자료구조 실습04

실습 내용

- Generic Sorted Linked List class 정의 및 구현
- Sorted Linked List Application 구현

예제: SortedLinkedList ADT(1/2)

```
template <typename T>
class LinkedList
{
public:
      LinkedList();                    // Constructor
     ~LinkedList();                    // Destructor

      void MakeEmpty();                // List를 비움..
      int GetLength() const;  // 리스트가 보유하고 있는 item 개수 반환
      int Add(T item);                 // 새로운 레코드를 리스트에 삽입.
      int Delete(T item);              // 입력된 레코드를 찾아서 삭제.
      int Retrieve(T &item);           // Primary key를 기준으로 데이터를 검색하고 해당 데이터를 가져옴
      void ResetList();                // 레코드 포인터 초기화
      void GetNextItem(T &item); // Current Pointer 가 다음 node 를 가리키도록 이동 후 해당 레코드를 가져옴

 private:
      NodeType<T> *m_pList;            // 리스트 포인터
      NodeType<T> *m_pCurPointer;      // current pointer
      int m_nLength;                   // 리스트에 저장된 레코드 수
};
```


예제: SortedLinkedList ADT(2/2)

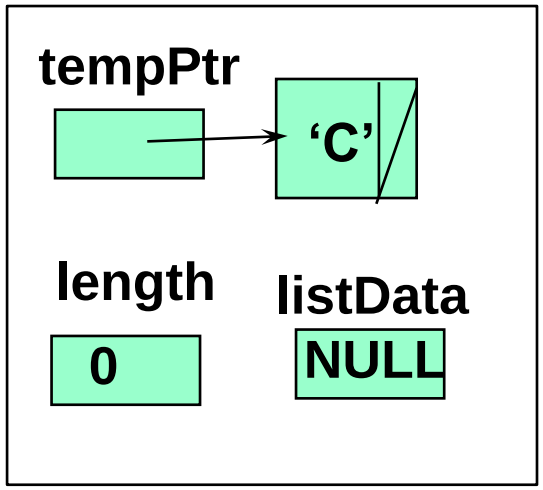
◎ 하나의 레코드를 저장하는 노드를 **struct**로 정의

```
template <typename T>
struct NodeType
{
    T info;                 // 노드에서 관리할 레코드
    NodeType *next;         // 다음 노드를 가리키는 포인터
};
```

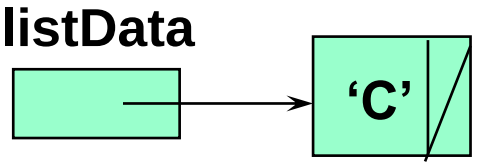

SortedList::Add()

동영상 강의자료 참조

Case 1: 리스트가 비어 있는 경우




```
if (listData==NULL)          {
      listData=tempPtr;
      tempPtr->next=NULL;
      length++;
 }
```



SortedList::Add()

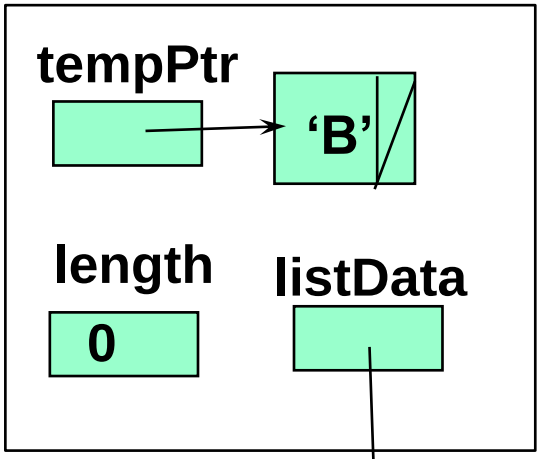
추가할 위치를 찾는다

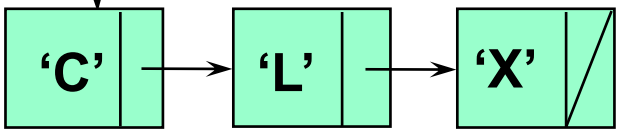
```
 location = predLoc=listData ;
moreToSearch = ( location !=NULL ) ;
while ( moreToSearch  )
{
    if (item <  location->info )  // 현재 노드 앞에 추가
          moreToSearch = false ;
    else if (item > location->info) {
          predLoc=location;
          location=location->next;
          moreToSearch = ( location !=NULL ) ;
    }
    else    // EQUAL:
         display error message and return
}

// 맨 앞이면 predLoc==location
// 중간이면  location!=NULL
// 맨 뒤면 location==NULL
```

