5. General Linear Lists

msdb@korea.ac.kr

Agenda

Instruction

OrderedList ADT

Instruction

It is a little different from list ADT in the book

We have only one question this time!

Submit orderedListADT.h, orderedListADT.c, main.c

OrderedList ADT

Implement OrderedList ADT.

It contains struct student as its elements.

Id (int), name (string), email (string)

Operations

createList - create a list

destroyList - destroy a list

insert - insert an element in ascending order of id

- Duplicate keys are not allowed.

remove - erase an element with specific id

retrieve - return an element with specific id

size – return the number of elements

isEmpty - return true if a list is empty

iterate - traverses a list

initIterator - initialize a position of an iterator

printList - print all elements in list

OrderedListADT – Type Definitions

```
#define STR_MAX 32

typedef struct student
{
     int id;
     char name[STR_MAX];
     char email[STR_MAX];
} STUDENT;

typedef struct node
{
     void* pData;
     struct node* next;
} NODE;

typedef struct
{
     int size;
     NODE* pos;
     NODE* head;
     NODE* rear;
     int(*compare) (void* pArg1, void* pArg2);
} LIST;
```

OrderedListADT – Function Declarations

```
LIST* createList(int(*compare)(void* pArg1, void* pArg2));
LIST* destoryList(LIST* pList);
bool insert(LIST* pList, void* pDataIn);
bool remove(LIST* pList, void* pKey);
bool retrieve(LIST* pList, void* pKey, void** pDataOut);
int size(LIST* pList);
bool isEmpty(LIST* pList);
bool iterate(LIST* pList, void** pDataOut);
void initIterator(LIST* pList);
void printList(LIST* pList, void(*print)(void* pArg));

void printStudent(void* pArg);
int cmpStudentId(void* pStudent1, void* pStudent2);

static bool _insert(LIST* pList, NODE* pPre, void* pDataIn);
static void _delete(LIST* pList, NODE* pPre, NODE* pCur);
static bool _search(LIST* pList, NODE** pPre, NODE** pCur, void* pKey);
```

OrderedListADT - _search

```
static bool _search(LIST* pList, NODE** pPre, NODE** pLoc, void* pKey)
{
    #define COMPARE (((*pList->compare)(pKey, (*pLoc)->pData)))
    #define COMPARE_LAST ((*pList->compare) (pKey, pList->rear->pData))

    int result;
    *pPre = NULL;
    *pLoc = pList->head;
    if (pList->size == 0)
        return false;

    if (COMPARE_LAST > 0)
    {
        *pPre = pList->rear;
        *pLoc = NULL;
        return false;
    }

    while ((result = COMPARE) > 0)
    {
        *pPre = *pLoc;
        *pLoc = (*pLoc)->next;
    }

    if (result == 0)
        return true;
    else
        return false;
}
```

MAIN

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include "orderedListADT.h"

STUDENT csDummy[] =
{
      {320001, "yangbong", "beebee@korea.ac.kr" },
      {320002, "hyoyong", "utility@korea.ac.kr" },
      {320003, "daegi", "waiting@korea.ac.kr" },
      {320004, "yondon", "ydchung@korea.ac.kr" },
      {320005, "minsoo", "msdb@korea.ac.kr" }
};

int main()
{
      int dummySize = sizeof(csDummy) / sizeof(STUDENT);
      LIST *csStudents = createList(cmpStudentId);

      // Insert dummy data
      for (int i = 0; i < dummySize; i++)
      {
            STUDENT* student = (STUDENT*)malloc(sizeof(STUDENT));
            student->id = csDummy[i].id;
            strcpy(student->name, csDummy[i].name);
            strcpy(student->email, csDummy[i].email);

            if (!insert(csStudents, (void*)student))
                  printf("Insertion failed\n");
      }

      // Iterate elements
      STUDENT* curStudent;
      while (iterate(csStudents, (void**)&curStudent))
            printStudent(curStudent);
      initIterator(csStudents);

