Data Structures

Lab # 03

Data & Knowledge Engineering Laboratory
Kyung Hee University, Korea

Chapter 3 Programming Exercise

Lab 3

1. Exercise 6 (한글 교재 6)
2. Binary Search
3. Case Study

1. Exercise 6 (한글 교재 6)

■ 문제

❖ 다음과 같은 규격 명세를 사용하여 2개의 정렬 리스트 ADT를 병합시키는 클라이언트 함수를 작성하여라.

- 클라이언트 함수 : 클래스의 멤버함수가 아님, 클래스 멤버함수들을 사용하는 외부 전역함수

MergeList(SortedType list1, SortedType list2, SortedType& result)
함수: 2개의 정렬 리스트를 Merge해서 세 번째 정렬 리스트를 만든다.
조건: list1과 list2는 초기화되어 있고 ComparedTo라는 함수를 사용해서 키에 의해 정렬되어 있다. list1과 list2는 같은 키를 갖지 않는다.
결과: 결과는 list1과 list2의 모든 요소를 가진 정렬 리스트이다.

a. 함수를 작성하여라.
b. Big-O 표기법으로 알고리즘을 표현하여라.

❖ 문제에 필요한 샘플 소스 코드의 경로

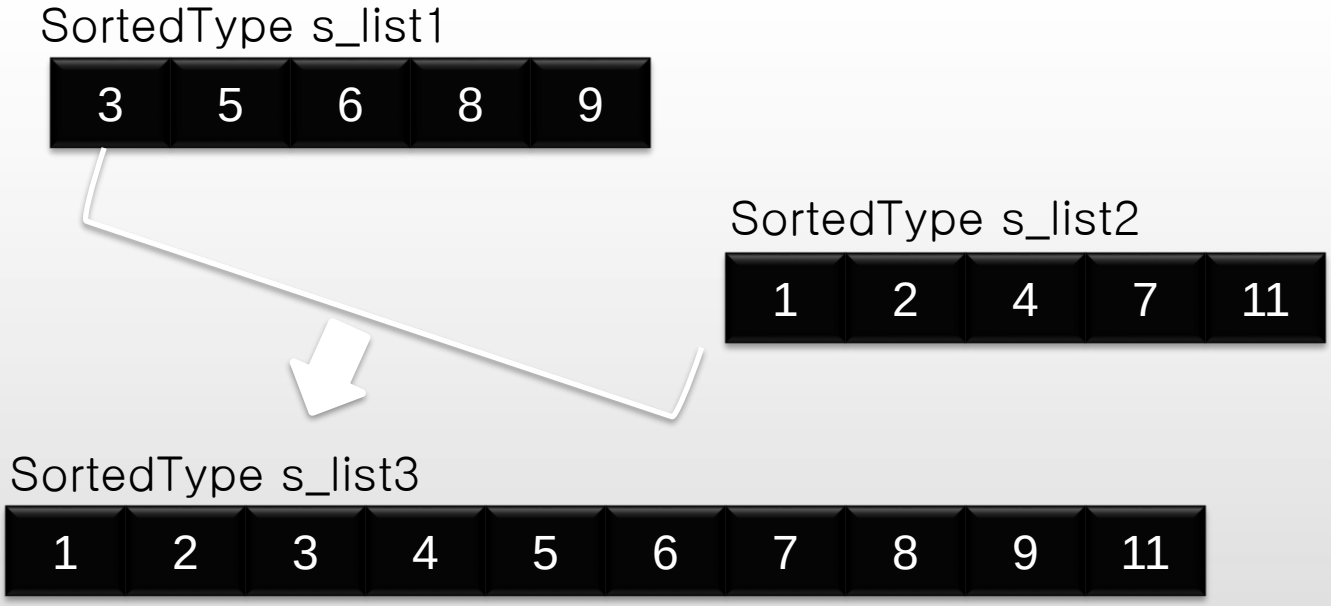
- ₩labplus₩Lab, c++ 3rd₩Chapter3₩Sorted"

ItemType.h
ItemType.cpp
sorted.h
sorted.cpp

4개의 파일을 사용합니다.

