1. 개요

LinkedList의 기본적인 부분인 리스트 생성, 데이터 추가/제거, 리스트 전체 출력 기능 외에 Python의 기능 중에서 입력 값에 따라 변수의 자료형을 스스로 결정해주는 기능과 유사하게 입력 값에 따라 정수형, 실수형, 문자열 세 가지 자료형으로 리스트에 저장, 우선 숫자형, 문자열 순으로 정렬 후 각각 숫자형 입력 값들은 크기에 따라 오름차순으로, 문자열 입력 값들은 첫번째 문자 값에 따라 사전순으로 정렬하도록 구현하였습니다. 리스트의 기본적인 기능은 강의자료에 작성해두신 코드와 다른 형태로 작성할 수가 없어 강의자료를 따라 작성하되 왜 그런 식으로 작성하였는지 보고서, 주석 등을 통해 설명하는 식으로 작성하였습니다.

1. 프로그램 구조 설명
   1. 함수에 대한 설명
2. LinkedList

void insert(ListNodePtr\*);

* obtainInput 함수를 이용하여 새로 생성한 노드에 데이터를 입력
* compare 함수를 이용하여 정렬 규칙에 따라 리스트 내에 노드를 삽입할 위치로 이동
* 포인터 값들을 수정하여 새로 생성한 노드를 기존 리스트에 연결

ListNodePtr delete(ListNodePtr\*);

* 리스트 내의 각각의 노드에 대해 삭제하려는 데이터와 자료형이 일치하는지 우선 검사
* 자료형이 일치하는 경우, 자료형에 따라 포인터를 형변환한 후 \* 연산자를 이용하여 저장된 값을 비교, 값이 일치하는 경우 노드 삭제
* LinkedList는 이전 노드를 거쳐야 다음 노드로 이동할 수 있으므로 순차 탐색만 가능
* 리스트를 탐색할 때, currentPtr->nextPtr->data 이런 식으로 포인터 하나를 이용하여 처리할 수도 있지만 이전에 탐색한 노드를 저장하는 previousPtr 을 사용하면 훨씬 이해하기 쉽게 작성할 수 있으므로 포인터 두 개를 이용하여 리스트 탐색 수행
* 경계 조건인 첫번째 노드의 경우는 다른 방식으로 처리해주어야 함

void printList(ListNodePtr);

* ListNode 구조체 안에 기록해둔 자료형에 따라 포인터를 형변환한 뒤 값을 출력

int compare(ListNodePtr, ListNodePtr);

* 숫자형, 문자열 순으로 정렬 후 각각 숫자형 입력 값들은 크기에 따라 오름차순으로, 문자열 입력 값들은 첫번째 문자 값에 따라 사전순으로 정렬하도록 구현

