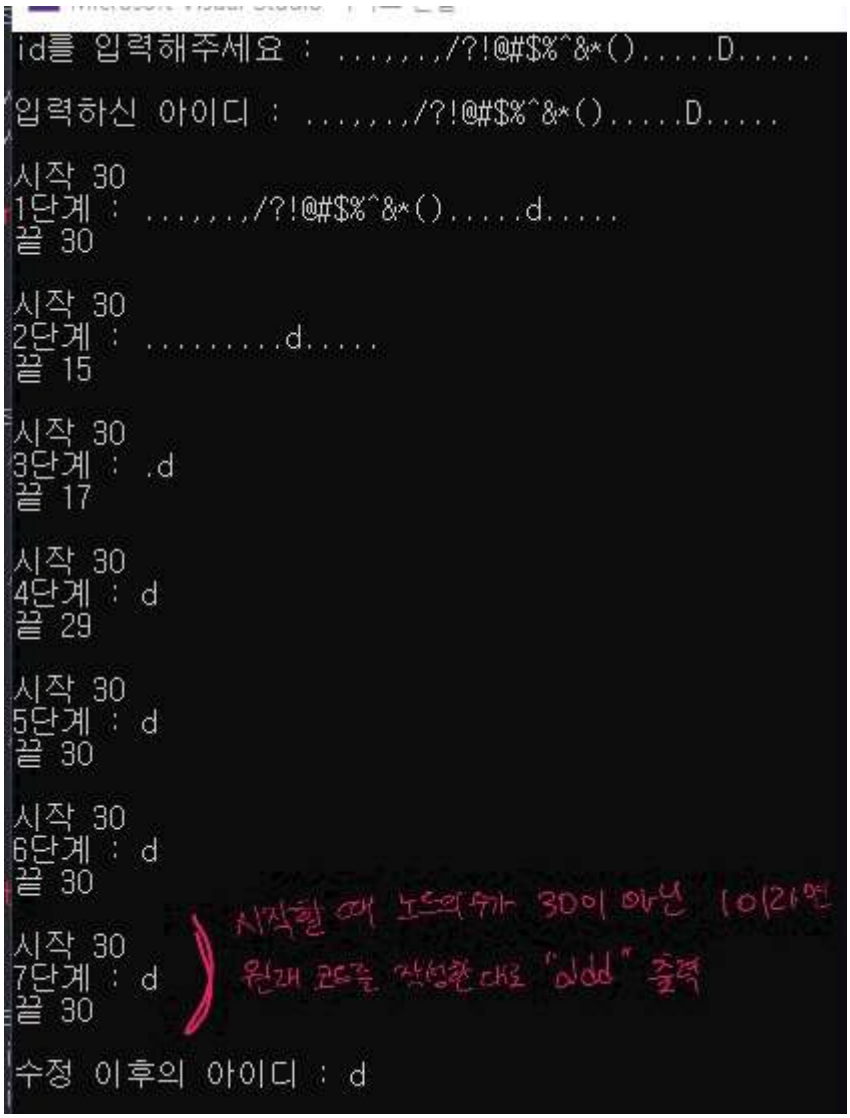
과제 3 : 신규 아이디 추천 프로그램 작성 202012468 김현서

1. 문제에 대한 자신의 알고리즘 설명(문제에 나와 있는 알고리즘 외에 다른 변경 사항이 있을 경우에만)

5번 문제 같은 경우 연결리스트에 노드가 없는 경우이기 때문에 원래같았으면 LCount 함수를 썼을 것 같지만, 왠지 모르게 오류가 났기 때문에 LFirst 함수로 맨 처음 노드를 불러오고 이 노드가 비어있다면 a를 집어넣는 형식으로 진행했습니다.

7번문제도 동일하게 while (LCount(&plist) < 3) 이라는 반복문을 써서 구하려다가, LCount 함수에 문제가 있어서 LFirst, LNext가 Null이냐 아니냐(int형으로는 0이냐 아니냐)를 이용해 코드를 작성했습니다.

