**Doubly LinkedList의 delete\_position 구현**

5645640 제갈건 04-18

1. 실행코드

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include <stdlib.h>

#include<malloc.h>

typedef int element;

// 이중 연결리스트 노드 타입

typedef struct DoubleListNode {

element data;

struct DoubleListNode\* Leftlink; //앞 노드를 가리키는 포인터

struct DoubleListNode\* Rightlink; // 뒷 노드를 가리키는 포인터

}DoubleListNode;

/\*

// 이중 연결리스트 초기화

void init(DoubleListNode\* phead) {

phead->Leftlink = phead;

phead->Rightlink = phead;

}\*/

// 이중 연결리스트의 노드 출력

void print\_dlist(DoubleListNode\* phead) {

DoubleListNode\* p;

printf("head->");

for (p = phead->Rightlink; p != phead; p = p->Rightlink) {

printf("|%d|<->", p->data);

}

printf("NULL\n");

}

// 새로운 데이터를 노드 before의 오른쪽에 삽입

void dinsert(DoubleListNode\* before, element data) {

DoubleListNode\* newnode = (DoubleListNode\*)malloc(sizeof(DoubleListNode));

newnode->data = data; // 노드 생성

newnode->Leftlink = before; // 새 노드의 왼쪽 링크를 before노드와 연결

newnode->Rightlink = before->Rightlink; //새 노드의 오른쪽 링크를 before노드의 오른쪽 링크가 가리키는 노드와 연결

before->Rightlink->Leftlink = newnode; // before노드의 오른쪽 링크가 가리키는 노드의 왼쪽 링크를 새 노드와 연결

before->Rightlink = newnode; //before노드의 오른쪽 링크를 새 노드와 연결

}

// 이중 포인터를 이용하여 리스트의 앞에 노드를 추가하는 함수

void dinsert\_p(DoubleListNode\*\* h, int position, int data) {

int k = 1;

DoubleListNode\* newNode, \* temp;

newNode = (DoubleListNode\*)malloc(sizeof(DoubleListNode));

newNode->data = data;

if (position == 0) { // insertion at beginning

/\*if((\*h) == NULL){

\* newNode->Leftlink = null;

\* newNode->rightlink = \*h;

\*}else{

\* newNode->Leftlink = null;

\* newNode->rightlink = \*h;

\* (\*h)->leftlink = newNode;

\* \*h = newNode;

\* }return;

\*/

newNode->Leftlink = NULL;

newNode->Rightlink = \*h;

(\*h)->Rightlink->Leftlink = newNode;

\*h = newNode;

print\_dlist(\*h);

return;

}

temp = \*h;

while ((k < position ) && temp->Rightlink != NULL) {

temp = temp->Rightlink;

k++;

}

if (temp->Rightlink == NULL) {// insert at the end

newNode->Rightlink = temp->Rightlink;

newNode->Leftlink = temp;

temp->Rightlink = newNode;

}

else {// insert in the middle

newNode->Rightlink = temp->Rightlink;

newNode->Leftlink = temp;

temp->Rightlink->Leftlink = newNode;

temp->Rightlink = newNode;

}

print\_dlist(\*h);

}

DoubleListNode\* dinsert\_first(DoubleListNode\* head, element data) {

DoubleListNode\* newnode = (DoubleListNode\*)malloc(sizeof(DoubleListNode));

newnode->data = data;

if (head == NULL) {

newnode->Leftlink = newnode;

newnode->Rightlink = newnode;

}

else {

newnode->Leftlink = head->Leftlink;

newnode->Rightlink = head;

head->Leftlink->Rightlink = newnode;

head->Leftlink = newnode;

}

head = newnode; // head 포인터 변경

print\_dlist(head);

return head;

}

//원하는 위치의 노드 오른쪽에 삽입

void dinsert\_pos(DoubleListNode\* head, element data, int count) {

DoubleListNode\* newnode = (DoubleListNode\*)malloc(sizeof(DoubleListNode));

DoubleListNode\* before = head;

newnode->data = data; // 노드 생성

for (int i = 0; i < count && before->Rightlink != head; i++) { // before->Rightlink != head 노드의 끝에서 멈추는 코드

before = before->Rightlink; // Leftlink를 사용하면 반대 방향으로 이동

}

newnode->Leftlink = before; // 새 노드의 왼쪽 링크를 before노드와 연결

newnode->Rightlink = before->Rightlink; //새 노드의 오른쪽 링크를 before노드의 오른쪽 링크가 가리키는 노드와 연결

before->Rightlink->Leftlink = newnode; // before노드의 오른쪽 링크가 가리키는 노드의 왼쪽 링크를 새 노드와 연결

before->Rightlink = newnode; //before노드의 오른쪽 링크를 새 노드와 연결

printf("left: 0x%p, me: 0x%p, right: 0x%p\n", newnode->Leftlink, newnode, newnode->Rightlink);

print\_dlist(head);

