Data Structures

Lab # 06

Data & Knowledge Engineering Laboratory
Kyung Hee University, Korea

Chapter 5 Programming Exercise

Heap

Heap이란?

- 프로그램이 실행될 때까지 알 수 없는 가변적인 양만큼의 데이터를 저장하기 위해 프로그램의 프로세스가 사용할 수 있도록 미리 예약되어 있는 메인 메모리 영역
- 동적 할당을 통하여 Heap영역에 메모리 할당
- 반드시 사용 후 메모리 해제를 해주어야 함 (누수 발생)

Heap 에러가 발생하는 경우

- Heap에 공간이 부족하여 더 이상 메모리를 할당 할 수 없는 경우 (Overflow)
- 이미 메모리를 해제한 경우와 할당되지 않은 메모리를 해제하는 경우 에러 발생

Exercise 1

■ 문 제

❖ ReplaceItem함수를 Linked Structure로 구현된 Stack에 추가하시오.

> Replace Item
> 함수 : 모든 oldItem을 newItem으로 바꾼다.
> 조건 : 스택은 초기화되어 있다.
> 결과 : 스택에 있는 각각의 oldItem은 newItem으로 바뀌진다.

❖ A. ReplaceItem 함수를 클라이언트로 작성한다.

- 프로토타입 : void ReplaceItem(StackType &stack, ItemType oldItem, ItemType newItem)

❖ B. ReplaceItem 함수를 멤버함수로 구현하시오.

- 프로토타입 : void ReplaceItem(ItemType oldItem, ItemType newItem);

❖ source_code₩labplus₩Lab, C++ 3rd₩Chapter5₩stackLL

1-help slide

A. 클라이언트 함수로 작성

```
void ReplaceItem(StackType &stack,
        ItemType oldItem, ItemType newItem)
{
        StackType temp_stack; //백업용 스택
        ItemType temp_Item; // top을 받는 아이템
        while(!stack.IsEmpty())
        {
                //Top,Pop을 사용
                //top의 결과가 oldItem과 같거나 다를 경우, temp_stack에 넣는 부분
                if(temp_Item == oldItem)
                {
                }else{
                }
        }
        while(!temp_stack.IsEmpty())
        {
                temp_stack의 내용을 stack에 복원.
        }
}
```

B. 멤버 함수로 작성

```
void StackType::ReplaceItem(ItemType oldItem, ItemType newItem)
{
        //함수 설명: 링크 구조로 연결되어 있는 노드들을 top 에서부터 시작하여 하나씩 검사해 가며 oldItem 값과 같은 아이템을 newItem으
        로 변경해 준다

        NodeType *location = topPtr;   //현재 노드를 가리기기 위한 포인터 선언 및 초기화

    while (location != NULL) {
        if(location->info == oldItem){
                //데어터를 교체한다.
                //location은 다음 노드를 가리킨다.
        }else
        {
                //location은 다음 노드를 가리킨다.
        }
}
```

Exercise 2

■ 문 제

❖ ReplaceItem함수를 Linked Structure로 구현된 Queue에 추가하시오

> Replace Item
> 함수 : 모든 oldItem을 newItem으로 바꾼다.
> 조건 : 큐는 초기화되어 있다.
> 결과 : 큐에 있는 각각의 oldItem은 newItem으로 바뀌진다.

❖ A. ReplaceItem 함수를 클라이언트로 작성한다.

- 프로토타입(함수 template 사용) : void ReplaceItem(QueType<ItemType> &queue, ItemType oldItem, ItemType newItem)

❖ B. ReplaceItem 함수를 멤버함수로 구현하시오.

- 프로토타입(template) : void ReplaceItem(ItemType oldItem, ItemType newItem);

❖ source_code₩labplus₩Lab, C++ 3rd₩Chapter5₩queLL

2-help slide

A. 클라이언트 함수로 작성

```
//template 클래스 이므로 template 함수를 사용한다. 따로 헤더 파일을 정의하는 것을 주의하라.
template <class ItemType>
void ReplaceItem(QueType<ItemType> &queue,  ItemType oldItem, ItemType newItem)
{
        // 임시로 사용할 큐를 선언.
        // Dequeue를 받을 변수 선언
        while(!queue.IsEmpty())
        {
                //Dequeue
                if(item == oldItem)
                {
                // 상황에 따라 Enqueue 결정
                }else{
                }
        }
        while(!temp_queue.IsEmpty())
        {
        …// queue로 내용을 복원.
        }
        }
```

B. 멤버 함수로 작성

```
temtplate <class ItemType>
void QueType<ItemType>::ReplaceItem(ItemType oldItem, ItemType newItem)
{
    …
  ptr = qFront;
  while (ptr != _____) {
          if(ptr->info == oldItem){ … }
          …
  }
}
```

Exercise 3

■ 문제

❖ 두 개의 연결 리스트를 하나의 연결 리스트로 병합하는 함수를 구현하시오.

