5645640 제갈건 0321

1. 실행코드

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include <stdlib.h>

#include<malloc.h>

#define max\_size 100 // 리스트의 최대크기

typedef struct ListNode {

int data = 0; // 실제 자료

struct ListNode\* link = NULL; // 다음 노드를 가리키는데 사용하는 포인터

}ListNode;

// 리스트가 비어 있는 지 확인하는 함수

bool is\_empty(ListNode\* head) {

if (head == NULL) {

printf("NULL : No element in the list!!\n");

printf("list is empty\n");

return true;

}

return false;

}

// 리스트가 가득 차 있는 지 확인하는 함수

bool is\_full(ListNode\* head) {

int size = 0;

for (ListNode\* p = head; p != NULL; p = p->link) size++;

if (size == max\_size) {

printf("list is full\n");

return true;

}

return false;

}

void print\_list(ListNode\* head) {

// 리스트가 비어있다면 print 하지마라

if (is\_empty(head)) return;

for (ListNode\* p = head; p != NULL; p = p->link) printf("%d->", p->data); // 검색 방식

printf("NULL\n");

}

//맨 처음에 노드 삽입(head에 삽입?)

ListNode\* insert\_first(ListNode\* head, int value) {

// 리스트가 가득 찼다면 insert하지마라

if (is\_full(head)) return head;

ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));

p->data = value;

p->link = head;

head = p; //head를 p로 바꾸고 p->link를 이전 head에 연결

print\_list(head);

return head; // head리턴

}

//pre뒤에 새로운 노드 삽입(pre를 모른다는 문제점이 있음, 우리는 head만 알고있다.)

ListNode\* insert(ListNode\* head, ListNode\* pre, int value) {

// 리스트가 가득 찼다면 insert하지마라

if(is\_full(head)) return head;

ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));

p->data = value;

p->link = pre->link; //pre가 가리키는 값을 p로 전환(pre가 가리키는 다음 값을 p에 연결)

pre->link = p; //p의 주소를 pre에 연결(pre가 가리키는 다음 값을 p로 전환)

print\_list(head);

return head; // pre가 head면 head값이 바뀌므로 head 리턴

}

void insert\_last(ListNode\* head, int value) {

ListNode\* p2 = head;

// 리스트가 가득 찼다면 insert하지마라

if (is\_full(head)) return ;

ListNode\* lst = (ListNode\*)malloc(sizeof(ListNode));

lst->data = value;

// 마지막 노드 들고오기

for (ListNode\* p = head; p != NULL; p = p->link) p2 = p;

p2->link = lst;

lst->link = NULL;

print\_list(head);

}

ListNode\* insert\_position(ListNode\* head) {

// 리스트가 가득 찼다면 insert하지마라

if (is\_full(head)) return head;

int pos = 0, value = 0;

//insert하려는 위치를 입력한다

printf("what insert list position? ");

scanf\_s("%d", &pos);

// pos가 올바르게 입력 되었는 지 검사, 리스트의 크기보다 크면 마지막 노드로 이동

if (pos > max\_size) {

printf("number over list max\_size\n");

return head;

}

//insert 하려는 item을 입력한다

printf("insert item: ");

scanf\_s("%d", &value);

ListNode\* p = head, \*q = NULL;

ListNode\* lst = (ListNode\*)malloc(sizeof(ListNode));

lst->data = value;

if (pos == 0) { //첫번째 노드에 insert할 때

lst->link = head; // head가 가리키는 값을 p로 전환

head = lst; //p의 주소를 head에 연결

print\_list(head);

return head;

}

// 해당 위치까지 이동

int i = 0;

while((i < pos) && (p != NULL)){

q = p;

p = p->link;

i++;

}

//해당 위치에 노드 삽입

q->link = lst;

lst->link = p;

print\_list(head);

return head;

}

ListNode\* delete\_first(ListNode\* head) {

// 리스트가 비어있다면 delete 하지마라

if (is\_empty(head)) {

print\_list(head);

return head;