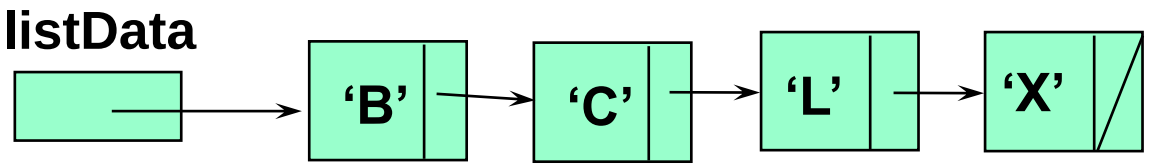

SortedList::Add()

Case 2: 맨 앞에 추가해야 하는 경우(**predLoc==location)**



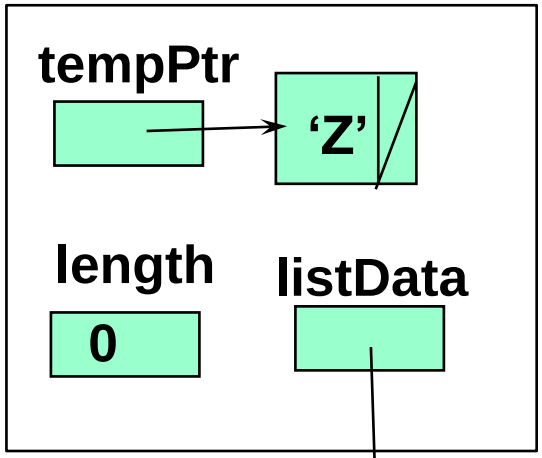



```
tempPtr->next = listData;
listData=tempPtr;
length++;
```



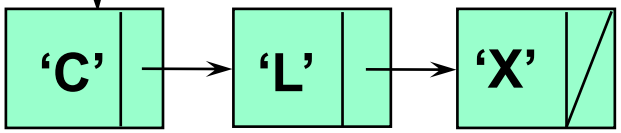
SortedList::Add()

Case 3: 중간 또는 맨 뒤에 추가해야 하는 경우

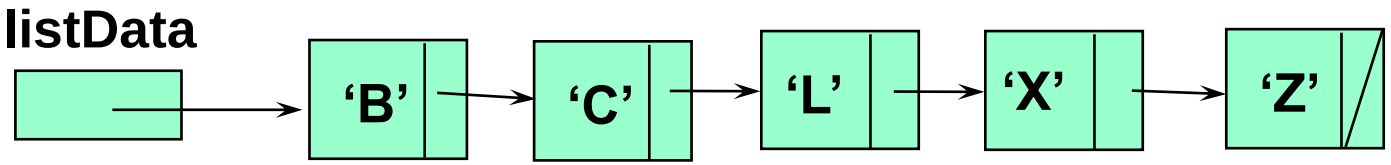


// 중간이면 location!=NULL
// 맨 뒤면 location==NULL

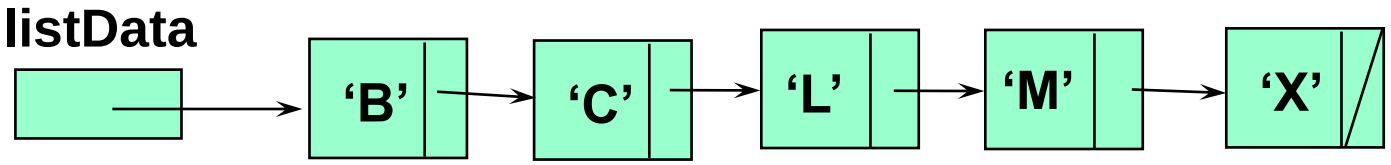
predLoc->next = tempPtr;
tempPtr->next=location;



