*// 이중 연결리스트 구현.*

#include <stdio.h>

#include <stdlib.h>

**typedef** **struct** node

{

**int** nData;

**struct** node\* pNext;

**struct** node\* pPrev;

}node;

node\* head = **NULL**;

node\* tail = **NULL**;

**void** insert(**int**, node\*\*, node\*\*); *// 노드 삽입함수. 삽입하면 자동 nData 기준 오름차순 정렬후 삽입한다.*

**void** showList(node\*, node\*); *// 리스트 전체 데이타 내용을 보여준다.*

**int** find(**int**, node\*, node\*, **int**\*); *// 특정한 nData값을 가진 노드를 찾는다.*

**void** deleteNode(**int**, node\*\*, node\*\*); *// 특정 노드 삭제.*

**int** main()

{

printf("이중 연결리스트 시작. \n\n");

**int** tailDepth = 0;

printf("find(%d). \n", 22);

printf("헤드부터 센 노드의 위치: %d \n", find(22, head, tail, &tailDepth));

printf("테일부터 센 노드의 위치: %d \n\n", tailDepth);

insert(11, &head, &tail);

insert(55, &head, &tail);

insert(33, &head, &tail);

insert(22, &head, &tail);

insert(66, &head, &tail);

insert(44, &head, &tail);

showList(head, tail);

printf("\n");

printf("find(%d). \n", 22);

printf("헤드부터 센 노드의 위치: %d \n", find(22, head, tail, &tailDepth));

printf("테일부터 센 노드의 위치: %d \n\n", tailDepth);

printf("find(%d). \n", 25);

printf("헤드부터 센 노드의 위치: %d \n", find(25, head, tail, &tailDepth));

printf("테일부터 센 노드의 위치: %d \n\n", tailDepth);

printf("find(%d). \n", 11);

printf("헤드부터 센 노드의 위치: %d \n", find(11, head, tail, &tailDepth));

printf("테일부터 센 노드의 위치: %d \n\n", tailDepth);

printf("deleteNode(%d). \n", 22);

deleteNode(22, &head, &tail);

showList(head, tail);

printf("\n");

printf("deleteNode(%d). \n", 66);

deleteNode(66, &head, &tail);

showList(head, tail);

printf("\n");

printf("deleteNode(%d). \n", 30);

deleteNode(30, &head, &tail);

showList(head, tail);

printf("\n");

printf("deleteNode(%d). \n", 11);

deleteNode(11, &head, &tail);

showList(head, tail);

printf("\n");

printf("insert(%d). \n", 77);

insert(77, &head, &tail);

showList(head, tail);

printf("\n");

printf("insert(%d). \n", 35);

insert(35, &head, &tail);

showList(head, tail);

printf("\n");

printf("끝. \n");

**return** 0;

}

**void** insert(**int** data, node\*\* headAddr, node\*\* tailAddr)

{

node\* newNode;

newNode = (node\*)malloc(**sizeof**(node));

newNode->nData = data;

newNode->pNext = **NULL**;

newNode->pPrev = **NULL**;

**if** (\*headAddr == **NULL** && \*tailAddr == **NULL**) *// 링크상의 노드가 하나도 없다면,*

{

\*headAddr = newNode;

\*tailAddr = newNode;

}

**else**

{

node\* pHead = \*headAddr;

node\* pTail = \*tailAddr;

**float** mid; *// 모든 노드들이 가진 nData의 평균값.*

*// mid 구하기.*

**int** nDataSum = 0;

**int** nodeCount = 0;

**while** ( pHead != **NULL** )

{

nDataSum += pHead->nData;

pHead = pHead->pNext;

nodeCount++;

}

mid = nDataSum / nodeCount;

*// printf("nDataSum: %d, nodeCount: %d, mid: %f \n", nDataSum, nodeCount, mid); // mid값 중간 확인.*

pHead = \*headAddr; *// 위에서 사용한 pHead 초기화.*

**if** (newNode->nData <= mid) *// 노드가 head에 가깝거나 정 가운데일때,*

{

**if** (!pHead->pNext || pHead->nData > newNode->nData) *// 리스트에 노드가 한개뿐일때 혹은 새 노드의 데이타가 최소값일때,*

{

*// 헤드를 새 노드로 교체.*

newNode->pNext = pHead;

pHead->pPrev = newNode;

\*headAddr = newNode;

}

**else**

{

**while** (pHead->pNext->nData < newNode->nData)

{

pHead = pHead->pNext;

}

newNode->pNext = pHead->pNext;

pHead->pNext->pPrev = newNode; *// 이 부분이 없으면 테일에서 출발한 링크에서 새 노드를 찾을 수 없음.*

newNode->pPrev = pHead;

pHead->pNext = newNode;