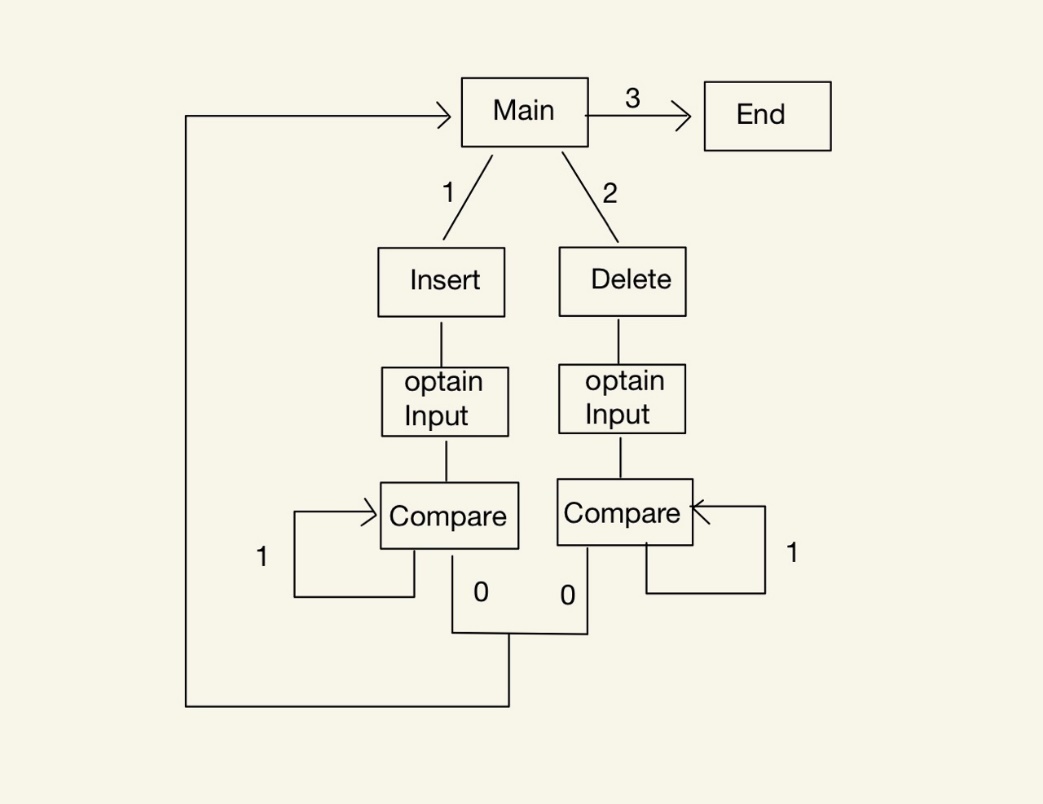
1. ObtainInput

void obtainInput(ListNodePtr newPtr);

* 입력한 값에서 문자를 하나씩 읽어와 buffer에 저장
* 문자가 하나씩 입력될 때마다 extendBuffer 함수를 이용하여 버퍼 크기 1씩 증가
* 문자 각각에 대해 숫자로만 이루어져 있다면 정수형, 숫자 + ‘.’ 으로 이루어져 있으면 실수형, 숫자, ‘.’ 외에 다른 값이 포함되어 있으면 문자열로 변환하여 노드에 자료형과 함께 저장

char\* extendBuffer(char\* buffer, size\_t\* sizePtr);

* 인자로 전달받은 버퍼의 크기보다 1바이트 큰 버퍼를 새로 생성하고 memcpy 함수를 사용하여 기존의 값을 복사한 후에 새로 생성한 버퍼를 반환함
* Malloc은 시간이 오래 걸리는 작업이기 때문에 사용횟수를 줄이기 위해 보통 버퍼의 크기를 정해둔 뒤에 필요하다면 두배씩 늘리는 방식 등을 사용하는 것으로 알고 있지만, malloc, memcpy 등 메모리 관련 함수를 연습해보기 위해 이렇게 작성하였음
  1. 다이어그램



1. 실행 결과

실패 1)

텍스트이(가) 표시된 사진

자동 생성된 설명

데이터를 입력받을 때 getchar 함수를 이용하여 입력받도록 작성하였다. Scanf(“%d”, &choice)로 정수를 입력받은 후에 입력 버퍼에 남아있던 ‘\n’을 처리해주지 않아 NULL(0)이 입력되거나 뒤이어 다른 값들을 입력할 경우 버퍼에 남아있던 개행문자 때문에 obtainInput 함수에서 입력값의 자료형을 문자열로 인식하여 숫자값으로는 앞에 와야 하는 값들이 리스트의 뒤로 가는 등 여러가지 문제가 생기는 것을 디버거를 통해 확인하였다.

Scanf 아래에 getchar를 추가하여 해결하였다.

실패 2)

텍스트이(가) 표시된 사진

자동 생성된 설명

리스트에서 삭제하려는 값을 찾지 못하는 문제가 발생하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

디버거를 통해 확인해보니 의도하지 않은 부분에서 함수가 종료되는 문제가 발생하였다.

각각의 자료형 별로 입력값에서 세가지 경우 (정수형, 실수형, 문자열), 비교할 기존의 리스트 값에서 세가지 경우 모두를 중첩 if문을 통해 직접 작성하였더니 코드 확인이 쉽지 않아 저런 실수가 생겼다. 그래서 두 노드의 데이터를 비교하여 다음 노드로 이동해야 할 경우 1, 그 곳에서 멈추는 경우 0을 반환하는 함수 compare를 작성하여 코드를 알아보기 쉽도록 수정하였다.

성공)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

입력값에 따라 정수형, 실수형, 문자열 자료형으로 구분하여 형변환 후 노드에 저장하였다.