}

//마지막 노드에 새로운 노드 삽입

void dinsert\_last(DoubleListNode\* head, element data) {

DoubleListNode\* newnode = (DoubleListNode\*)malloc(sizeof(DoubleListNode));

DoubleListNode\* before = head;

newnode->data = data; // 노드 생성

while (before->Rightlink != head) { //노드끝까지 이동

before = before->Rightlink;

}

newnode->Leftlink = before; // 새 노드의 왼쪽 링크를 before노드와 연결

newnode->Rightlink = before->Rightlink; //새 노드의 오른쪽 링크를 before노드의 오른쪽 링크가 가리키는 노드와 연결

before->Rightlink->Leftlink = newnode; // before노드의 오른쪽 링크가 가리키는 노드의 왼쪽 링크를 새 노드와 연결

before->Rightlink = newnode; //before노드의 오른쪽 링크를 새 노드와 연결

printf("left: 0x%p, right: %p\n" , before, before->Rightlink);

print\_dlist(head);

}

// 이중 포인터를 이용하여 리스트의 앞에 노드를 삭제하는 함수

void ddelete\_p(DoubleListNode\*\* h, int position) {

int k = 1;

DoubleListNode\* removed, \* temp;

removed = (DoubleListNode\*)malloc(sizeof(DoubleListNode));

if (position == 0) { // insertion at beginning

removed = (\*h)->Rightlink;

if (removed == \*h) return;

(\*h)->Rightlink = removed->Rightlink;

(\*h)->Leftlink = removed->Leftlink;

free(removed);

return;

}

temp = \*h;

do {

temp = temp->Rightlink;

k++;

} while ((k <= position - 1) && temp->Rightlink->Rightlink != \*h);

if (temp->Rightlink->Rightlink == \*h) {// delete at the end

removed = temp->Rightlink;

temp->Rightlink = \*h;

(\*h)->Leftlink = removed->Leftlink;

free(removed);

}

else {// insert in the middle

removed = temp->Rightlink;

temp->Rightlink = removed->Rightlink;

removed->Rightlink->Leftlink = removed->Leftlink;

free(removed);

}

print\_dlist(\*h);

}

// 노드 removed를 삭제(head의 바로 오른쪽 노드만 삭제가능)

void ddelete\_first(DoubleListNode\* head) {

DoubleListNode\* removed = (DoubleListNode\*)malloc(sizeof(DoubleListNode));

removed = head->Rightlink;

if (removed == head) return;

head->Rightlink = removed->Rightlink;

head->Leftlink = removed->Leftlink;

free(removed);

print\_dlist(head);

}

void ddelete\_pos(DoubleListNode\* head, int count) {

DoubleListNode\* before = head;

DoubleListNode\* removed = NULL;

for (int i = 0; i < count - 1 && before->Rightlink->Rightlink != head; i++) { // before->Rightlink != head 노드의 끝에서 멈추는 코드

before = before->Rightlink; // Leftlink를 사용하면 반대 방향으로 이동

}

removed = before->Rightlink;

if(removed == head) return;

before->Rightlink = removed->Rightlink;

removed->Rightlink->Leftlink = removed->Leftlink;

free(removed);

print\_dlist(head);

}

// 마지막 노드 삭제

void ddelete\_last(DoubleListNode\* head) {

DoubleListNode\* before = head;

DoubleListNode\* removed = NULL;

while (before->Rightlink->Rightlink != head) { //노드를 마지막 직전까지 이동

before = before->Rightlink;

}

removed = before->Rightlink;

if (removed == head) return;

before->Rightlink = head;

head->Leftlink = removed->Leftlink;

free(removed);

print\_dlist(head);

}

// 메인

int main(void) {

DoubleListNode\* head = NULL;

printf("추가 단계\n");

for (int i = 0; i < 3; i++) {

head = dinsert\_first(head, i); // head오른쪽에 insert

printf("right: 0x%p\n", head->Rightlink);

//printf("\n");

}

printf("\n");

for (int i = 2; i < 6; i++) {

dinsert\_pos(head, i, 0); // head오른쪽에 insert

//printf("right: 0x%p\n", head->Rightlink);

//printf("\n");