1-help slides (1/2)

- **이번 문제는 Sorted List 2개를 병합하여 결과를 다른 하나의 리스트에 저장하는 함수를 구현하고 테스트 드라이버를 작성하는 과제입니다.**
  - s_list1과 s_list2를 병합하여 s_list3에 저장하는 예시) (단, 정렬리스트이므로 결과는 순차적으로 정렬되있어야함)



- **a. 함수를 작성하여라**
  - "메인 함수가 정의된 문서"에 함수의 구현부 작성

1-help slides (2/2)

예제)

```
int main()
{
        SortedType s_list1, s_list2, s_list3; // 리스트 선언
        ItemType item1,item2,item3,item4,item5, … ;  //필요한 만큼 선언
        item1.Initialize(1); //item을 필요한 만큼 초기화
         s_list1.InsertItem(item3);  //리스트에 값을 넣는다.
        return 0;
}

void MergeList(SortedType list1, SortedType list2, SortedType& result)
{
        // 리스트의 current position을 초기화 한다.
        // list1과 list2의 길이를 LengthIs()함수를 통해 얻는다.
        // 정렬 리스트이므로  InsertItem(…)함수에 정렬하는 기능이 있다.
        // 따라서 list1의 길이만큼  반복하여 GetNextItem(…)로 item을 받아 result에 넣는다.
        // list2역시 길이만큼 루프를 돌며 GetNextItem(…)로 item을 얻고, result에 넣는다.
}
```

결과를 리턴하기 위해 reference 타입으로 선언

현재 ItemType.h에 MaxItem이 5로 정의되어 있습니다.

- 두 리스트내의 원소 개수의 합이 5를 넘지 않게하거나 MaxItem을 더 큰값으로 설정해주세요.

2. Binary Search

■ 문제

❖ A. 이진 탐색(binary search)를 위한 함수 BinarySearch()를 구현한다.

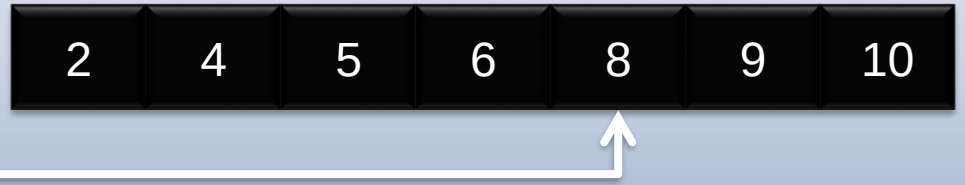
- int BinarySearch (int array[], int sizeOfArray, int value) :
  - 이 함수는 세 개의 인수를 갖는데, 첫번째는 integer의 배열이고, 두번째 변수는 배열의 크기(원소의 개수), 세번째 인수는 배열에서 찾고자 하는 integer값이다.
- 리턴 값 :
  - integer로서 배열의 몇 번째에 있는지를 나타낸다. 만일 배열에 찾는 대상이 없는 경우에는 -1을 리턴하도록 한다..

B. BinarySearch를 '수정'하여 찾고자 하는 값보다 작거나 같은 값들 중에서 가장 큰 값을 리턴하게 하려면 어떻게 하는가?

찾는 값: 7
리턴 값: 6

C. BinarySearch를 '수정'하여 찾고자 하는 값보다 크거나 같은 값들 중에서 가장 작은 값을 리턴하게 하려면 어떻게 하는가?

찾는 값: 7
리턴 값: 8



2-help slides (1/2)