그 후 숫자형 값들은 값의 크기에 따라 정렬하였다. 문자열 값들은 숫자형 값들의 뒤로 이동시킨 후 첫번째 문자의 값에 따라 사전순으로 정렬하였다.

1. 고찰

디버거로 한줄씩 확인해보기 전에 겉으로 보기에는 입력된 숫자값들이 크기에 따라 정렬되지 않아 정렬에 관련된 코드에서 문제가 발생한 것이 아닌가 하고 생각하였다. 하지만 수정 후에도 계속 같은 문제가 발생하여 코드 전체를 디버거를 통해 한 줄씩 실행하였다.

발견한 원인은 메뉴 선택 후에 남아있던 개행 문자가 숫자값 데이터 앞에 “\n123” 이런 식으로 남아있어 숫자값으로 리스트에 기록되지 않고 문자열 값으로 리스트에 기록되어 리스트의 뒤쪽으로 이동하지만 리스트 출력 시에는 숫자 형태 그대로 나오기 때문에 정렬이 되지 않은 것 처럼 보이게 되는 것이었다.

데이터 입력 시, 개행 문자가 버퍼에 남아 원하지 않는 값이 입력되는 것은 C프로그래밍 실습 강의 등 기본적인 C언어 실습 시간에서 짚고 넘어가는 문제이다. 그런데 그런 사소한 부분을 놓쳐서 전혀 예상하지 못한 문제가 발생하는 것을 보니 사소한 부분까지 꼼꼼하게 확인하는 것이 프로그램을 짜는데 중요한 부분이라는 것을 알게 되었다.

비교를 위한 compare 함수를 활용하여 코드 길이를 줄이자 코드를 한 눈에 확인할 수 있게 되어 괄호 범위 등으로 생기는 오류를 해결할 수 있었다. 이 과정에서 함수로 작성이 가능한 동작들은 최대한 함수로 작성하는 것이 오류를 줄일 수 있는 한 방법임을 알게 되었다.

1. 프로그램 소스 코드

LinkedList.h

#ifndef \_\_LINKEDLIST\_H\_\_

#define \_\_LINKEDLIST\_H\_\_

struct listNode {

// 저장된 데이터의 자료형 기록하기 위한 변수

// 0 : 정수형, 1 : 실수형, 2 : 문자열

int type;

void\* dataPtr;

struct listNode\* nextPtr;

};

typedef struct listNode ListNode;

typedef ListNode\* ListNodePtr;

void insert(ListNodePtr\*);

ListNodePtr delete(ListNodePtr\*);

int isEmpty(ListNodePtr);

void printList(ListNodePtr);

int compare(ListNodePtr, ListNodePtr);

#endif // !\_\_LINKEDLIST\_H\_\_

LinkedList.c

#include <stdio.h>

#include <stdlib.h>

#include "LinkedList.h"

#include "ObtainInput.h"

void insert(ListNodePtr\* sPtr)

{

ListNodePtr newPtr, previousPtr, currentPtr;

newPtr = malloc(sizeof(ListNode));

// 메모리가 정상적으로 할당되었는지 검사

if (newPtr != NULL) {

// 새로 메모리를 할당받은 노드에 데이터 입력

obtainInput(newPtr);

newPtr->nextPtr = NULL;

previousPtr = NULL;

currentPtr = \*sPtr;

// compare 함수 이용하여 노드를 삽입할 위치로 이동

while (compare(newPtr, currentPtr)) {

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

// 경계 조건 (리스트의 가장 끝) 처리

if (previousPtr == NULL) {

newPtr->nextPtr = \*sPtr;

\*sPtr = newPtr;

}

// 일반적인 경우

else {

previousPtr->nextPtr = newPtr;

newPtr->nextPtr = currentPtr;

}

}

else

printf("Not inserted. No memory available.\n");

}

ListNodePtr delete(ListNodePtr\* sPtr)

{

ListNodePtr previousPtr, currentPtr, tempPtr, inputPtr;

// 사용자가 삭제하기를 원하는 값을 입력받기 위해 임시 노드 생성

// optainInput 함수를 그대로 사용하기 위해 따로 입력받기 위한 코드를 작성하지 않고

// 노드를 하나 만들어 사용하였다.