}

}

**else** **if**(newNode->nData > mid) *// 노드가 tail에 가까울때,*

{

**if** (!pTail->pPrev || pTail->nData < newNode->nData) *// 리스트에 노드가 한개뿐일때 혹은 새 노드의 데이터가 최대값일때,*

{

*// 테일을 새 노드로 교체.*

newNode->pPrev = pTail;

pTail->pNext = newNode;

\*tailAddr = newNode;

}

**else**

{

**while** (pTail->pPrev->nData > newNode->nData)

{

pTail = pTail->pPrev;

}

newNode->pPrev = pTail->pPrev;

pTail->pPrev->pNext = newNode; *// 이 부분이 없으면 헤드에서 출발한 링크에서 새 노드를 찾을 수 없음.*

newNode->pNext = pTail;

pTail->pPrev = newNode;

}

}

}

}

**void** showList(node\* head, node\* tail)

{

printf("헤드부터 오름차순으로 \n");

**while** (head)

{

printf("%d ", head->nData);

head = head->pNext;

}

printf("\n");

printf("테일부터 내림차순으로 \n");

**while** (tail)

{

printf("%d ", tail->nData);

tail = tail->pPrev;

}

printf("\n");

}

**int** find(**int** data, node\* head, node\* tail, **int**\* tailDepth)

{

**int** depth = 1;

**if** (!head && !tail) *// 리스트가 비어있다면,*

{

\*tailDepth = 0; *// 테일뎁스에도 같이 0을 입력.*

**return** 0; *// 0 반환하고 종료.*

}

**while** (head->nData != data)

{

**if** (head->pNext == **NULL**) *// head가 리스트의 끝까지 와도 일치하는 데이타가 없다면,*

{

depth = 0;

**break**;

*// 이중으로 확인해야 할 필요가 없다면 아래 코드 사용.*

*// \*tailDepth = 0; // 테일뎁스도 같이 0을 입력.*

*// return 0; // 0 반환하고 함수 종료.*

}

**else**

{

depth++;

head = head->pNext;

}

}

\*tailDepth = 1;

**while** (tail->nData != data)

{

**if** (tail->pPrev == **NULL**) *// tail이 리스트의 끝까지 와도 일치하는 데이타가 없다면,*

{

\*tailDepth = 0;

**break**;

}

**else**

{

\*tailDepth += 1;

*// \*tailDepth++; Expression result unused 경고. 위 코드랑 뭐가 달라서?*

tail = tail->pPrev;

}

*// 헤드뎁스에서 노드가 없을때 리턴 0 처리해서 끝낸다면 위의 if문이 필요 없으므로 아래 코드로 대체.*

*// \*tailDepth += 1;*

*// tail = tail->pPrev;*

}

**return** depth;

}

**void** deleteNode(**int** data, node\*\* headAddr, node\*\* tailAddr)

{

**int** headDepth = 0;

**int** tailDepth = 0;

node\* pHead = \*headAddr;

node\* pTail = \*tailAddr;

headDepth = find(data, pHead, pTail, &tailDepth);

**if** (headDepth == 1) *// 삭제 대상 노드가 헤드노드라면,*

{

pHead->pNext->pPrev = **NULL**;

\*headAddr = pHead->pNext;

free(pHead);

}

**else** **if** (tailDepth == 1) *// 삭제 대상 노드가 테일노드라면,*

{

pTail->pPrev->pNext = **NULL**;

\*tailAddr = pTail->pPrev;

free(pTail);

}

**else** **if** (headDepth == 0 && tailDepth == 0) *// 삭제 대상 노드가 리스트상에 존재하지 않는다면,*

{

printf("해당 데이타값을 가진 노드가 없어서 삭제가 불가능합니다. \n\n");

**return**;

}

**else**

{

**if** (headDepth <= tailDepth) *// 삭제 대상 노드가 헤드에 더 가깝거나 리스트의 가운데라면,*

{

**while** (headDepth > 0)

{

pHead = pHead->pNext;

headDepth--;

}

pHead->pPrev->pNext = pHead->pNext;

pHead->pNext->pPrev = pHead->pPrev;

pHead->pNext = **NULL**;

pHead->pPrev = **NULL**;

free(pHead);

}

**else** **if**(headDepth > tailDepth) *// 삭제 대상 노드가 테일에 더 가깝다면,*

{

**while** (tailDepth > 0)

{

pTail = pTail->pPrev;

tailDepth--;

}

pTail->pPrev->pNext = pTail->pNext;

pTail->pNext->pPrev = pTail->pPrev;

pTail->pPrev = **NULL**;

pTail->pNext = **NULL**;

free(pTail);

}

}

}