1. 코드 및 주석

#include <stdio.h>

#include <stdlib.h>

typedef struct \_node//node의 구조를 담은 구조체

{

int data;

struct \_node \* next;

} Node;

typedef int LData;

typedef struct \_linkedList//Linkedlist를 담은 구조체

{

Node \* head;

Node \* cur;

Node \* tail;

Node \* newNode;

} LinkedList;

typedef LinkedList List;

int main(void) {

List a;//a 리스트 선언

List b;//b 리스트 선언

int readData;

printf("a 리스트에 들어갈 자연수를 입력하세요\n");

a.head = (Node\*)malloc(sizeof(Node));//

a.tail = a.head;

while (1) {

printf("자연수 입력: ");

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

if (readData < 1)

break;

/\*\*\* 노드의 추가과정 \*\*\*/

a.newNode = (Node\*)malloc(sizeof(Node));//a.newNode에 새로운 메모리 동적할당 해서 주솟값 저장

a.newNode->data = readData;//a.newNode의 주솟값이 가리키는 곳의 data에 입력받은 데이터 값 저장

a.newNode->next = NULL;//next값은 아직 없으니 null

a.tail->next = a.newNode;//a.tail이 가르키는 next값에 새로운 노드 주솟값, 즉 newNode 이전 노드next에 newNode값 저장

a.tail = a.newNode;//이제 tail주소값은 newNode의 주솟값을 가르킴.

}

printf("\n");

printf("b 리스트에 들어갈 자연수를 입력하세요\n");

b.head = (Node\*)malloc(sizeof(Node));//더미노드 선언해서 head 에 주솟값

b.tail = b.head;

while (1) {

printf("자연수 입력: ");

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

if (readData < 1)

break;

/\*\*\* 노드의 추가과정 \*\*\*/

b.newNode = (Node\*)malloc(sizeof(Node));//a.newNode에 새로운 메모리 동적할당 해서 주솟값 저장

b.newNode->data = readData;//a.newNode의 주솟값이 가리키는 곳의 data에 입력받은 데이터 값 저장

b.newNode->next = NULL;//next값은 아직 없으니 null

b.tail->next = b.newNode;//a.tail이 가르키는 next값에 새로운 노드 주솟값, 즉 newNode 이전 노드next에 newNode값 저장

b.tail = b.newNode;//이제 tail주소값은 newNode의 주솟값을 가르킴.

}

printf("\n");

List c;

c.head = (Node\*)malloc(sizeof(Node));//더미노드 선언해서 head 에 주솟값

c.tail = c.head;

c.cur = c.head;

a.cur = a.head->next;//a.cur에 첫번째 노드 주소 저장

b.cur = b.head->next;//b.cur에 첫번째 노드 주소 저장

while ((a.cur->next != NULL) || (b.cur->next != NULL)) {//둘다 null이 될 때 까지 비교 실행

c.newNode = (Node\*)malloc(sizeof(Node));

c.cur->next = c.newNode;

c.cur = c.newNode;

if ((a.cur->data) <= (b.cur->data)) {

c.cur->data = a.cur->data;/\*c의 첫번째 노드 데이터에다가 a의 데이터가 담긴 노드의 주솟값을 저장. 예를 들어 data 1이 담긴 노드의

주소값이 100인 경우 c의 첫번째 노드 데이터에 100을 저장\*/

a.cur = a.cur->next;//a.cur의 값을 다음 노드 주소 값으로 바꿔줌.