- **Binary Search Algorithm 소스 코드를 바탕으로 BinarySearch 함수(전역 함수)를 구현합니다.**
  - ❖ Binary Search Algorithm 소스 코드

```
void  SortedType::RetrieveItem ( ItemType&  item,   bool&  found )
//  ASSUMES info ARRAY SORTED IN ASCENDING ORDER
{  int midPoint ;
   int  first  =  0;
   int    last  = length - 1 ;
   bool  moreToSearch  =  ( first  <=  last ) ;
   found = false ;
   while ( moreToSearch  &&  !found )
   {   midPoint  =  ( first + last ) / 2 ;
      switch ( item.ComparedTo( info [ midPoint ] ) )
      {
          case   LESS:
             last = midPoint - 1 ;
             moreToSearch = ( first <= last ) ;
             break ;
          case  GREATER  :   first = midPoint + 1 ;
             moreToSearch = ( first <= last ) ;
             break ;
          case  EQUAL    :   found = true  ;
             item = info[ midPoint ] ;
             break ;
      }
   }
}
```

개념만 참고!
세부 코드 수정 필요!

2-help slides (2/2)

- 다음 Test Driver를 이용하여 구현한 BinarySearch()함수를 테스트 합니다.

Ex) TestDriver

```
int BinarySearch(int [], int , int );

int main()
{
   int list[10] = {1,2,3,4,5,6,7,8,9,10};
   int result = BinarySearch(list, 10, 11);
   cout << result << endl; // -1 리턴
   result = BinarySearch(list, 10, 7);
   cout << result << endl; // 6 리턴
   return 0;
}
```

10은 magic number?
flexible 코드를 위한 Tip:
배열 크기 구하기
Example)
Sizeof(list)/sizeof(list[0])