- **가정 : 각각의 리스트는 중복 아이템이 존재하지 않음**
- 정렬 연결 리스트는 오름차순으로 정렬되어 있음 Ex) [1]->[2]->[3]...

❖ **A. MergeLists 클라이언트 함수 작성**

- 프로토타입(함수 템플릿) : void MergeLists(SortedType<ItemType> &l_a, SortedType<ItemType> &l_b, SortedType<ItemType> &result)

❖ **B. MergeLists 멤버 함수 작성**

- 프로토타입(탬플릿) : void MergeLists(SortedType<ItemType> &other, SortedType<ItemType> &result);

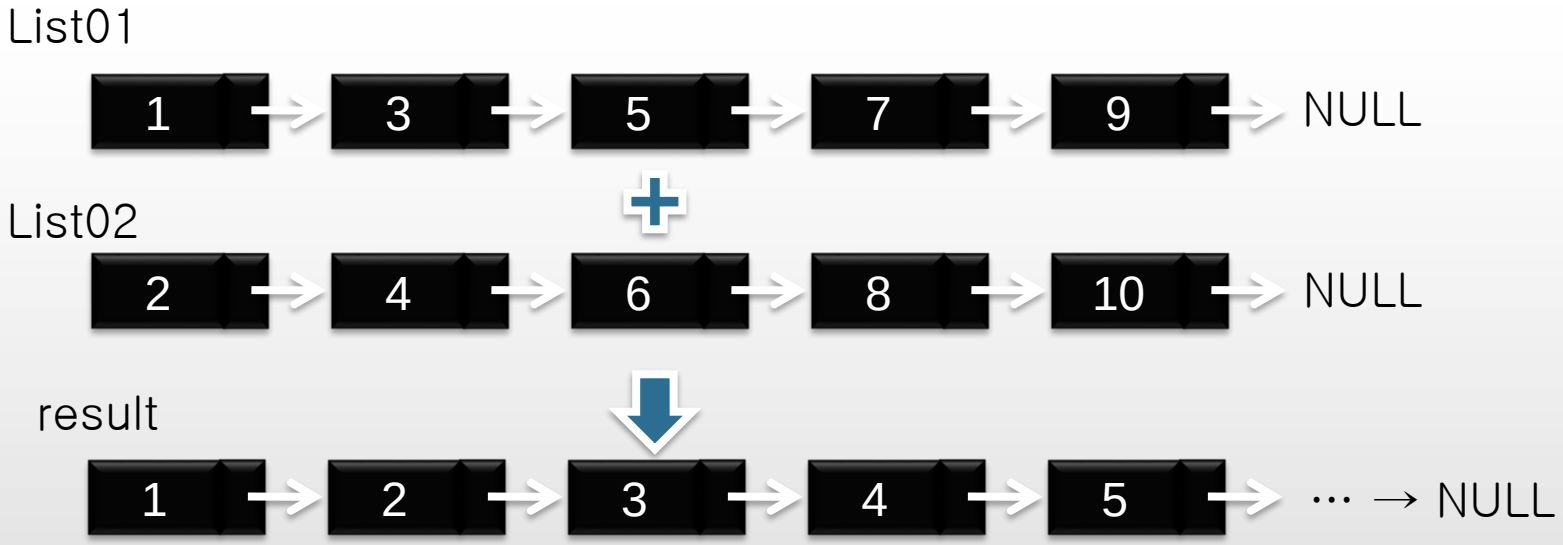
❖ **C. A를 UnsortedType으로 작성**

❖ **D. B를 UnsortedType으로 작성**

❖ source_code₩labplus₩Lab, C++ 3rd₩Chapter5₩ListLL

두 개의 정렬 연결리스트를 합병하는 예제

- 정렬 연결 리스트를 사용하는 경우 (문제 A,B)