'Z' 추가



'M' 추가



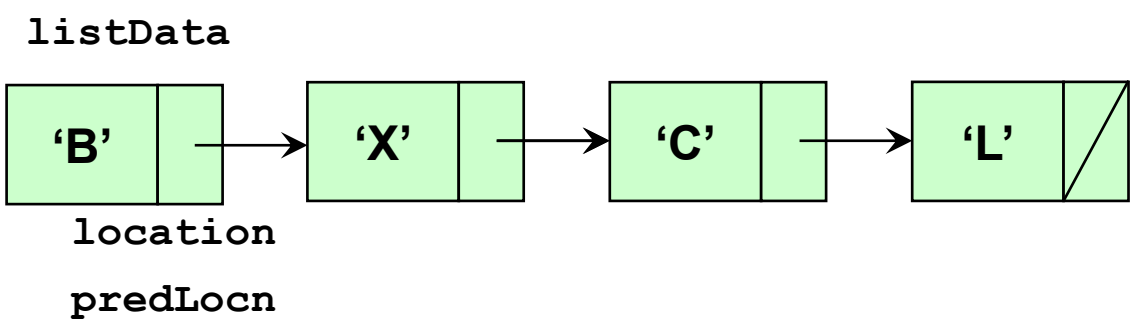
```
void  SortedType :: InsertItem ( ItemType  item )
{    bool  moreToSearch ;
     NodeType<ItemType>*  location, predLoc, tempPtr ;
     // Allocate new node and store new item
     tempPtr = new  NodeType<ItemType> ;
     tempPtr->info = item ;
     // Empty list. Add the first item
     if (listData==NULL)        {
            listData=tempPtr;    tempPtr->next=NULL;     length++;
       }
     else   {
           // Find the position of new item
           location = predLoc=listData ;
          moreToSearch = ( location !=NULL ) ;
          while ( moreToSearch  )
          {
               if ( item< location->info  ) // insert at the front of the current node
                    moreToSearch = false ;
```

```
                    else if (item>location->info) {
                        predLoc=location;
                        location=location->next;
                        moreToSearch = ( location !=NULL ) ;
                     }
                     else   // case EQUAL: error message
                }
            }
            if (predLoc==location) { // add to front
                tempPtr->next = listData;     listData=tempPtr;
            }
            else {       // add between  two nodes or to the end(location==NULL)
                predLoc->next = tempPtr;
                tempPtr->next=location;
            }
            length++ ;
        }
    }
```

SortedList::Delete()

DeleteItem 'C'

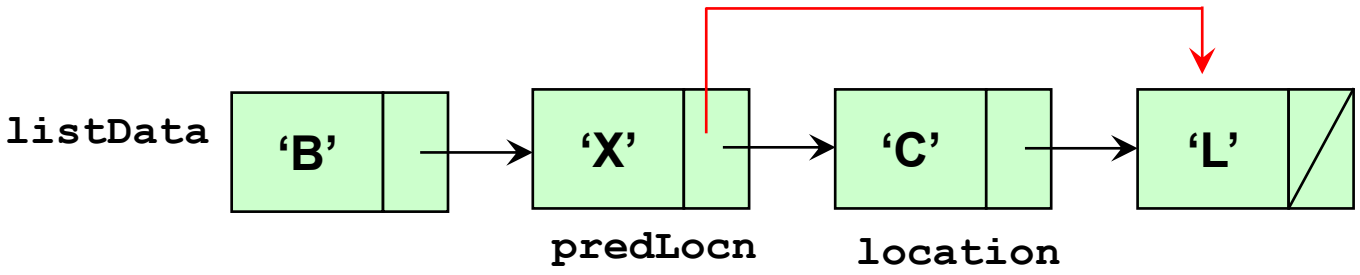
```
location = predLoc = listData;
moreToSearch= (location!=NULL);
while(moreToSearch) {
  if(location->info != DeleteItem  {
     predLoc = location; location = location->next;
     moreToSearch = (location !=NULL);}
  else  { // delete current item
     preLoc->next = location->next;
     delete location;
     moreToSearch = false; }
}
```



SortedList::Delete()

DeleteItem ‘C’

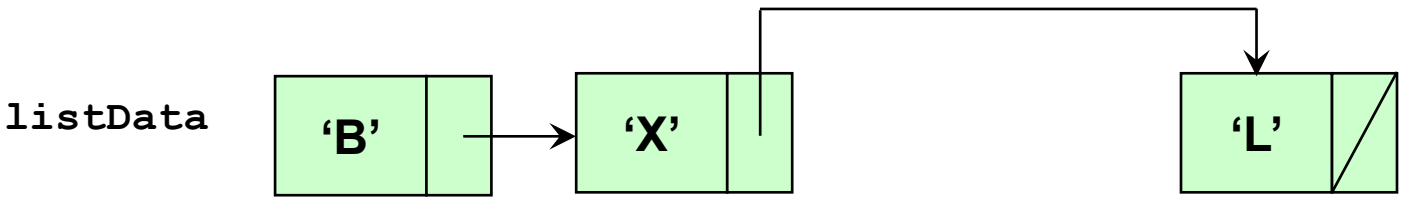
```
location = predLoc = listData;
moreToSearch=(location!=NULL);
while(moreToSearch) {
  if(location->info != DeleteItem  {
     predLoc = location; location = location->next;
     moreToSearch = (location !=NULL);}
  else  { // delete current item
     predLoc->next = location->next;
     delete location;
     moreToSearch = false; }
}
```