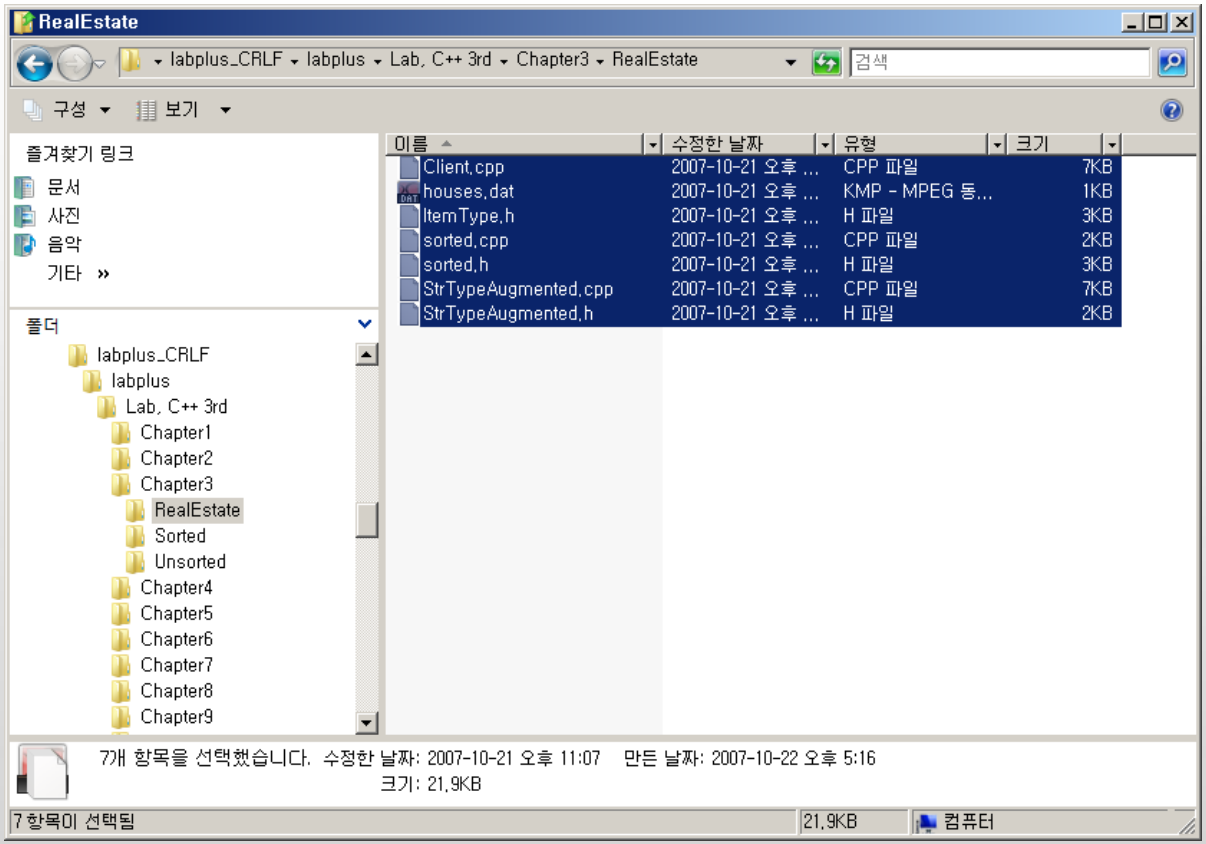
3. Case study

문제

- 교재 Case Study에 있는 프로그램 소스를 자세히 읽고 프로그램을 실제로 실행해 본다.
- A. HouseType에 bathroom의 개수를 나타내는 bathrooms 변수를 추가하고 이를 입력하고 출력할 수 있도록 프로그램을 수정한다
- B. HouseType에 relational operator < 와 ==를 overloading하고 ComparedTo함수를 이 두 연산자를 이용하여 구현하도록 수정한다.

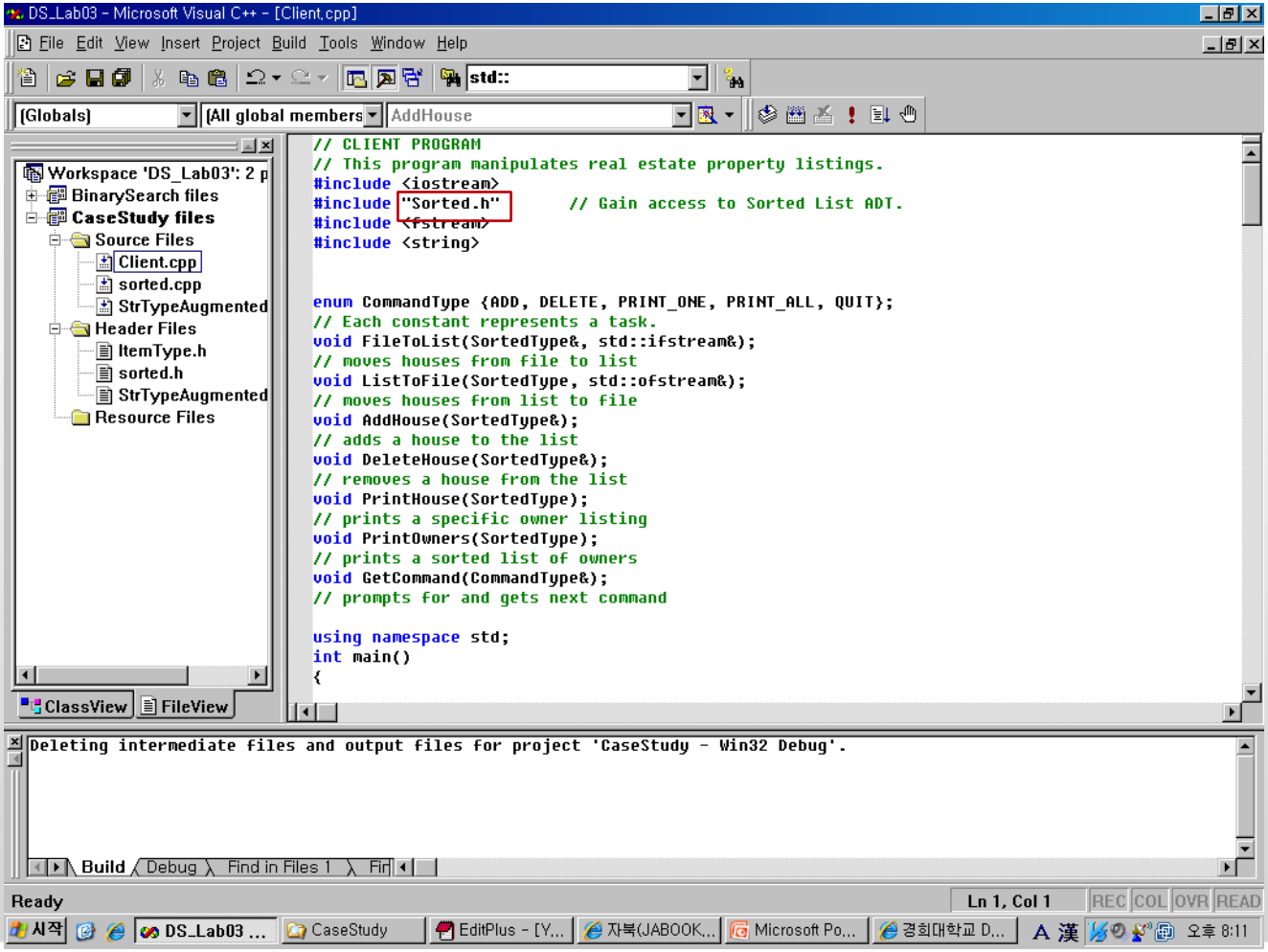
3. help-slide (1/7)