3-help slide

A. MergeLists 클라이언트 함수 작성

- SortedType List의 경우, InsertItem()을 호출할 때마다 정렬이 되도록 맞추어 준다는 것을 활용하여 구현

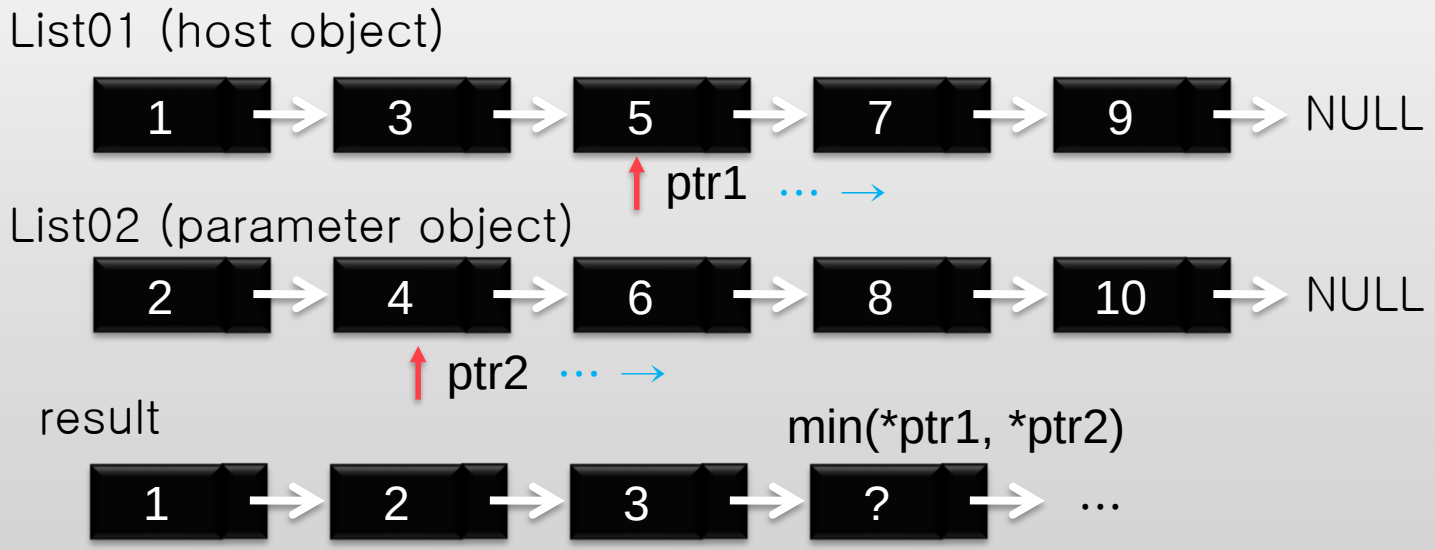
```
// MergeLists.h
template <class ItemType>
void MergeLists(SortedType<ItemType> &l_a, SortedType<ItemType> &l_b, SortedType<ItemType> &result)
{
        // l_a 리스트의 아이템들을 result에 삽입
        // l_b 리스트의 아이템들을 result에 삽입
        // l_a, l_b 리스트의 아이템들은 ResetList()와 GetNextItem()로 구성된 iterator를 사용하여 획득
}
```