SortedList::Delete()

DeleteItem ‘C’

```
location = predLoc = listData;
moreToSearch=(location!=NULL);
while(moreToSearch) {
  if(location->info != DeleteItem  {
     predLoc = location; location = location->next;
     moreToSearch = (location !=NULL);}
  else  { // delete current item
     predLoc->next = location->next;
     delete location;
     moreToSearch = false; }
}
```



예제: console

◎ List를 테스트할 driver는 다음과 같이 작성함

◎ 첨부한 소스코드 참조바람

```
--- ID – Command -----
        1 : Insert Item
        2 : Delete Item
        3 : Replace Item
        4 : Retrieve Item
        5 : Display all Item
        0 : Quit

Choose a Command -->
```


실습 : List의 응용

◎ 실습 3에서 구현한 배열을 이용한 sorted list를 이용하여 상품들의 자세한 정보를 관리하는 master list를 구현.

☞ masterList : 쇼핑몰에 등록된 모든 상품의 자세한 정보를 관리(상품의 고유번호 순으로 정렬)

◎ 실습 4 예제에서 구현한 sorted singly list를 이용하여 카테고리 리스트를 구현.

☞ 일단 카테고리가 하나라고 가정한다. (예: 노트북)

☞ 노트북 카테고리에 소속된 상품정보에 관한 다음과 같은 정보를 저장하는 SimpleProductType을 정의한다.

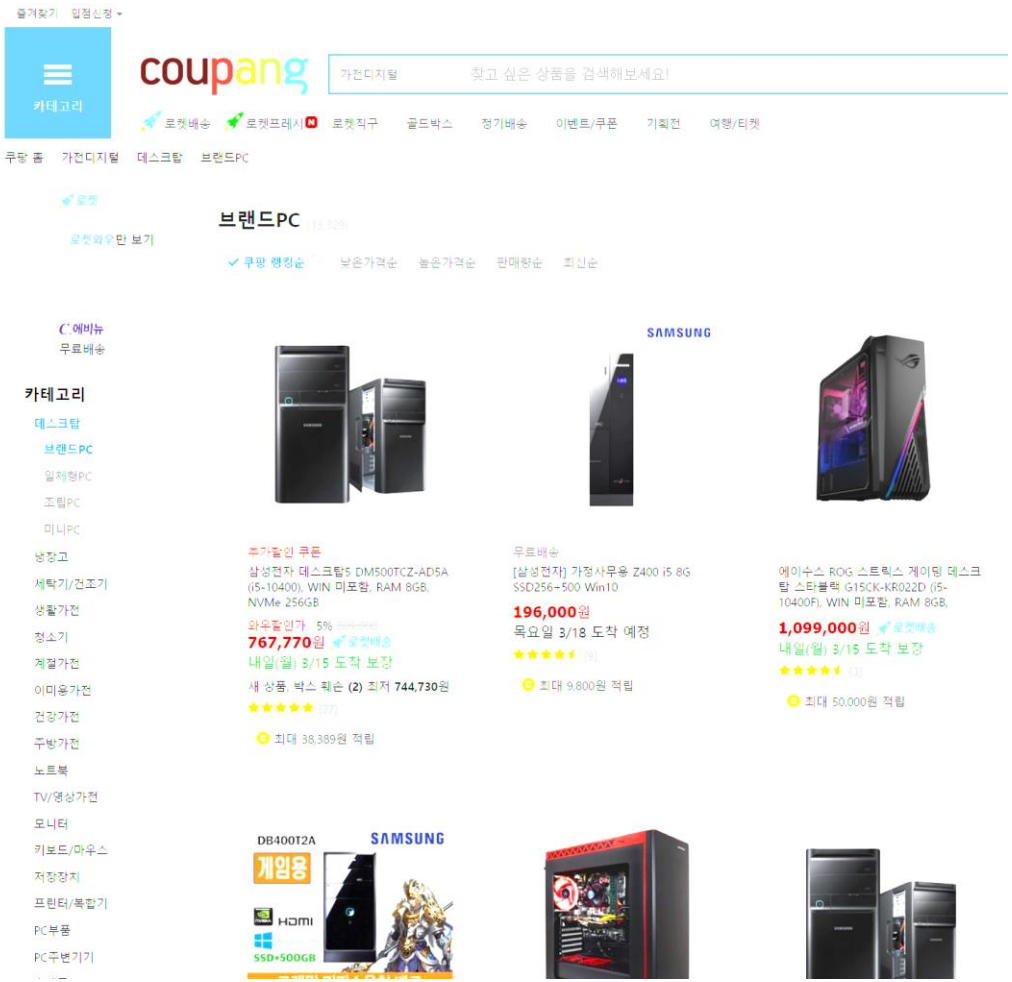
```
string ID;                    // 상품 ID
```