- **Case Study에 있는 프로그램 소스를 자세히 읽고 프로그램을 실제로 실행해 본다.**
  - ❖ Case Study에 나온 소스는 교제 소스 코드에 제공되고 있습니다.



3-help stlide (2/7)

- **Step1. 교재가 제공하는 소스코드를 사용하는 프로젝트 폴더에 복사한 후 복사한 파일을 프로젝트에 추가합니다.**

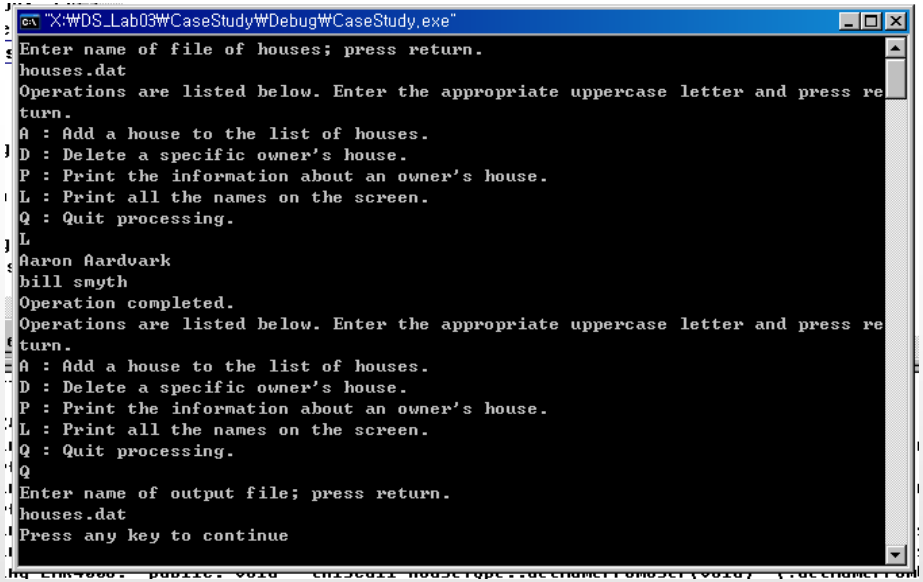


헤더 이름 그림과 다르니 참고!

ItemType.h 구현 부분을 ItemType.cpp로 분리해야 실행됨!!

