자료구조 Homework #6



충북대학교 컴퓨터공학과 2022040014 오재식

/\*

\* singly linked list

\*

\* Data Structures

\* School of Computer Science

\* at Chungbuk National University

\*/

#include<stdio.h>

#include<stdlib.h>

/\*구조체 선언\*/

typedef struct Node {

int key;

struct Node\* link;

} listNode;

typedef struct Head {

struct Node\* first;

}headNode;

/\* Head 포인터 선언 \*/

headNode\* initialize(headNode\* h);

int freeList(headNode\* h);

int insertFirst(headNode\* h, int key);

int insertNode(headNode\* h, int key);

int insertLast(headNode\* h, int key);

int deleteFirst(headNode\* h);

int deleteNode(headNode\* h, int key);

int deleteLast(headNode\* h);

int invertList(headNode\* h);

void printList(headNode\* h);

int main()

{

char command;

int key;

headNode\* headnode=NULL;

printf("------ 2022040014 --- ohjaesik -------");

do{

printf("----------------------------------------------------------------\n");

printf(" Singly Linked List \n");

printf("----------------------------------------------------------------\n");

printf(" Initialize = z Print = p \n");

printf(" Insert Node = i Delete Node = d \n");

printf(" Insert Last = n Delete Last = e\n");

printf(" Insert First = f Delete First = t\n");

printf(" Invert List = r Quit = q\n");

printf("----------------------------------------------------------------\n");

printf("Command = ");

scanf(" %c", &command);

switch(command) {

case 'z': case 'Z':

headnode = initialize(headnode);

break;

case 'p': case 'P':

printList(headnode);

break;

case 'i': case 'I':

printf("Your Key = ");

scanf("%d", &key);

insertNode(headnode, key);

break;

case 'd': case 'D':

printf("Your Key = ");

scanf("%d", &key);

deleteNode(headnode, key);

break;

case 'n': case 'N':

printf("Your Key = ");

scanf("%d", &key);

insertLast(headnode, key);

break;

case 'e': case 'E':

deleteLast(headnode);

break;

case 'f': case 'F':

printf("Your Key = ");

scanf("%d", &key);

insertFirst(headnode, key);

break;

case 't': case 'T':

deleteFirst(headnode);

break;

case 'r': case 'R':

invertList(headnode);

break;

case 'q': case 'Q':

freeList(headnode);

break;

default:

printf("\n >>>>> Concentration!! <<<<< \n");

break;

}

}while(command != 'q' && command != 'Q');

return 1;

}

headNode\* initialize(headNode\* h) {

/\* headNode가 NULL이 아니면 다 메모리를 해제함 \*/

if(h != NULL)

freeList(h);

/\* headNode가 가리키는 값에 headNode만큼의 메모리를 할당 \*/

headNode\* temp = (headNode\*)malloc(sizeof(headNode));

temp->first = NULL;

return temp;

}

int freeList(headNode\* h){

/\* h에 할당된 메모리 해제

\* headNode의 모든 메모리 해제.

\*/

listNode\* p = h->first;

listNode\* prev = NULL;

while(p != NULL) {

prev = p;

p = p->link;

free(prev);

}

free(h);

return 0;

}

/\* 새로운 node값을 추가 \*/

int insertNode(headNode\* h, int key) {

listNode\* node = (listNode\*)malloc(sizeof(listNode)); //메모리 할당

node->key = key;

node->link = NULL;

if (h->first == NULL) //처음 값이 없을시 처음 값에 삽입

{

h->first = node;

return 0;

}

listNode\* n = h->first;

listNode\* trail = h->first;

/\* key를 알맞은 위치에 넣기 위한 반복문 \*/

while(n != NULL) {

if(n->key >= key) {

/\* 삽입된 값이 가장 작은 값일 경우 노드의 맨 앞으로 삽입 \*/

if(n == h->first) {

h->first = node;

node->link = n;

} else { /\* 삽입된 값이 trail의 값보다 크고 n의 값보다 작은 위치에 삽입 \*/

node->link = n;

trail->link = node;

}

return 0;

