연결리스트 - 순차리스트

3주차-강의

남춘성

SUNG KYUN KWAN UNIVERSITY

추상 자료형 ADT(Abstract Data Type)

SUNG KYUN KWAN UNIVERSITY

- 추상자료형 정의
  - 구체적인 기능의 완성과정을 언급하지 않고, 순수하게 기능이 무엇인지를 나열하는 것.
  - 기능의 논리적인 관계를 정의하기 보다는 기능의 명세를 가르키는 것.
  - 자료형의 구체적인 내용을 기술하기 보다는 연산을 위주로 정의.
  - 예)
    - 지갑 :
      - 지갑에서 돈을 넣는다.
      - 지갑에서 돈을 꺼낸다.

```
void PutMoneyu(Wallet *pw, int coinNum, int billNum)
int TakeOutMoney(Wallet *pw, int coinNum, int billNum)
```

- 리스트의 추상자료형 순서
  - 리스트의 ADT를 정의
  - ADT를 근거로 리스트 자료구조를 활용한 main함수를 정의
  - ADT를 근거로 리스트를 구현

배열을 이용한 순차 리스트와 ADT 정의

SUNG KYUN KWAN UNIVERSITY

- 순차 리스트와 연결리스트
  - 차이점
    - 순차 리스트 : 배열을 기반으로 구현된 리스트
    - 연결 리스트 : 메모리의 동적 할당을 기반으로 구현된 리스트
  - 공통점
    - 데이터를 나란히(하나의 열) 저장, 중복된 데이터의 저장을 막지 않음(허용)

- 리스트의 ADT 정의
  - 리스트 초기화
  - 리스트에 데이터 저장
  - 저장된 데이터의 탐색 및 탐색 초기화
  - 다음 데이터의 참조(반환)
  - 데이터 삭제 (바로 이전에 참조(반환)가 이루어진)
  - 현재 저장되어 있는 데이터 수를 반환

배열을 이용한 순차 리스트의 ADT 구현 - I

SUNG KYUN KWAN UNIVERSITY

- 리스트의 ADT 구현
  - 자료형 List 정의
    - 리스트의 저장소인 배열 정의 : Ldata(int)형인 arr를 LIST_LEN 만큼 정의
    - 저장된 데이터의 수 : 저장된 원소의 수(num of Data)
    - 데이터 참조위치를 기록 : 현재 데이터가 저장된 참조 위치 (current Position)

```
#define LIST_LEN 100    //배열의 길이 정의

typedef int LData;        // int형인 Ldata 선언

typedef struct __ArrayList  //배열기반 리스트 정의한 구조체
{
   LData arr[LIST_LEN];    // 리스트의 저장소인 배열 선언
   int numOfData;          // 저장된 데이터수 선언
   int curPosition;        //  데이터 참조위치 선언
}ArrayList;

typedef ArrayList List;      // ArrayList 형인 List 선언
```

ArrayList.h 파일에 정의

배열을 이용한 순차 리스트의 ADT 구현 – II

SUNG KYUN KWAN UNIVERSITY

- 리스트의 ADT 구현
  - 리스트 초기화 : void ListInit(List * plist);
    - 리스트 생성 후 제일 먼저 호출되는 함수
    - 초기화할 리스트의 주소 값을 인자로 전달
  - 리스트에 데이터 저장(삽입) : void Linsert(List * plist, Ldata data);
    - 리스트에 데이터 자장, 삽입할 매개변수 data를 리스트에 저장
  - 저장된 데이터의 탐색 및 탐색 초기화 : int Lfirst(List * plist, LData * pdata);
    - 데이터의 참조를 위해 초기화가 진행
    - 현재의 위치를 새로 설정(탐색의 첫번째 과정)
    - 첫 번째 데이터가 pdata가 가리키는 메모리에 저장하여 main에서 사용
    - 참조 성공시 TRUE(1), 실패 시 FALSE(0) 반환

배열을 이용한 순차 리스트의 ADT 구현 – III

SUNG KYUN KWAN UNIVERSITY

- 리스트의 ADT 구현
  - 다음 데이터의 참조(반환) : int LNext(List * plist, LData * pdata);
    - 순차적인 참조를 위해 반복 호출(리스트에 포함된 데이터 수)
    - 참조된 데이터의 다음 데이터가 pdata가 가리키는 메모리에 저장
    - LFirst함수 이후에 호출하여 다음 데이터 참조
    - 참조 성공 시 TRUE(1), 실패 시 FALSE(0) 반환

  - 데이터 삭제 (바로 이전에 참조(반환)가 이루어진) : LData LRemove(List * plist);
    - LFrist 혹은 LNext 함수의 마지막 반환 데이터를 삭제(호출 : 특정 데이터 매칭시)
    - 삭제된 데이터는 반환
    - 반복 호출을 허용하지 않음
    - 리스트 원소의 위치 조정 및 개수 조정