3-help slide (3/7)

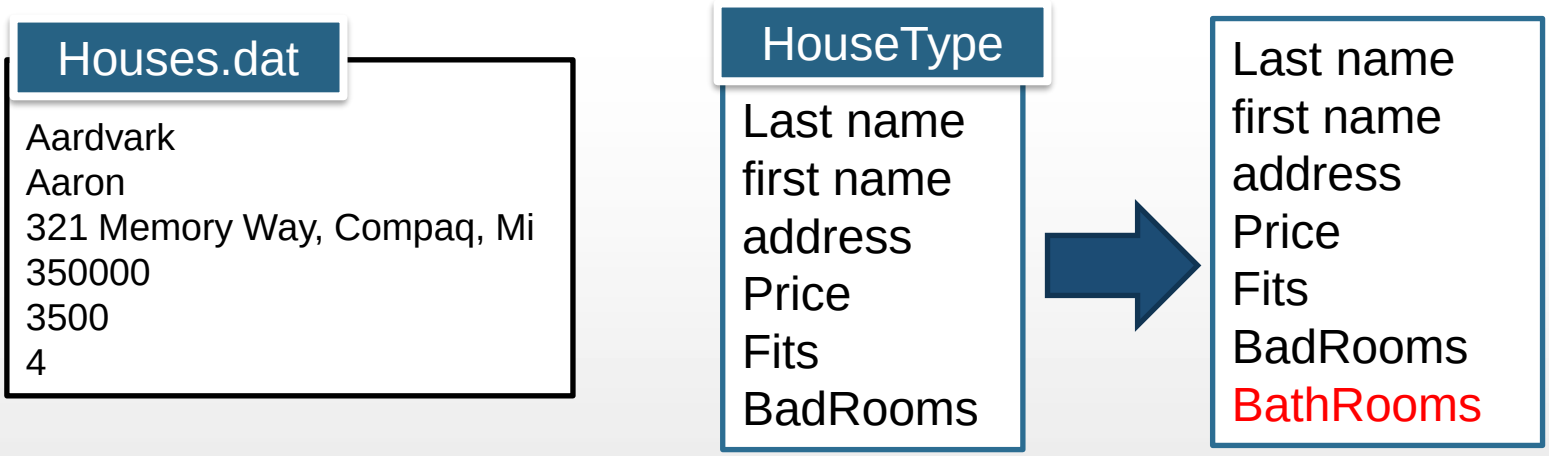
- **Step2. 모든 소스코드에 대해서 개행문자 변환을 한 후에 컴파일 후 실행합니다.**
  - 실행화면



  - 실행하면 다음과 같은 메시지가 나옵니다.
    - Enter name of file of houses; press return.
    - houses.dat를 입력하세요. (소스와 같이 제공되는 파일입니다.
  - 메뉴가 나오면
    - 'L'을 선택해 기본으로 들어있는 이름이 제대로 출력되는지 확인합니다.
    - 다른 메뉴를 테스트 한 후 A,B번 문제를 하세요.

3-help slide (4/7)

