //free curNode
}
```

연결 리스트(Linked List)

- **제출**
  - 개인 실습
    - 삽입 함수
    - 삭제 함수
    - 오늘 자정까지 제출
  - 과제
    - 연결리스트 구현 (lab3.docx)
    - 다음 주 목요일 자정까지 제출

과제

- **Lab03.docx**
  - Insert (i)
    - 주어진 키를 갖고 있는 노드 뒤에 새로운 노드 생성
    - 입력 받은 키를 갖고 있는 노드가 리스트에 없으면 에러 메세지 출력
  - Delete (d)
    - 주어진 키를 갖고 있는 노드를 삭제
    - 입력 받은 키를 갖고 있는 노드가 리스트에 없으면 에러 메세지 출력
  - Find the previous node (f)
    - 주어진 키를 갖고 있는 노드 앞에 있는 노드를 찾음
    - 입력 받은 키를 갖고 있는 노드가 리스트에 없으면 에러 메세지 출력
  - Show the entire list (p)
    - 리스트에 있는 노드들의 키 값을 출력
    - 리스트가 비어있을 경우, 비어있다는 메세지 출력

과제

- **초기 선언**

```
typedef struct Node *PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;
typedef int ElementType;
struct Node
{
    ElementType   element;
    Position          next;
};
```

과제

- 함수
  - List MakeEmpty(List L);
  - int IsEmpty(List L);
  - int IsLast(Position P, List L);
  - void Delete(ElementType X, List L);
  - Position FindPrevious (ElementType X, List L);
  - Position Find(ElemenType X, List);
  - void Insert (ElementType X, List L, Position P);
  - void DeleteList (List L);

과제

- **함수**
  - List MakeEmpty(List L);

```
//create header node
List MakeEmpty(List L) {
        L = (List)malloc(sizeof(struct Node));
        L->element = -30;
        L->next = NULL;
        return L;
}
```

  - head pointer 초기화에 사용

과제

- **함수**
  - int IsEmpty(List L);

```
int isEmpty(List L) {
    return L->next == NULL;
}
```

  - 리스트가 비어 있는지 확인

과제

- **함수**
  - int IsLast(Position P, List L);

```
int isLast(Position P, List L) {
        Position cur = L;
        while(cur->next != NULL) {
                cur = cur->next;
        }
        return P == cur;
}
```

  - 입력된 노드의 위치가 리스트의 끝에 있는지 확인

과제

- **함수**
  - void DeleteList (List L);

```
void DeleteList(List L) {
    Position P = NULL, Tmp = NULL;
    P = L->next; /* Header assumed */
    L->next = NULL;
    while (P != NULL)
    {
        Tmp = P->next;
        free(P);
        P = Tmp;
    }
}
```

  - 리스트에 있는 모든 노드 삭제 (할당 해제)

과제

- **프로그램**

  - Input 파일 열기 (fopen)
  - Header pointer 초기화 (MakeEmpty)
  - 반복문과 스위치문, 함수들을 활용하여 command 처리 부분 작성
    - (i , d, f, p)

  - Input 파일의 명령들을 모두 실행한 후, 리스트 삭제 (DeleteList)
  - Input 파일 닫기 (fclose)
  - 프로그램 종료