}

trail = n;

n = n->link;

}

/\* 반복문이 종료되도 값이 삽입되지 않았을 경우 가장 뒤에 값을 삽입 \*/

trail->link = node;

return 0;

}

/\*\*

\* list의 가장 마지막에 key를 삽입

\*/

int insertLast(headNode\* h, int key) {

listNode\* node = (listNode\*)malloc(sizeof(listNode)); //메모리 할당

node->key = key;

node->link = NULL;

if (h->first == NULL) //처음 값이 없을 때 처음 값에 삽입

{

h->first = node;

return 0;

}

listNode\* n = h->first;

while(n->link != NULL) { //마지막 노드를 찾는 반복문

n = n->link;

}

n->link = node;

return 0;

}

/\*\*

\* list의 제일 첫번째 위치에 key 삽입

\*/

int insertFirst(headNode\* h, int key) {

listNode\* node = (listNode\*)malloc(sizeof(listNode)); //메모리 할당

node->key = key;

node->link = h->first; //원래 처음 값을 삽입된 값에 링크에 연결

h->first = node; // 새로 삽입된 값을 처음 링크에 연결

return 0;

}

/\*\*

\* list에서 key에 해당하는 값을 삭제

\*/

int deleteNode(headNode\* h, int key) {

if (h->first == NULL)

{

printf("nothing to delete.\n");

return 0;

}

listNode\* n = h->first;

listNode\* trail = NULL;

/\* key에 해당하는 노드를 찾는 과정 \*/

while(n != NULL) {

if(n->key == key) {

/\* 삭제할 노드가 처음 값일 때 두번째 값을 처음 값으로 설정\*/

if(n == h->first) {

h->first = n->link;

} else { /\* 삭제할 노드 앞의 노드와 뒤의 노드를 연결 \*/

trail->link = n->link;

}

free(n);

return 0;

}

trail = n;

n = n->link;

}

/\* key값을 찾지 못했을 때 \*/

printf("cannot find the node for key = %d\n", key);

return 0;

}

/\*\*

\* list의 가장 마지막 값을 삭제

\*/

int deleteLast(headNode\* h) {

if (h->first == NULL)

{

printf("nothing to delete.\n");

return 0;

}

listNode\* n = h->first;

listNode\* trail = NULL;

/\* 처음노드가 마지막 노드일때 노드를 삭제 \*/

if(n->link == NULL) {

h->first = NULL;

free(n);

return 0;

}

/\* 가장 마지막 노드를 찾는 과정 \*/

while(n->link != NULL) {

trail = n;

n = n->link;

}

/\* 마지막 노드를 가리키는 link를 삭제하고 메모리 해제 \*/

trail->link = NULL;

free(n);

return 0;

}

/\*\*

\* list의 첫 번재 값을 삭제

\*/

int deleteFirst(headNode\* h) {

if (h->first == NULL)

{

printf("nothing to delete.\n");

return 0;

}

listNode\* n = h->first;

h->first = n->link;

free(n);

return 0;

}

/\*\*

\* 현재 리스트의 순서를 역방향으로 바꿈

\*/

int invertList(headNode\* h) {

if(h->first == NULL) {

printf("nothing to invert...\n");

return 0;

}

listNode \*n = h->first;

listNode \*trail = NULL;

listNode \*middle = NULL;

while(n != NULL){ // 가리키는 링크의 방향을 반대로 바꿈.

trail = middle;

middle = n;

n = n->link;

middle->link = trail;

}

h->first = middle;

// head가 가리키는 값을 원래 가장 마지막의 값을 가리키도록 함.

return 0;

}

void printList(headNode\* h) {

int i = 0;

listNode\* p;

printf("\n---PRINT\n");

if(h == NULL) {

printf("Nothing to print....\n");

return;

}

p = h->first;

while(p != NULL) {

printf("[ [%d]=%d ] ", i, p->key);

p = p->link;

i++;

}

printf(" items = %d\n", i);

}

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트, 명판, 득점판이(가) 표시된 사진

자동 생성된 설명

<https://github.com/ohjaesik/homework6>