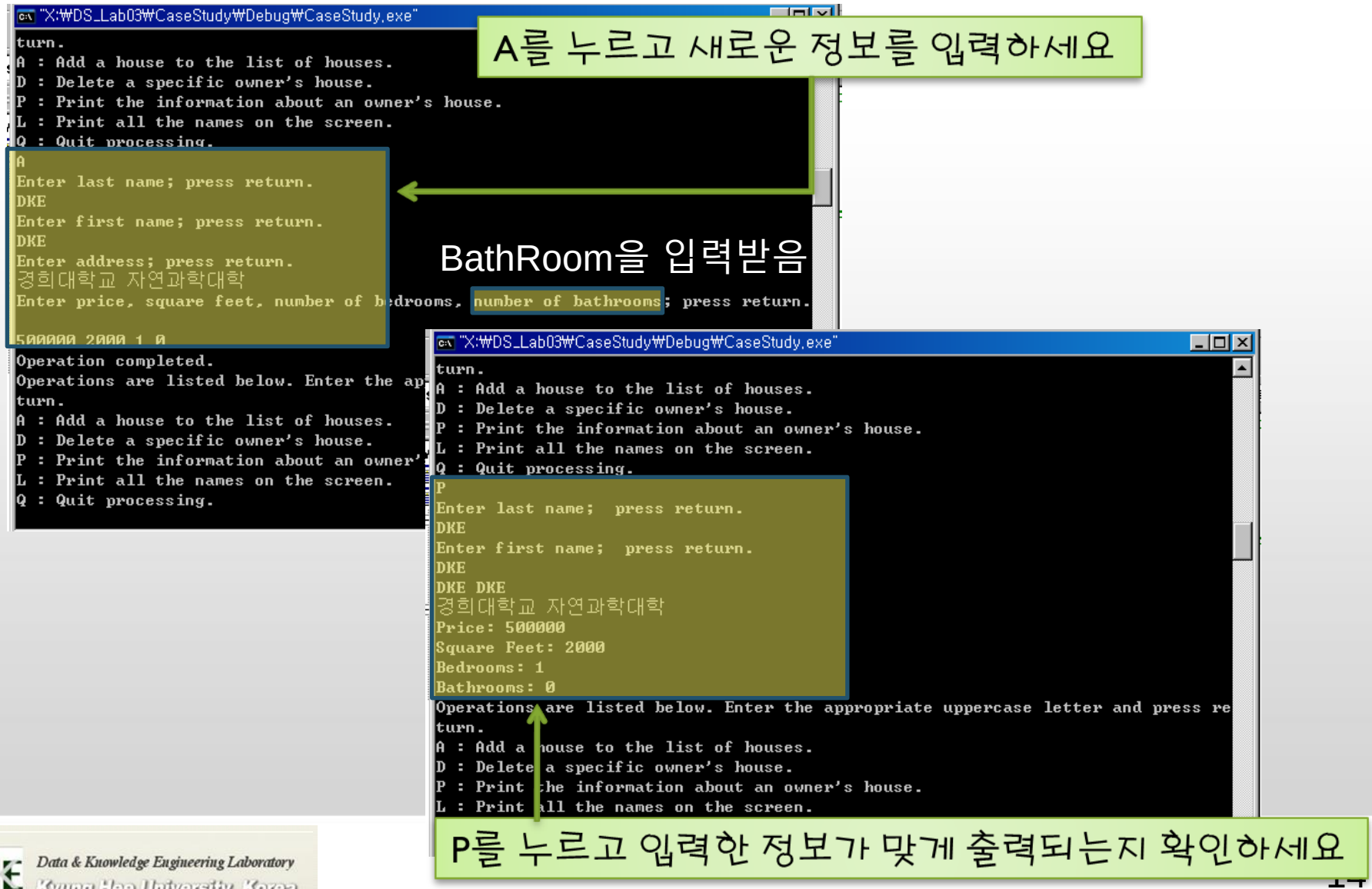
- **A. 번 문제의 변경해야 할 사항**
  - 현재 저장된 파일의 format (ascii file)



- **HoustType(ItemType) 클래스에 private 변수로 int BathRooms을 추가**
- **파일 입출력 연산**
  - 위 순서에 맞추어 각 요소를 읽고 쓸 수 있게 수정
  - BathRooms 변수를 읽고 쓸 수 있도록 추가
- **화면 입출력 연산**
  - 위 순서에 맞추어 각 요소가 화면에 출력되고 사용자로부터 입력 받을 수 있게 수정
  - BathRooms 변수를 읽고 쓸 수 있도록 추가

3-help slide (5/7)

A번 테스트



3-help slides (6/7)