}

else {

c.cur->data = b.cur->data;

b.cur = b.cur->next;

}

}

/\*남아있는 데이터들 정리\*/

if ((a.cur->next == NULL) && (b.cur->next == NULL)) {

if (a.cur->data < b.cur->data) {

c.newNode = (Node\*)malloc(sizeof(Node));

c.cur->next = c.newNode;

c.cur = c.newNode;

c.cur->data = a.cur->data;

c.newNode = (Node\*)malloc(sizeof(Node));

c.cur->next = c.newNode;

c.cur = c.newNode;

c.cur->data = b.cur->data;

}

else {

c.newNode = (Node\*)malloc(sizeof(Node));

c.cur->next = c.newNode;

c.cur = c.newNode;

c.cur->data = b.cur->data;

c.newNode = (Node\*)malloc(sizeof(Node));

c.cur->next = c.newNode;

c.cur = c.newNode;

c.cur->data = a.cur->data;

}

}

/\*c 리스트 출력하기\*/

c.cur = c.head;

int i = 0;

while (c.cur->next!=NULL) {

printf("%d ", c.cur->next->data);

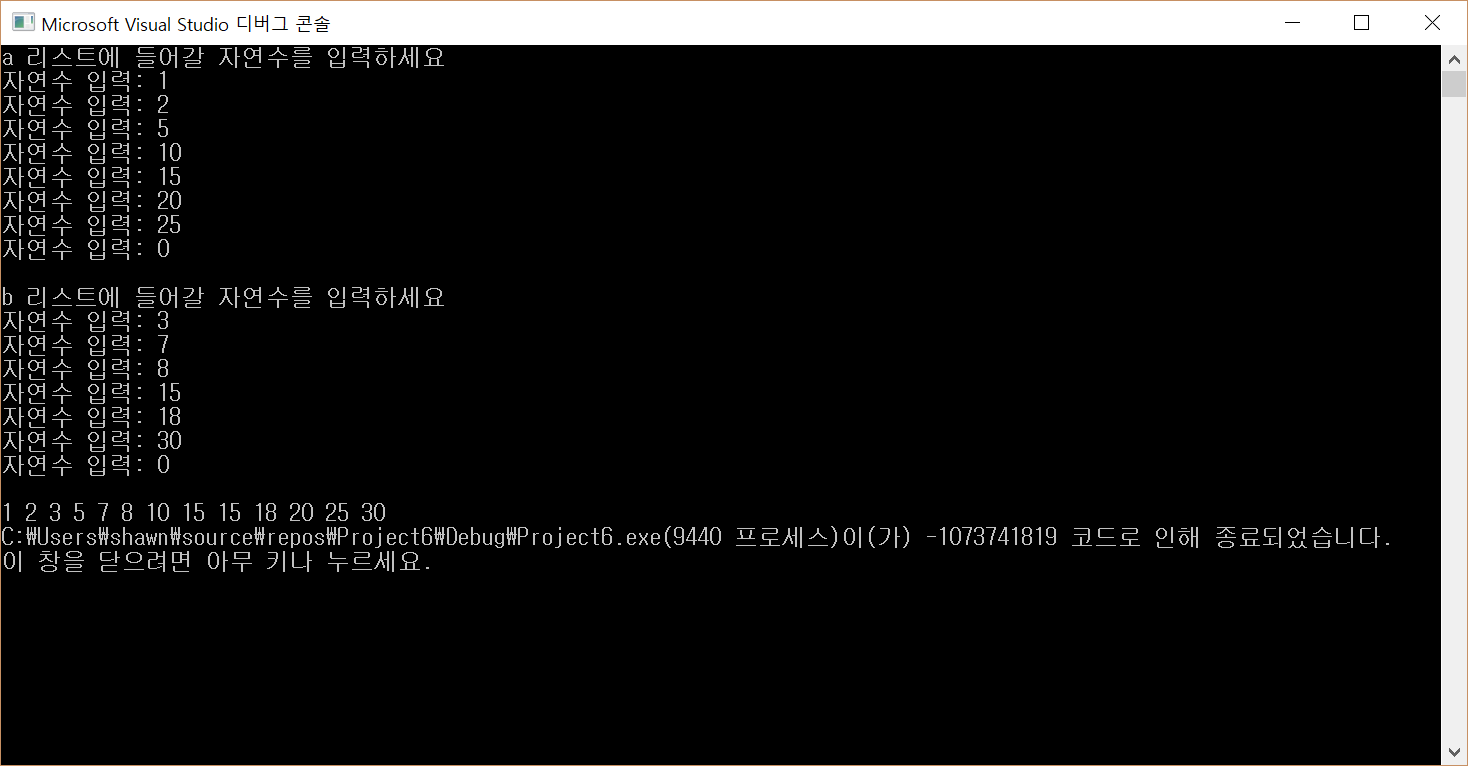
c.cur = c.cur->next;

