**4장 단순연결 리스트를 이용한 리스트 ADT(개정 3판)**

#include <stdio.h>

#include <stdlib.h>

typedef int element;

typedef struct ListNode { // 노드 타입

element data;

struct ListNode \*link;

} ListNode;

// 오류 처리 함수

void error(char \*message)

{

fprintf(stderr, "%s\n", message);

exit(1);

}

// 노드 pre 뒤에 새로운 노드 삽입

ListNode\* insert\_next(ListNode \*head, ListNode \*pre, element value)

{

ListNode \*p = (ListNode \*)malloc(sizeof(ListNode)); //(1)

p->data = value; //(2)

p->link = pre->link; //(3)

pre->link = p; //(4)

return head; //(5)

}

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

{

ListNode \*p = (ListNode \*)malloc(sizeof(ListNode)); //(1)

p->data = value; // (2)

p->link = head; // 헤드 포인터의 값을 복사 //(3)

head = p; // 헤드 포인터 변경 //(4)

return head; // 변경된 헤드 포인터 반환

}

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

{

ListNode \*temp = head;

ListNode \*p = (ListNode \*)malloc(sizeof(ListNode)); //(1)

p->data = value;

p->link = NULL;

// (2)

if (head == NULL) // 공백 리스트이면

head = p;

else {

while (temp->link != 0)

temp = temp->link;

temp->link = p;

}

return head; //헤드 포인터 반환

}

// pre가 가리키는 노드의 다음 노드를 삭제한다.

ListNode\* delete\_next(ListNode \*head, ListNode \*pre) // pre가 정확이 있다고 가정(NULL이 아니고)

{

ListNode \*removed;

removed = pre->link;

pre->link = removed->link; // (2)

free(removed); // (3)

return head; // (4)

}

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

{

ListNode \*removed;

if (head == NULL) error("삭제할 항목이 없음");

removed = head; // (1)

head = removed->link; // (2)

free(removed); // (3)

return head; // (4)

}

ListNode\* delete\_last(ListNode \*head)

{

ListNode \*temp = head;

ListNode \*prevTemp;

ListNode \*removed;

if (head == NULL) error("삭제할 항목이 없음");

if (temp->link == NULL) { // 하나의 노드

removed = temp->link;

free(removed);

return NULL;

}

else { // 둘 이상의 노드

while (temp->link != NULL) {

prevTemp = temp;

temp = temp->link;

}

removed = temp;

prevTemp->link = NULL;

free(removed);

return head; // 그대로

}

}

void print\_list(ListNode \*head)

{

ListNode \*p;

for (p = head; p != NULL; p = p->link)

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

printf("NULL \n");

}

ListNode \*search(ListNode \*head, int x)

{

ListNode \*p;

p = head;

while( p != NULL ){

if( p->data == x ) return p; // 탐색 성공

p = p->link;

}

return NULL; // 탐색 실패일 경우 NULL 반환

}

ListNode \*concat(ListNode \*head1, ListNode \*head2)

{

ListNode \*p;

if( head1 == NULL ) return head2;

else if( head2 == NULL ) return head1;

else {

p = head1;

while( p->link != NULL )

p = p->link;

p->link = head2;

return head1;

}

}

ListNode \*reverse(ListNode \*head)

{

// 순회 포인터로 p, q, r을 사용

ListNode \*p, \*q, \*r;

p = head; // p는 역순으로 만들 리스트

q = NULL; // q는 역순으로 만들 노드

while (p != NULL){

r = q; // r은 역순으로 된 리스트. r은 q, q는 p를 차례로 따라간다.

q = p ;

p = p->link ;

q->link =r; // q의 링크 방향을 바꾼다.

}

return q; // q는 역순으로 된 리스트의 헤드 포인터

}

int main(void)

{

ListNode \*head1 = NULL, \*head2;

int i;

head1 = insert\_last(head1, 10);

print\_list(head1);

head1 = insert\_last(head1, 20);

print\_list(head1);

head1 = insert\_last(head1, 30);

print\_list(head1);

head2 = reverse(head1);

print\_list(head2);

head2 = delete\_last(head2);

print\_list(head2);

head1 = NULL;

head1 = insert\_first(head1, 100);

head1 = insert\_first(head1, 200);

print\_list(head1);

head1 = concat(head1, head2);

print\_list(head1);

}