- **B. HouseType에 relational operator < 와 ==를 overloading하고 ComparedTo함수를 이 두 연산자를 이용하여 구현하도록 수정한다.**
  - ItemType클래스 맴버함수로 정의하여 HouseType 형 객체들에 대해서 크기 비교를 수행한다고 가정하였을 때 다음과 같이 구현할 수 있다.
    - bool HouseType::Less(const HouseType& other)

      {....}
    - 사용 예:

      ItempType a, b;

      if(a.LessForItemType(b)) { ....}
    - 문제점 : 코드의 가독성이 좋지 않고 사용하기 불편함, 크기 비교의 결과를 이해하기가 어려움 (a가 큰가? b가큰가?)
  - **Operator Overloading**을 구현하여 HouseType형 객체들에 대해서 크기 비교를 수행한다면 사용법은 다음과 같다.
    - 사용 예 :
    - if(a < b)
    - 두 값을 연자 '<'를 비교하게되면, 코드의 가독성이 좋아지고, 사용이 용이함
    - 함수 명 Less를 operator<로 바꿔주기만 하면 OK!

      bool ItemType::operator<(const ItemType& other)

      {....}

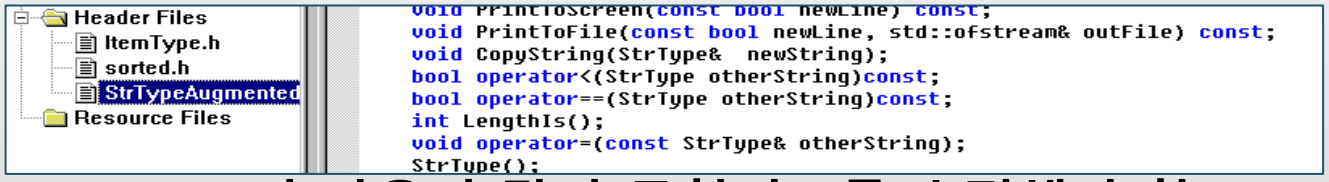
ItemType.cpp, ItemType.h 파일의 HouseType Class에 대해서
bool operator<(const HouseType& other); 메소드 추가 구현

3-help slide (7/7)

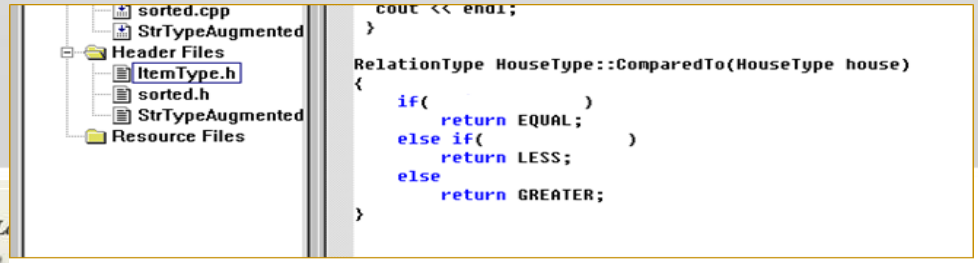
- HouseType의 객체를 비교하기위한 함수는 현재 다음과 같이 객체 내의 이름에 관련된 멤버변수에 대해서 비교하도록 정의되어 있다.

```
RelationType HouseType::ComparedTo(HouseType house)
{
  if (lastName < house.lastName)
    return LESS;
  else if (house.lastName < lastName)
    return GREATER;
  else if (firstName < house.firstName)
    return LESS;
  else if (house.firstName < firstName)
    return GREATER;
  else return EQUAL;
}
```

- 위와 같은 방법으로 비교할수 있도록 Operator overloading을 구현한다. Operator overloading의 예는 같이 제공되는 StrTypeAugmented에 구현되어져 있습니다. 구현 예를 참고하세요



- ComparedTo()가 다음과 같이 구현되도록 수정해야 함
  - ❖ 구현한 HouseType에 대한 operator를 이용하여 객체를 비교함



Hint
"*this" 를 활용