}

ListNode\* removed;

removed = head;

head = removed->link;

free(removed);

print\_list(head);

return head;

}

void Delete\_last(ListNode\* head) {

// 리스트가 비어있다면 delete 하지마라

if (is\_empty(head)) return;

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

ListNode\* lst = (ListNode\*)malloc(sizeof(ListNode));

for (ListNode\* p = head; p->link != NULL; p = p->link) {

// 마지막의 바로 앞 노드 들고오기

lst = p;

// 마지막 노드 들고오기

removed = p->link;

}

lst->link = NULL;

free(removed);

print\_list(head);

}

ListNode\* Delete\_position(ListNode\* head) {

// 리스트가 비어있다면 delete 하지마라

if (is\_empty(head)) return head;

ListNode\* p = head, \*lst = NULL;

int pos = 0;

//삭제하려는 위치 입력

printf("what delete list position? ");

scanf\_s("%d", &pos);

//num가 올바르게 입력되었는 지 검사, 리스트의 크기보다 크면 마지막 노드로 이동

if (pos < 0 || pos > max\_size) {

printf("number is out of range\n");

return head;

}

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

// 해당 위치가 가장 앞에 있으면 다음을 실행

if (pos == 0) {

removed = head;

head = removed->link;

free(removed);

print\_list(head);

return head;

}

// 가장 앞이 아니면 해당 위치까지 이동

int i = 0;

while((i < pos) && (removed != NULL)){

lst = p;

p = p->link;

removed = p->link;

i++;

}

//노드가 가장 마지막에 있으면 해당 함수 실행

if (removed == NULL) {

lst ->link = NULL;

free(p);

free(removed);

print\_list(head);

return head;

}

// 노드 삭제

lst->link = p->link;

free(p);

print\_list(head);

return head;

}

//모든 노드를 삭제하는 함수

void DeleteLinkedList(ListNode\* head) {

ListNode\* auxilaryNode, \* iterator;

iterator = head;

while (iterator) {

printf("삭제할 노드의 데이터 : %d\n", iterator->data);

auxilaryNode = iterator->link;

free(iterator);

iterator = auxilaryNode;

}

head = NULL;

}

//검색 함수

ListNode\* search\_list(ListNode\* head, int data){

ListNode\* p = head;

while (p != NULL) {

if (p->data == data) return p;

p = p->link;

}

return NULL;

}

int main() {

ListNode\* head = NULL;

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

head = insert\_first(head, i \* 10);

}

head = insert\_position(head);

head = Delete\_position(head);

DeleteLinkedList(head);

return 0;

}

1. 실행화면

텍스트이(가) 표시된 사진

자동 생성된 설명

(위치가 리스트의 크기를 넘어선 경우: 리스트의 마지막에 insert, delete한다.)

텍스트이(가) 표시된 사진

자동 생성된 설명

(위치가 처음을 가리키는 경우: 리스트의 시작 부분에 insert, delete한다.)

텍스트이(가) 표시된 사진

자동 생성된 설명

(위치가 어느 한 부분 경우: 리스트의 해당 부분에 insert, delete한다.)

텍스트이(가) 표시된 사진

자동 생성된 설명

(위치가 어느 한 부분 경우: 리스트의 해당 부분에 insert, delete한다.)

1. 고찰

Delete\_position을 구현하는 과정에서 어려움을 겪었습니다. 마지막 노드가 삭제될 경우, 마지막에서 두 번째 노드가 삭제되지 않는 오류, 어떤 한 부분의 노드가 삭제될 경우 두 번째 노드가 삭제되지 않는 오류, 끝에서 두 번째와 두 번째 노드가 삭제될 경우 중간의 노드가 삭제되지 않는 등 여러 가지 오류가 발생했습니다. 하지만 많은 시행착오를 거쳐 모든 경우에 대해 정상적으로 작동하는 결과를 얻을 수 있었습니다.