3-help slide

B. MergeLists 멤버 함수 작성

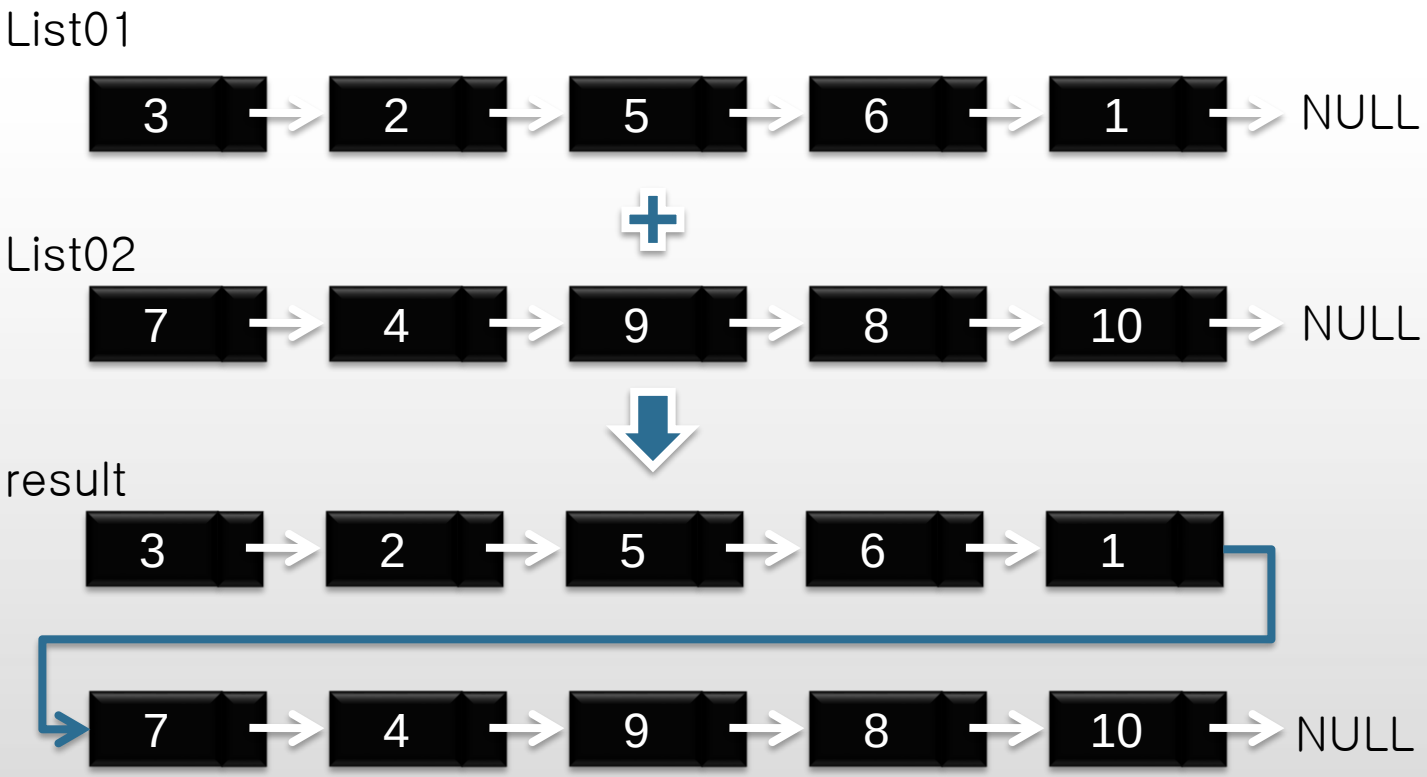
```
Template <class ItemType>
void SortedType<ItemType>::MergeLists(
                SortedType<ItemType> &other, SortedType<ItemType> &result)
{
    1. NodeType의 포인터 변수 2개를 추가하고, 각각 listData와 other list의 listData를 가리키도록 한다.
     Ex) NodeType<ItemType> *ptr1 = listData;

    2. 아이템을 비교 하며 합병을 시작한다. 단, 둘 리스트의 길이가 같다는 보장이 없으므로, 길이가 짧은 쪽이 끝났을 경우 긴쪽을 그 뒤에 붙이는 예외 처리를 하도록 한다.
    Ex) while(ptr1 != NULL && ptr2 != NULL) 그리고 둘 중 하나가 먼저 끝났을 때에 대한 처리.
}
```



두 개의 비정렬 연결 리스트를 합병하는 예제

- 한 리스트를 다른 리스트의 뒤에 붙이는 것으로 처리



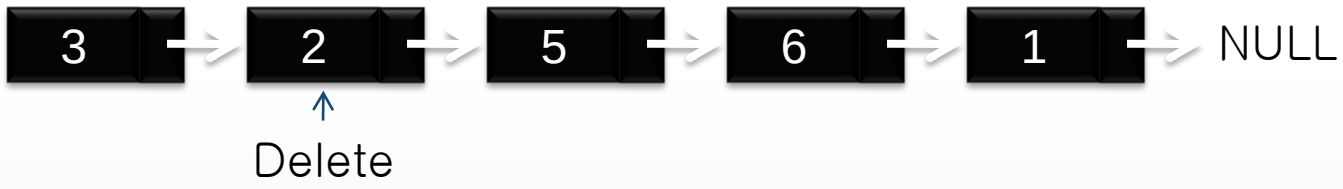
Exercise 4

■ 문 제

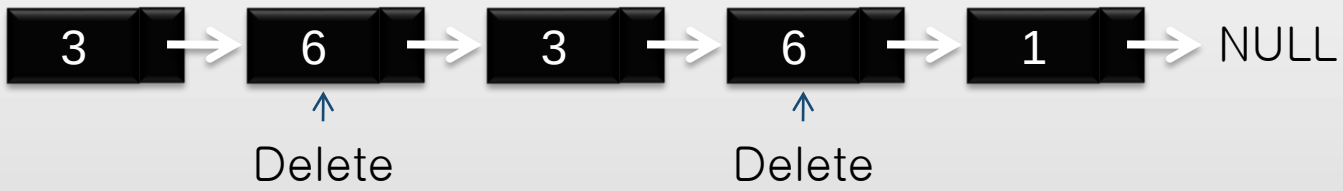
- **비정렬 리스트**내에서 Item을 삭제하는 DeleteItem 함수를 다음 요구사항을 만족하도록 수정하시오.
- **A. 리스트에는 같은 아이템이 존재 하지 않는다.**
  - 기존의 DeleteItem은 List내에 값이 있다는 전재 조건이 존재함, 없을 경우 에러가 발생함
  - 에러 발생 조건 : 값이 없는 상태로 마지막 노드를 검사할 경우 할당되지 않은 노드(NULL값을 갖음)에 대해서 접근하게 된다. 이는 메모리상에 할당이 되어 있지 않기 때문에 에러가 발생함
  - 해결 방법 : 삽입 과정에서 사용했던 preLoc과 location 변수 2개를 사용하는 방법이 있음
  - 에러가 발생하지 않도록 함수를 수정하시오.
- **B. 리스트에는 동일한 값을 가지는 아이템이 존재한다.**
  - 검색하고자 하는 아이템과 리스트 내에서 일치하는 모든 아이템들을 삭제하도록 함수를 수정하시오.

