5645640 제갈건 0427

1. 실행코드

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include <stdlib.h>

#include<malloc.h>

#include<time.h>

#define max\_size 100000 // 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

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

return head; // head리턴

}

ListNode\* insert\_position(ListNode\* head, int pos, int value) {

// 리스트가 가득 찼다면 insert하지마라

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

// 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;

}

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

if (pos == 0) { //첫번째 노드에 insert할 때

head->data += value;

//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->data += value;

//print\_list(head);

return head;

}

ListNode\* Delete\_position(ListNode\* head, int pos) {

// 리스트가 비어있다면 delete 하지마라

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

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

//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);

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);

return head;

}

// 노드 삭제

lst->link = p->link;

free(p);

return head;

}

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 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");

}

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

return head;

}

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

DoubleListNode\* before = head;

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

}

before->data += data;

}

// 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);

}

}

int main(void) {

int arry[10000], y = 0;

ListNode\* head = NULL;

DoubleListNode\* dhead = NULL;

// 삽입

clock\_t array\_start = clock(); // 함수의 시작 시간 저장

for (int i = 0; i < 10000; i++) {

arry[i] = i;

for (int t = 0; t < 500; t++) y++; // 시간 측정을 위한 더미코드

}

clock\_t array\_finish = clock(); // 함수의 종료시간 저장

double array\_duration = (double)(array\_finish - array\_start) / CLOCKS\_PER\_SEC; // Iteration함수의 소요시간 계산

printf("Array에서 10000개의 값을 넣는 시간 : %lf초\n", array\_duration);

clock\_t list\_start = clock(); // 함수의 시작 시간 저장

for (int i = 0; i < 10000; i++) {

head = insert\_first(head, i);

for (int t = 0; t < 500; t++) y++;

}

clock\_t list\_finish = clock(); // 함수의 종료시간 저장

double list\_duration = (double)(list\_finish - list\_start) / CLOCKS\_PER\_SEC; // Iteration함수의 소요시간 계산

printf("Linked List에서 10000개의 값을 넣는 시간 : %lf초\n", list\_duration);

clock\_t dlist\_start = clock(); // 함수의 시작 시간 저장

for (int i = 0; i < 10000; i++) {

dhead = dinsert\_first(dhead, i);

for (int t = 0; t < 500; t++) y++;

}

clock\_t dlist\_finish = clock(); // 함수의 종료시간 저장

double dlist\_duration = (double)(dlist\_finish - dlist\_start) / CLOCKS\_PER\_SEC; // Iteration함수의 소요시간 계산

printf("Doubly Linked List에서 10000개의 값을 넣는 시간 : %lf초\n", dlist\_duration);

printf("==========================================================================\n");

// Random Access

clock\_t array\_start2 = clock(); // 함수의 시작 시간 저장

for (int i = 0; i < 1000; i++) {

int j = rand() % 10000;

arry[j] += 1;

for (int t = 0; t < 500; t++) y--;

}

clock\_t array\_finish2 = clock(); // 함수의 종료시간 저장

double array\_duration2 = (double)(array\_finish2 - array\_start2) / CLOCKS\_PER\_SEC; // Iteration함수의 소요시간 계산

printf("Array에서 Random Access한 시간 : %lf초\n", array\_duration2);

clock\_t list\_start2 = clock(); // 함수의 시작 시간 저장

for (int i = 0; i < 1000; i++) {

int j = rand()%10000;

head = insert\_position(head, j, j);

for (int t = 0; t < 500; t++) y--;

}

clock\_t list\_finish2 = clock(); // 함수의 종료시간 저장

double list\_duration2 = (double)(list\_finish2 - list\_start2) / CLOCKS\_PER\_SEC; // Iteration함수의 소요시간 계산

printf("Linked List에서 Random Access한 시간 : %lf초\n", list\_duration2);

clock\_t dlist\_start2 = clock(); // 함수의 시작 시간 저장

for (int i = 0; i < 1000; i++) {

int j = rand() % 10000;

dinsert\_pos(dhead, j, j);

for (int t = 0; t < 500; t++) y--;

}

clock\_t dlist\_finish2 = clock(); // 함수의 종료시간 저장

double dlist\_duration2 = (double)(dlist\_finish2 - dlist\_start2) / CLOCKS\_PER\_SEC; // Iteration함수의 소요시간 계산

printf("Doubly Linked List에서 Random Access한 시간 : %lf초\n", dlist\_duration2);

printf("==========================================================================\n");