각 단계별 함수마다 앞뒤에 printf("%d ", LCount(&plist));를 넣어서 노드의 수가 줄어드는지 아닌지를 확인해 보았으나, LRemove를 했을 때, LInsert, LChange를 했을 때 노드의 수가 증감을 해야 하지만 하지 않아서 이런 오류가 생긴 것 같습니다. 이 오류가 왜 생겼는지는 아직 모르는 바입니다. 처음에는 단지 같은 list, data를 써서 그런가 했지만, numOfData의 값은 이와 상관없이 메모리 주소 값에 반영되는 것이라고 알고 있기 때문에 이는 아닌 것 같았습니다. 아래가 제가 겪은 문제 중 하나입니다.



이런 식으로 만약 2단계에서 15로 끝났으면 3단계에서는 15개의 노드를 가지고 시작해야하는 게 맞는데 이 부분에서 오류가 생겼습니다.

그래서 현재 작성한 코드에는 LCount 함수가 쓰이지 않았습니다.

2. 자신이 사용한 연결리스트의 종류와 사용하게 된 이유를 설명(작성은 문제의 유형, 자신의 알고리즘을 기반으로 설명)

양방향 더미 리스트를 사용했습니다.

그 이유는 만약 양방향 연결 리스트를 쓰게 된다면 1~7단계동안 아이디를 수정하는 과정에서 앞뒤로 자유로운 이동이 가능해지기 때문에 단순 연결리스트를 쓰는 것보다 더 쉽게 아이디를 수정할 수 있을 것 같았기 때문입니다.

또한, 일반 양방향 리스트가 아닌 더미노드 양방향 연결리스트(head와 tail이 더미노드)를 쓴 이유는 첫 노드와 마지막 노드를 구하기도 쉽습니다.(첫 노드를 구하는 것은 1~7단계에서 5, 7단계 제외 모든 단계에서 사용, 마지막 노드를 구하는 것은 4, 6, 7단계에서 사용됨) 그리고 만약 연결리스트에서 모든 노드들이 삭제된다고 하여도 리스트의 형태는 남아 있을 수 있다는 생각이 들어서였습니다.

3. 수업시간에 배운 리스트의 ADT 변경 시 무엇을 왜 변경했는지

```
void LChange(List* plist, Data data)    // 같은 노드의 데이터를 변경하는 함수를 추가함
{
    Node* newNode = (Node*)malloc(sizeof(Node));    // newNode 동적할당으로 할당해주기
    newNode->data = data;   // newNode의 데이터는 LChange 함수 내의 data와 같다

    newNode->next = plist->cur->next;   // newNode의 next 포인터는 cur의 다음 노드를 가리킴
    plist->cur->next->prev = newNode;   // cur 다음 노드의 prev 포인터는 newNode를 가리킴

    newNode->prev = plist->cur; // newNode의 prev 포인터를 cur(cur의 next)을 가리킴
    plist->cur->next = newNode; // cur 노드의 next 포인터는 newNode를 가리킴

    plist->cur = plist->cur->next;  // cur 포인터를 newNode로 옮겨서 LNext를 실행했을 때 바로 다음 노드로 넘어가도록 함

    (plist->numOfData)++;   // newNode가 연결리스트에 추가되었으므로 1개 추가함
}

void LMultiply(List* plist)     // 현재 노드를 복사해주는 함수
{
    Data data = plist->cur->data;   // data에 현재 노드의 data 값을 집어넣음

    LChange(plist, data);   // cur 이후에 바로 data 값을 갖는 함수를 삽입해주는 함수
}
```