      // Search elements
      int searchKeyOrder[] = { 320001, 320003, 120001 };
      for (int i = 0; i < sizeof(searchKeyOrder) / sizeof(int);   }
      i++)
      {
            int key = searchKeyOrder[i];
            STUDENT* pStudent;

            if (retrieve(csStudents, (void*)&key,
            (void**)&pStudent))
            {
                  printf("Student found (key: %d)\n", key);
                  printStudent(pStudent);
            } else
                  printf("Search failed (key: %d)\n", key);
      }

      // Remove elements
      int eraseKeyOrder[5] = { 320001, 320005, 320003, 320004,
      320002 };
      for (int i = 0; i < sizeof(eraseKeyOrder) / sizeof(int);
      i++)
      {
            int key = eraseKeyOrder[i];

            if (remove(csStudents, (void*)&key))
                  printf("Erase successed (key: %d)\n", key);
            else
                  printf("Erase failed (key: %d)\n", key);

            printList(csStudents, printStudent);
      }

      // List empty check
      if (isEmpty(csStudents))
            printf("Empty\n");
      else
            printf("Not empty\n");

      // Destory list
      destoryList(csStudents);

      return 0;
```

Output

```
####################
# Insert dummy data

####################
# Iterate elements
STUDENT ID: 320001
NAME: yangbong
EMAIL: beebee@korea.ac.kr
--------------------
STUDENT ID: 320002
NAME: hyoyong
EMAIL: utility@korea.ac.kr
--------------------
STUDENT ID: 320003
NAME: daegi
EMAIL: waiting@korea.ac.kr
--------------------
STUDENT ID: 320004
NAME: yondon
EMAIL: ydchung@korea.ac.kr
--------------------
STUDENT ID: 320005
NAME: minsoo
EMAIL: msdb@korea.ac.kr
--------------------
```

```
####################
# Search elements
Student found (key: 320001)
STUDENT ID: 320001
NAME: yangbong
EMAIL: beebee@korea.ac.kr
--------------------
Student found (key: 320003)
STUDENT ID: 320003
NAME: daegi
EMAIL: waiting@korea.ac.kr
--------------------
Search failed (key: 120001)

####################
# Remove elements
Erase successed (key: 320001)
STUDENT ID: 320002
NAME: hyoyong
EMAIL: utility@korea.ac.kr
--------------------
STUDENT ID: 320003
NAME: daegi
EMAIL: waiting@korea.ac.kr
--------------------
STUDENT ID: 320004
```

```
NAME: yondon
EMAIL: ydchung@korea.ac.kr
--------------------
STUDENT ID: 320005
NAME: minsoo
EMAIL: msdb@korea.ac.kr
--------------------
====================
Erase successed (key: 320005)
STUDENT ID: 320002
NAME: hyoyong
EMAIL: utility@korea.ac.kr
--------------------
STUDENT ID: 320003
NAME: daegi
EMAIL: waiting@korea.ac.kr
--------------------
STUDENT ID: 320004
NAME: yondon
EMAIL: ydchung@korea.ac.kr
--------------------
====================
Erase successed (key: 320003)
STUDENT ID: 320002
NAME: hyoyong
EMAIL: utility@korea.ac.kr
```

```
--------------------
STUDENT ID: 320004
NAME: yondon
EMAIL: ydchung@korea.ac.kr
--------------------
====================
Erase successed (key: 320004)
STUDENT ID: 320002
NAME: hyoyong
EMAIL: utility@korea.ac.kr
--------------------
====================
Erase successed (key: 320002)
====================

####################
# List empty check
Empty
계속하려면 아무 키나 누르십시오 . . .
```

Common Mistakes

To use boolean type and values:

- Include <stdbool.h>

Integer / Integer = Integer

- Use type casting.
- (double)integer / integer = double

Be careful with dynamic memory allocation.

- Avoid memory leaks

Use the correct pair of dynamic memory allocation operators.

- malloc - free