  - 현재 저장되어 있는 데이터 수를 반환 : int LCount(List * plist);
    - 리스트에 저장되어 있는 데이터의 수를 반환

배열을 이용한 순차 리스트의 ADT 구현 – IV

SUNG KYUN KWAN UNIVERSITY

- C로 순차 리스트 구현 방식

ArrayList.h

순차 리스트 자료구조의
헤더파일 정의(지금까지)

ArrayList.c

순차 자료구조의
소스파일 정의

ListMain.c

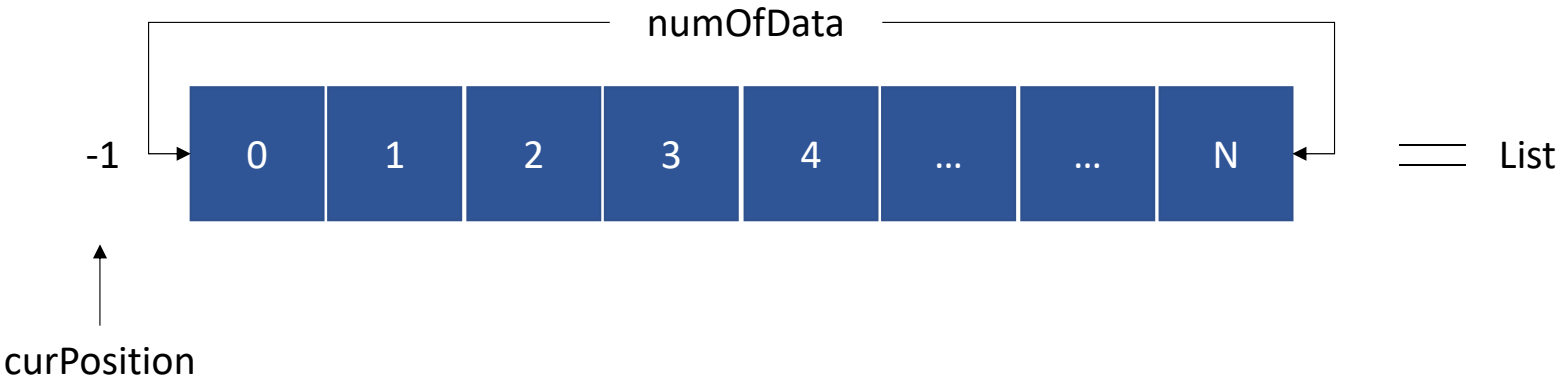
순차 리스트 main 함수
정의

배열을 이용한 순차 리스트의 ADT 구현 – V

SUNG KYUN KWAN UNIVERSITY

- 리스트의 ADT 구현
  - 리스트 초기화 : void ListInit(List * plist);

```
void ListInit(List *plist) {
        plist->numOfData = 0;
        plist->curPosition = -1;  // arr의 index가 -1임을 가르키는 것은 아무런
                                  // 위치에도 참조하지 않았음을 의미함
}
```



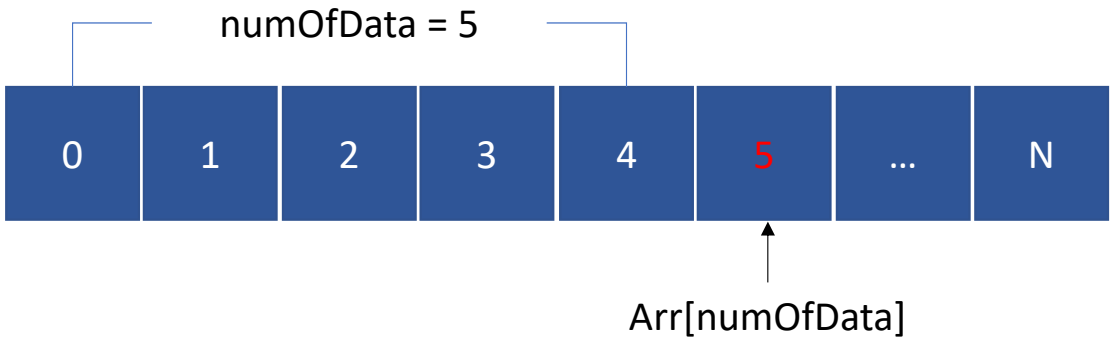
배열을 이용한 순차 리스트의 ADT 구현 – VI

SUNG KYUN KWAN UNIVERSITY

- 리스트의 ADT 구현
  - 리스트에 데이터 저장(삽입) : void Linsert(List * plist, Ldata data);

```
void LInsert(List * plist, LData data) {
        if (plist->numOfData > LIST_LEN) {    // 주어진 공간보다 크다면(공간없음)
                puts( " 저장이 불가능합니다. " );
                return;
         }

         plist->arr[plist->numOfData] = data;       // 빈공간에 데이터 저장
         (plist->numOfData)++;                      // 저장된 데이터 수 증가
}
```