// 삭제

clock\_t array\_start3 = clock(); // 함수의 시작 시간 저장

for (int i = 0; i < 1000; i++) {

int j = rand()%10000;

arry[j] = -1;

for (int t = 0; t < 500; t++) y++;

}

clock\_t array\_finish3 = clock(); // 함수의 종료시간 저장

double array\_duration3 = (double)(array\_finish3 - array\_start3) / CLOCKS\_PER\_SEC; // Iteration함수의 소요시간 계산

printf("Array에서 1000개의 값을 삭제한 시간 : %lf초\n", array\_duration3);

clock\_t list\_start3 = clock(); // 함수의 시작 시간 저장

for (int i = 0; i < 1000; i++) {

int j = rand()%(1000 - i);

head = Delete\_position(head, j);

for (int t = 0; t < 500; t++) y--;

}

clock\_t list\_finish3 = clock(); // 함수의 종료시간 저장

double list\_duration3 = (double)(list\_finish3 - list\_start3) / CLOCKS\_PER\_SEC; // Iteration함수의 소요시간 계산

printf("Linked List에서 1000개의 값을 삭제한 시간 : %lf초\n", list\_duration3);

clock\_t dlist\_start3 = clock(); // 함수의 시작 시간 저장

for (int i = 0; i < 1000; i++) {

int j = rand() % (1000 - i);

ddelete\_p(&dhead, j);

for (int t = 0; t < 500; t++) y++;

}

clock\_t dlist\_finish3 = clock(); // 함수의 종료시간 저장

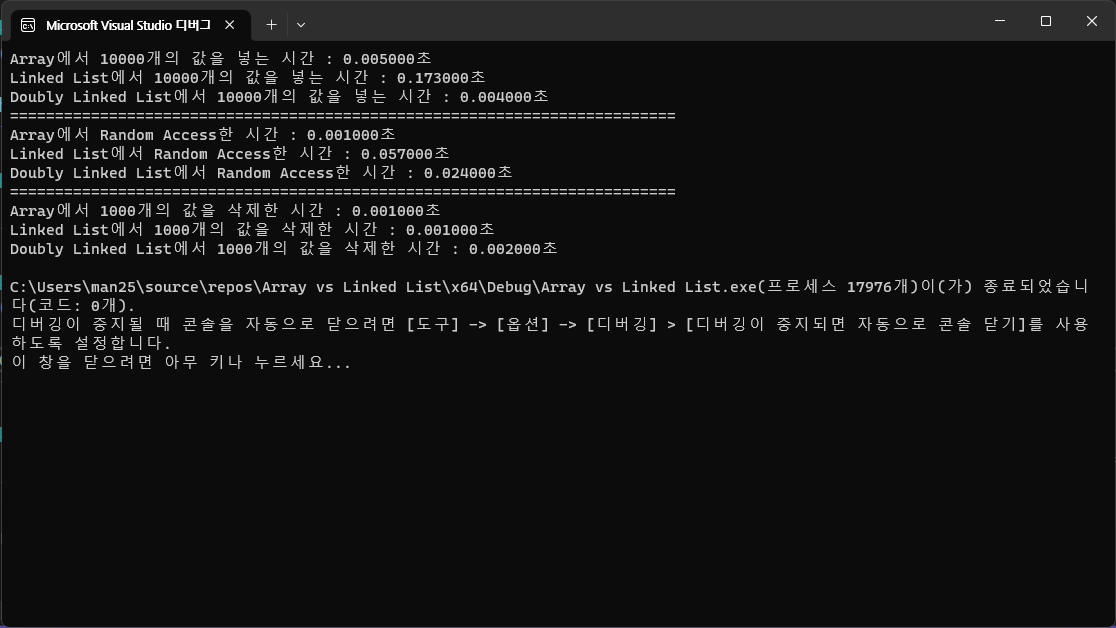
double dlist\_duration3 = (double)(dlist\_finish3 - dlist\_start3) / CLOCKS\_PER\_SEC; // Iteration함수의 소요시간 계산

printf("Doubly Linked List에서 1000개의 값을 삭제한 시간 : %lf초\n", dlist\_duration3);

return 0;

}

1. 실행화면



1. 고찰

메인 함수에 각각의 시간 측정 사이마다 더미 코드를 넣어 시간 측정의 길이를 늘렸습니다.