4-문제 추가 설명

- **A. 리스트에는 같은 아이템이 존재 하지 않음**

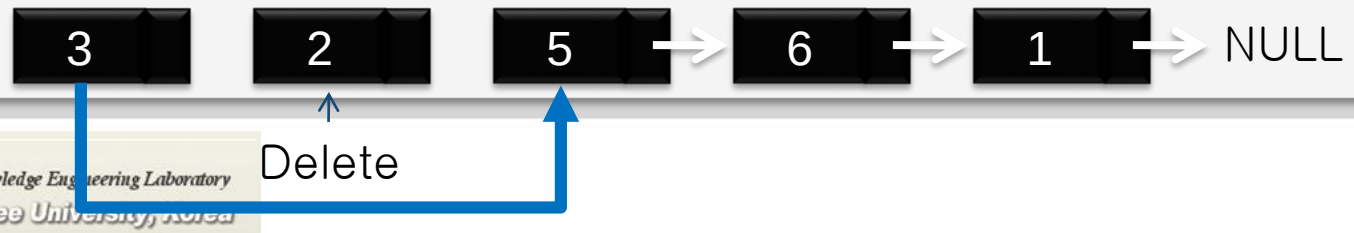


- **B. 리스트에는 동일한 값을 갖는 아이템이 존재함**



주의점!! 아이템 삭제 후에도 리스트는 유지 되어야 함

해결 방법 : 삭제되는 아이템 이전 아이템과 다음 아이템을 연결



A,B 문제에 대한 예제

모든 아이템을 삭제할 방법은 다양 하게 작성할 수 있습니다. 아래는 그 중 한 예입니다.

```
void DeleteItem(ItemType item)
{
    루프를 돌기 위해 Bool 변수를 추가합니다.
    아이템이 존재하는지 검색합니다.
    bool 변수가 참일 동안 루프를 수행합니다.
    {
    루프 내에서 매치되는 아이템을 삭제 합니다.
    아이템이 존재하는지 다시 검색합니다.
    }
}
```

Exercise5

문 제

- **정렬 리스트**내에서 Item을 삭제하는 DeleteItem 함수를 다음 요구사항을 만족하도록 수정하시오.
- A. 리스트에는 같은 아이템이 존재 하지 않는다.
  - 기존의 DeleteItem은 List내에 값이 있다는 전재 조건이 존재함, 없을 경우 에러가 발생함
  - 에러 발생 조건 : 값이 없는 상태로 마지막 노드를 검사할 경우 할당되지 않은 노드(NULL값을 갖음)에 대해서 접근하게 된다. 이는 메모리상에 할당이 되어 있지 않기 때문에 에러가 발생함
  - 해결 방법 : 삽입 과정에서 사용했던 preLoc과 location 변수 2개를 사용하는 방법이 있음
  - 에러가 발생하지 않도록 함수를 수정하시오.
- B. 리스트에는 동일한 값을 가지는 아이템이 존재한다.
  - 검색하고자 하는 아이템과 리스트 내에서 일치하는 모든 아이템들을 삭제하도록 함수를 수정하시오.

5-문제 추가 설명

■ **A. 리스트에는 같은 아이템이 존재 하지 않음**

Delete →item을 삭제 후 DeleteItem 함수 수행 중지

문제 11번처럼 값이 없을 때의 경우도 처리해줄 것!

■ **B. 리스트에는 같은 아이템이 존재할 수 있다.** 삭제되는 숫자보다 큰 수가 나타나면 삭제 연산 종료되도록 함

Delete Delete

주의점!! 아이템 삭제 후에도 리스트는 유지 되어야 함

해결 방법 : 삭제되는 아이템 이전 아이템과 다음 아이템을 연결

Delete Delete