inputPtr = malloc(sizeof(ListNode));

if (inputPtr) {

// 입력

obtainInput(inputPtr);

// compare 함수는 리스트의 다음 노드로 이동할지 여부만을 알 수 있으므로

// 저장된 값을 비교하기 위한 코드는 직접 작성하였음

// 리스트의 첫번째 노드는 일반적인 경우가 아니므로

// 별도로 처리해주었음

if (inputPtr->type == (\*sPtr)->type) {

if ((inputPtr->type == 0 && \*((int\*)(inputPtr->dataPtr)) == \*((int\*)((\*sPtr)->dataPtr))) ||

(inputPtr->type == 1 && \*((double\*)(inputPtr->dataPtr)) == \*((double\*)((\*sPtr)->dataPtr))) ||

(inputPtr->type == 2 && !strcmp((char\*)(inputPtr->dataPtr), (char\*)((\*sPtr)->dataPtr)))) {

tempPtr = \*sPtr;

\*sPtr = (\*sPtr)->nextPtr;

free(tempPtr->dataPtr);

free(tempPtr);

return inputPtr;

}

}

previousPtr = \*sPtr;

currentPtr = (\*sPtr)->nextPtr;

// compare 함수를 이용하여 이동

// currentPtr에 저장된 값이 삭제하려는 값과 같거나 작은 경우

while (compare(inputPtr, currentPtr)) {

previousPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

}

// currentPtr에 저장된 값과 삭제하려는 값이 같은지 확인

if (currentPtr != NULL) {

if ((inputPtr->type == 0 && \*((int\*)(inputPtr->dataPtr)) == \*((int\*)(currentPtr->dataPtr))) ||

(inputPtr->type == 1 && \*((double\*)(inputPtr->dataPtr)) == \*((double\*)(currentPtr->dataPtr))) ||

(inputPtr->type == 2 && !strcmp((char\*)(inputPtr->dataPtr), (char\*)(currentPtr->dataPtr)))) {

tempPtr = currentPtr;

previousPtr->nextPtr = currentPtr->nextPtr;

free(tempPtr->dataPtr);

free(tempPtr);

return inputPtr;

}

}

}

return NULL;

}

int isEmpty(ListNodePtr sPtr)

{

return sPtr == NULL;

}

void printList(ListNodePtr currentPtr)

{

if (currentPtr == NULL)

printf("List is empty.\n\n");

else {

printf("The list is:\n");

while (currentPtr != NULL) {

// 노드에 저장된 데이터의 자료형에 따라 포인터 형변환 후 참조하여 출력

if (currentPtr->type == 0)

printf("%d --> ", \*((int\*)(currentPtr->dataPtr)));

else if (currentPtr->type == 1)

printf("%f --> ", \*((double\*)(currentPtr->dataPtr)));

else

printf("%s --> ", ((char\*)(currentPtr->dataPtr)));

currentPtr = currentPtr->nextPtr;

}

printf("NULL\n\n");

}

}

int compare(ListNodePtr newNode, ListNodePtr oldNode)

{

if (oldNode) {

if (newNode->type == 2) { // 추가하려는 값이 문자열

if (oldNode->type != 2) // 리스트의 값이 숫자형일 경우

return 1;

else if (((char\*)(newNode->dataPtr))[0] > ((char\*)(oldNode->dataPtr))[0]) // 사전순 정렬

return 1;

}

else if (newNode->type == 0) { // 추가하려는 값이 정수형

if (oldNode->type == 0 && \*((int\*)(newNode->dataPtr)) > \*((int\*)(oldNode->dataPtr))) // 리스트의 값이 정수형

return 1;

else if (oldNode->type == 1 && \*((int\*)(newNode->dataPtr)) > \*((double\*)(oldNode->dataPtr))) // 리스트의 값이 실수형

return 1;

}

else { // 추가하려는 값이 실수형

if (oldNode->type == 0 && \*((double\*)(newNode->dataPtr)) > \*((int\*)(oldNode->dataPtr)))

return 1;

else if (oldNode->type == 1 && \*((double\*)(newNode->dataPtr)) > \*((double\*)(oldNode->dataPtr)))

return 1;

}

}

return 0;