}

/\*

printf("위치 마지막에 44 insert\n");

dinsert\_pos(head, 44, 10); //10번째 노드(마지막 위치)에 44를 insert

printf("포인터 3번째 노드에 100 insert\n");

dinsert\_p(&head, 3, 100);

printf("마지막 노드에 10 insert\n");

dinsert\_last(head, 10); // 마지막 노드에 10 insert

\*/

printf("\n삭제단계\n첫번쨰 노드 delete\n");

ddelete\_first(head);

printf("\n2번째 위치 delete\n");

ddelete\_pos(head, 2); // 2번째 노드 삭제

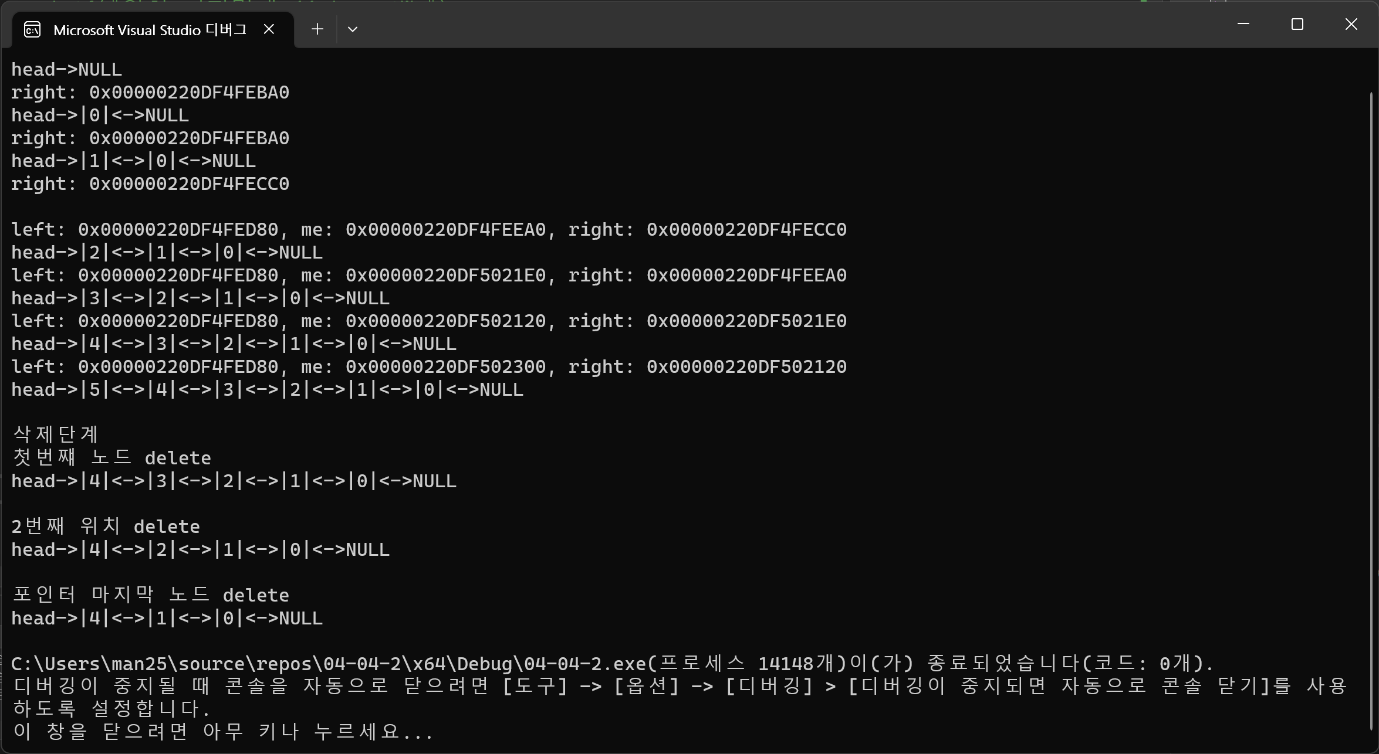
printf("\n포인터 마지막 노드 delete\n");

ddelete\_p(&head, 200);

return 0;

}

1. 실행화면



1. 고찰

이중 포인터로 delete를 구현하는 것이 어려울 것이라 생각하였지만 기존에 사용하던 head를 (\*h)로 전환하고 ADT를 사용할 때 head를 &head로 바꾸니 바로 실행이 되었습니다. 리스트를 구현할 수록 포인터와 리스트 링크를 다루는 것에 익숙해 지는 것 같았습니다.