카테고리


카테고리
coupang
뷰티
로켓배송
로켓프레시 N
2021 설날 N
로켓직구
패션의류/잡화
뷰티
출산/유아동
식품
주방용품
생활용품
홈인테리어
가전디지털
스포츠/레저
자동차용품
도서/음반/DVD
완구/취미
문구/오피스
TV/영상가전
냉장고
세탁기/건조기
생활가전
청소기
계절가전
이미용가전
건강가전
주방가전
노트북
데스크탑
모니터
키보드/마우스
브랜드PC
일체형PC
조립PC
미니PC

카테고리 마지막 항목

카테고리 마지막 항목을 선택하면 해당 항목에 속한 상품들만 출력된다.



실습 : List의 응용

◎ application class에 다음과 같은 멤버변수를 추가한다.

- ☞ List_Sorted_Array<ItemType> masterList;
- ☞ List_S_SLinked<SimpleProductType> cateList; //카테고리

◎ 다음과 같은 기능을 추가

- ☞ AddToCateList(): 카테고리에 등록할 상품 ID와 추가적인 정보를 읽어서 cateList에 추가
- ☞ DisplayCateList(): cateList()에 등록된 상품의 자세한 정보를 화면에 출력
  - ➢ 방법: cataList에 있는 상품정보(SimpleProductType)을 차례로 읽어서 masterList에서 검색하여 화면에 출력
- ☞ FindProductInCate(): 카테고리에 있는 상품을 검색하여 자세한 정보를 화면에 출력

실습과제 4

◎ 과제 내용:

- 실습 3에서 구현한 배열을 이용한 sorted list를 이용하여 상품들의 자세한 정보를 관리하는 master list를 구현한다.
  - masterList : 쇼핑몰에 등록된 모든 상품의 자세한 정보를 상품의 고유번호 순으로 정렬하여 관리
- 실습 3에서 구현한 QueueList를 이용한 장바구니 기능을 추가
- 실습 4에서 구현한 sorted singly list를 이용하여 하나의 카테고리 정보를 저장하는 카테고리 클라스(CategoryType)을 구현
  - 여기에는 다음과 같은 정보를 포함할 수 있다.
    - ◆ Int itemNum; // 소속된 상품 수
    - ◆ List_S_Slink<SimpleProductType> cateList: // 현 카테고리에 소속된 상품 리스트
  - 카테고리에 소속된 하나의 상품정보를 표현하기 위한 자료형(SimpleProductType)을 정의. 여기에는 다음과 같은 정보가 포함될 수 있다.
    - ◆ string name; // 상품명
    - ◆ int ID; // 상품 ID

실습과제 4

- application class에 다음과 같은 멤버변수를 추가한다.
  - List_Sorted_Array<ItemType> masterList;
  - BasketType basket; // 장바구니
  - CategoryType category; // 메뉴 카테고리
- Application class에 다음과 같은 기능을 추가
  - 실습 2, 3, 4에서 구현한 기능들을 결합하여 완성한다.
- 과제 제출
  - 2021년 4월 4(일)일 23:50 까지
  - 방법은 이전과 동일

- 계층적인 카테고리 시스템 구현을 어떻게 할 것인지를 구상해 보기 바람.
- 인터넷 쇼핑몰은 GUI를 카테고리를 선택한다. 같은 기능을 console program으로 구현할 경우에 어떻게 할 것인지 생각해 보기 바람.
- 이를 위해서 2017년에 자료구조를 수강한 학생들이 구현한 멀티미디어 관리 시스템 소스를 첨부하였으니 분석하여 보세요.
  - 첨부한 폴더로 멀티미디어를 관리한다. 여기서 sub_folder가기와 이전 폴더 가기 기능이 도움이 될 수 있다.