배열을 이용한 순차 리스트의 ADT 구현 – VII

SUNG KYUN KWAN UNIVERSITY

- 리스트의 ADT 구현
  - 저장된 데이터의 탐색 및 탐색 초기화 : int Lfirst(List * plist, LData * pdata);

```
int LFirst(List *plist, LData *pdata) {
        if (plist->numOfData == 0)          // 저장된 데이터가 없는 경우
                return FALSE;

        plist->curPosition = 0;             // 현재 위치를 새로 설정
        *pdata = plist->arr[0];             // 첫 번째 위치에 있는 공간에 데이터 확인
        return TRUE;
}
```

배열을 이용한 순차 리스트의 ADT 구현 – VIII

SUNG KYUN KWAN UNIVERSITY

- 리스트의 ADT 구현
  - 다음 데이터의 참조(반환) : int LNext(List * plist, LData * pdata);

```
int LNext(List *plist, LData *pdata) {
        if (plist->curPosition >= (plist->numOfData) - 1) // 참조할 데이터가 없는 경우
                return FALSE;

        (plist->curPosition)++;   // 이전위치에서 다음위치로 이동
        *pdata = plist->arr[plist->curPosition];  // 현재 위치의 데이터를 pdata로 저장

        return TRUE;
}
```

배열을 이용한 순차 리스트의 ADT 구현 – IX

SUNG KYUN KWAN UNIVERSITY

- 리스트의 ADT 구현
  - 데이터 삭제 (바로 이전에 참조(반환)가 이루어진) : LData LRemove(List * plist);

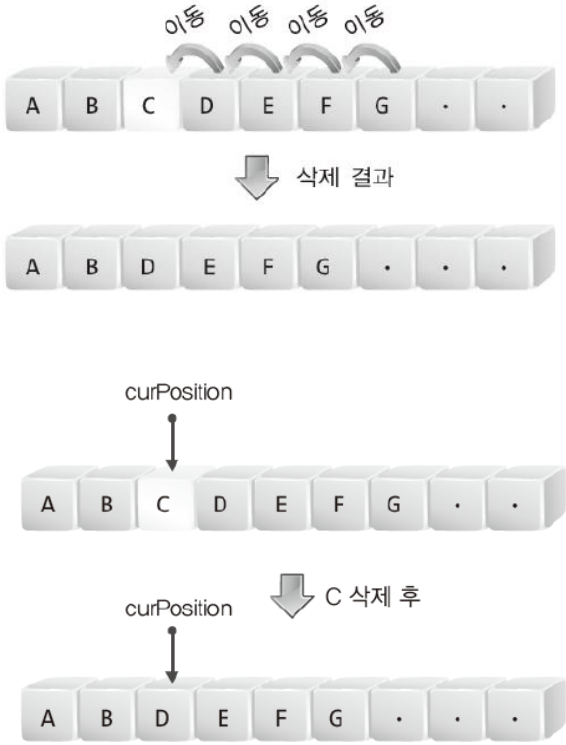
```
LData LRemove(List *plist) {
        int rpos = plist->curPosition;   // 삭제할 데이터 index 값
        int num = plist->numOfData;
        int i;

        LData rdata = plist->arr[rpos];  // 삭제할 데이터 임시 저장

        for (I = rpos; I < num – 1; i++)  // 삭제할 곳으로 데이터 이동진행
                plist->arr[i] = plist->arr[I + 1];

        (plist->numOfData)--;   // 데이터 수 감소
        (plist->curPosition)--;  //   참조위치 감소

        return rdata;
}
```



배열을 이용한 순차 리스트의 ADT 구현 – X

SUNG KYUN KWAN UNIVERSITY

- 리스트의 ADT 구현
  - 현재 저장되어 있는 데이터 수를 반환 : int LCount(List * plist);

```
int LCount(List *plist) {
        return plist->numOfData;    // 전체 데이터 수를 반환
}
```

배열을 이용한 순차 리스트의 ADT 활용

SUNG KYUN KWAN UNIVERSITY

- 구조체 변수의 주소값을 저장하여 좌표저장 방법 구현
  - 새롭게 정의된 구조체 Point를 기존의 ArrayList로 지정한 리스트에 주소값을 저장하여 주소를 가르키는 구조체의 값을 지정하여 출력하도록 함.
  - 선언된 구조체는 동적할당을 통해 생성함으로 삭제시 free 함수를 통해 삭제해야 함.

```
typedef Point * LData;  // 주소를 담기 위해 헤더파일 변경

typedef struct __ArrayList{
        LData arr[LIST_LEN];
        int numOfData;
        int curPosition;
}ArrayList;
```

```
ppos = LRemove(&list);  // 삭제 후 선언된 주소를 저장
free(ppos); // 삭제된 데이터를 품은 공간 없앰
```