}

ObtainInput.h

#ifndef \_\_OBTAININPUT\_H\_\_

#define \_\_OBTAININPUT\_H\_\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "LinkedList.h"

void obtainInput(ListNodePtr newPtr);

char\* extendBuffer(char\* buffer, size\_t\* sizePtr);

int isDigit(char c);

int isPeriod(char c);

#endif // !\_\_OBTAININPUT\_H\_\_

ObtainInput.c

#include "ObtainInput.h"

// 데이터를 입력받아 노드에 저장

void obtainInput(ListNodePtr newPtr)

{

char c, \* buffer = NULL;

size\_t size = 0;

int isString = 0;

int \_isPeriod = 0;

// 개행문자 입력 전까지 각각의 문자에 대해

while ((c = getchar()) != '\n') {

// 버퍼의 크기 1 증가

buffer = extendBuffer(buffer, &size);

// 값 입력

buffer[size - 1] = c;

}

// 버퍼의 크기 1 증가시킨 후

buffer = extendBuffer(buffer, &size);

// 문자열의 끝을 나타내는 \0 입력

buffer[size - 1] = '\0';

for (int i = 0; i < size - 1; i++) {

// 정수형과 실수형을 구분하기 위해 입력값에 '.' 포함 여부 확인

if (isPeriod(buffer[i]))

\_isPeriod = 1;

// 입력값에 숫자, '.' 외에 다른 값이 존재한다면 문자열

else if (!isDigit(buffer[i])) {

isString = 1;

break;

}

}

if (isString) {

newPtr->type = 2;

newPtr->dataPtr = buffer;

}

else if (\_isPeriod) {

double\* dataPtr = malloc(sizeof(double));

if (dataPtr)

\*dataPtr = atof(buffer);

newPtr->type = 1;

newPtr->dataPtr = dataPtr;

free(buffer);

}

else {

int\* dataPtr = malloc(sizeof(int));

if (dataPtr)

\*dataPtr = atoi(buffer);

newPtr->type = 0;

newPtr->dataPtr = dataPtr;

free(buffer);

}

}

// 버퍼의 크기 1 증가

char\* extendBuffer(char\* buffer, size\_t\* sizePtr)

{

// 기존 버퍼보다 1바이트 큰 버퍼 생성

char\* newBuffer = malloc(\*sizePtr + 1);

// 메모리 이상없이 할당되었는지 확인

if (newBuffer)

// 기존의 값 복사

memcpy(newBuffer, buffer, \*sizePtr);

if (buffer)

// 기존의 메모리 할당 해제

free(buffer);

(\*sizePtr)++;

return newBuffer;

}

int isDigit(char c)

{

return c >= '0' && c <= '9';

}

int isPeriod(char c)

{

return c == '.';

}

Main.c

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include "LinkedList.h"

void instructions();

void instructions()

{

printf("Enter your choice:\n"

" 1 to insert an element into the list.\n"

" 2 to delete an element from the list.\n"

" 3 to end.\n");

}

// main 함수는 강의자료의 형식 그대로 사용

int main()

{

ListNodePtr startPtr = NULL;

int choice, dump;

instructions();

printf("? ");

dump = scanf("%d", &choice);

while (choice != 3) {

switch (choice) {

case 1:

printf("Enter integer, float, or string: ");

insert(&startPtr);

printList(startPtr);

break;

case 2:

if (!isEmpty(startPtr)) {

ListNodePtr tempPtr;

printf("Enter integer, float, or string to be deleted: ");

if (tempPtr = delete(&startPtr)) {

if (tempPtr->type == 0)

printf("%d deleted.\n", \*((int\*)(tempPtr->dataPtr)));

else if (tempPtr->type == 1)

printf("%f deleted.\n", \*((double\*)(tempPtr->dataPtr)));

else

printf("%s deleted.\n", (char\*)(tempPtr->dataPtr));

printList(startPtr);

}

else

printf("Not found.\n\n");

}

else

printf("List is empty.\n\n");

break;

default:

printf("Invalid choice.\n\n");

instructions();

break;

}

printf("? ");

dump = scanf("%d", &choice);

}

printf("End of run.\n");

return 0;

}

1. 자료 출처

리스트의 동작 부분은 강의자료 참고하여 작성하였음.