우선 LChange의 경우 1단계에서 LRemove 함수를 써서 대문자가 저장된 데이터를 갖고 있는 노드를 삭제한 후 LInsert 함수를 썼을 때 맨 뒤에 소문자 노드들이 추가되는 것을 보고 새로 만든 함수입니다. cur이 위치해있는 바로 다음 위치에 newNode를 삽입해 넣고, cur과 cur의 다음 노드 사이에 끼워넣습니다. 그리고 cur 포인터를 newNode로 옮겨서 newNode를 cur로 만들고, 노드의 수를 하나 추가합니다. data는 밖에서 입력해줘야 하는데, 1단계의 경우 data를 이전 노드의 pdata(대문자)를 소문자로 바꿔 temp=tolower(pdata)로 따로 저장해두고 밖에서 입력해줍니다.

다음으로 LMultiply의 경우 7단계에서 쓰이는데, 아이디기 2자 이하일 경우 LLast 값을 확인하고 LLast에 저장된 데이터를 가져와서 새 노드를 붙여넣는 그런 함수입니다. main 함수에서 LLast를 불러왔기 때문에 cur은 tail의 직전 노드가 될 것이고, 따라서 LMultiply 함수를 쓰면 맨 마지막 노드에 LLast와 같은 데이터를 갖는 노드를 추가할 수 있습니다.

LMultply 안에 LChange 함수를 넣어주었는데, 맨 마지막에 LLast와 같은 값을 갖는 노드를 넣어준다는 의미이기 때문에 LChange가 아닌 LInsert 함수를 넣고 Data 인자 안에 plist->cur->data를 의미하는 데이터만 넣어주면 됩니다.

```
int LPrev(List* plist, Data* pdata) // 이전 노드를 읽어들임
{
    if (plist->cur->prev == plist->head)    // cur의 이전 노드가 head를 가리키고 있는 경우 FALSE 반환
        return FALSE;

    plist->cur = plist->cur->prev;  // cur 포인터를 cur의 prev 포인터가 가리키고 있는 노드로 옮김
    *pdata = plist->cur->data;      // pdata를 cur의 data로 지정
    return TRUE;
}

int LLast(List* plist, Data* pdata)     // 새로 추가한 함수. 마지막 값을 알 수 있음
{
    if (plist->tail->prev == plist->head)   // tail의 이전 노드가 head라면 빈 연결리스트이기 때문에 FALSE 반환
        return FALSE;

    plist->cur = plist->tail->prev;     // cur 포인터를 tail의 이전 노드로 옮김
    *pdata = plist->cur->data;  // *pdata는 cur이 가리키고 있는 data를 의미함
    return TRUE;
}
```

다음으로는 LPrev 함수입니다. 이 함수는 LNext와 정반대로 cur을 next포인터가 아닌 prev 포인터를 써서 왼쪽으로 움직이는 함수입니다. 3단계에서 마침표가 2번 이상 연속된 부분들 하나의 마침표로 치환할 때 LRemove를 써준 후에 반복문으로 LNext가 써지고, 또 LNext가 써지면 만약 .이 3개 이상 있을 때는 이를 막지 못한 다는 것을 그림을 그려가며 이해하고 받아들였고, 이 현상을 막기 위해 LRemove를 써준 후 LPrev 함수를 한 번 써 줘서 반복문으로 LNext가 써졌을 때 우리가 봐야 하는 노드(만약 2개가 .이어서 1개가 지워졌다면 남은 .을 데이터로 갖는 노드)가 cur 노드가 된다는 걸 알았습니다.

마지막으로는 LLast 함수입니다. 이 함수는 head의 바로 다음 값을 알려주는 LFirst와는 정반대로 tail의 바로 전 노드를 알려주는 함수입니다. 이 함수를 사용해 4(LLast를 사용해 마지막 노드 값을 반환하고 '.'이 마지막에 위치할 때 LRemove), 6(4단계와 동일), 7(마지막 값을 확인하고 데이터를 복사해서 다음 노드로 추가하기)단계를 해결할 수 있습니다.