}

return 0;

}

2. 결과 콘솔 창



3. Time complexity

시간복잡도 분석을 위해 가장 핵심적인 연산을 찾아야 한다. 이 코드에서 가장 핵심적인 연산은 비교연산이다. (코드: if ((a.cur->data) <= (b.cur->data))) 이 비교연산의 결과에 따라 나머지 연산들이 얼마나 진행될 지 결정된다. 각각 a리스트의 길이 n1, b리스트의 길이 n2라고 하면 최대 시간 복잡도는 O(n1+n2)이다. 연산 한번당 c 리스트에 정렬되어서 들어가기 때문이다.

4. 코드분석 이유 및 설명

1. 구조체 선언
2. typedef struct \_node//node의 구조를 담은 구조체
3. {
4. int data;
5. struct \_node \* next;
6. } Node;
7. typedef int LData;
8. typedef struct \_linkedList//Linkedlist를 담은 구조체
9. {
10. Node \* head;
11. Node \* cur;
12. Node \* tail;
13. Node \* newNode;
15. } LinkedList;

각각의 개별적인 노드를 선언함과 더불어 list a, list b를 구분할 수 있게 list 구조체도 선언했다.

2) 헤드포인터만 담은 노드 선언

a.head = (Node\*)malloc(sizeof(Node));//

처음 노드부터 데이터, 다음노드 링크로 구성하지 않고 맨 처음 노드는 옆의 노드의 주소값만 저장할 수 있도록 만들어 매번 list에 입력을 넣을때마다 head=NULL;인지 비교하지 않도록 구성하였다.

4) a,와 b 오름차순 정렬을 위한 코드

while ((a.cur->next != NULL) || (b.cur->next != NULL)) {//둘다 null이 될 때 까지 비교 실행

A 리스트와 b리스트 둘다 마지막 노드에 도착할 때 까지 비교를 진행.

if ((a.cur->data) <= (b.cur->data)) {

c.cur->data = a.cur->data;/\*c의 첫번째 노드 데이터에다가 a의 데이터가 담긴 노드의 주솟값을 저장. 예를 들어 data 1이 담긴 노드의

주소값이 100인 경우 c의 첫번째 노드 데이터에 100을 저장\*/

a.cur = a.cur->next;//a.cur의 값을 다음 노드 주소 값으로 바꿔줌.

}

else {

c.cur->data = b.cur->data;

b.cur = b.cur->next;

}

A와 b의 요소들을 하나씩 비교해 가며 c리스트에 넣고 인덱스 값을 증가시키는 방향으로 정렬

if ((a.cur->next == NULL) && (b.cur->next == NULL)) {

if (a.cur->data < b.cur->data) {

c.newNode = (Node\*)malloc(sizeof(Node));

c.cur->next = c.newNode;

c.cur = c.newNode;

c.cur->data = a.cur->data;

c.newNode = (Node\*)malloc(sizeof(Node));

c.cur->next = c.newNode;

c.cur = c.newNode;

c.cur->data = b.cur->data;

}

else {

c.newNode = (Node\*)malloc(sizeof(Node));

c.cur->next = c.newNode;

c.cur = c.newNode;

c.cur->data = b.cur->data;

c.newNode = (Node\*)malloc(sizeof(Node));

c.cur->next = c.newNode;

c.cur = c.newNode;

c.cur->data = a.cur->data;

}

}

A와 b 모두 마지막의 노드만 남았을 경우 처리