섹션 A - 링크된 목록

1. **(insertSortedLL)** 사용자에게 정수를 입력하도록 요청하고 이를 오름차순으로 연결된 목록에 삽입하는 C 함수 `insertSortedLL()`을 작성합니다. `insertSortedLL()` 함수는 현재 연결된 목록에 정수가 이미 있는 경우 정수를 삽입하지 않아야 합니다. 이 함수는 새 항목이 추가된 인덱스 위치를 반환해야 하며, 성공적으로 완료할 수 없는 경우 -1 값을 반환해야 합니다. 링크된 목록이 정렬된 링크된 목록이거나
   빈 목록입니다.

   ```
   ListNode *temp;
   temp = ll->head;
   ```

   ```
   curindex <= ll->index
   ```

   함수 프로토타입은 다음과 같습니다:

   ```
   int insertSortedLL(LinkedList *ll, int item);
   ```

   현재 연결된 목록이 **2, 3, 5, 7, 9인** 경우.
   값이 **8인** `insertSortedLL()`을 호출하면 다음과 같은 링크된 목록이 생성됩니다:
   2, 3, 5, 7, 8, 9.

   이 함수는 다음과 같이 새 항목이 추가된 인덱스 위치를 반환해야 합니다: 인덱스 4에 값 8이 추가되었습니다.

   현재 연결된 목록이 **5, 7, 9, 11, 15.**
   값이 **7인** `insertSortedLL()`을 호출하면 다음과 같은 링크된 목록이 생성됩니다:
   5, 7, 9 , 11, 15.

   함수가 성공적으로 완료되지 않았으므로(연결된 목록에 7이라는 값을 삽입하지 않았으므로) -1이라는 값을 반환해야 합니다:

   인덱스 -1에 값 7이 추가되었습니다.

   몇 가지 샘플 입력 및 출력은 다음과 같습니다:

   ```
   1: 정렬된 링크 목록에 정수를 삽입합니다:
   2: 가장 최근 입력 값의 인덱스를 인쇄합니다: 3: 정렬된 링크 목록을 인쇄합니다:
   0: 종료합니다:

   선택(1/2/3/0)을 입력하세요: 1
   링크된 목록에 추가할 정수를 입력합니다: 2 결과 링크된 목록: 2
   ```

선택(1/2/3/0)을 입력하세요: 1
링크된 목록에 추가할 정수를 입력합니다: 3 결과 링크된 목록: 2 3

선택(1/2/3/0)을 입력하세요: 1
링크된 목록에 추가할 정수를 입력합니다: 5

결과 링크 목록: 2 3 5

선택(1/2/3/0)을 입력하세요: 1
연결된 목록에 추가할 정수를 입력합니다: 7 결과 링크된 목록: 2 3 5 7

선택(1/2/3/0):1을 입력하세요.
연결된 목록에 추가할 정수를 입력합니다: 9 결과 링크된 목록: 2 3 5 7 9

선택(1/2/3/0)을 입력하세요: 1
연결된 목록에 추가할 정수를 입력합니다: 8 결과 링크된 목록: 2 3 5 7 8 9

선택(1/2/3/0)을 입력하세요: 2 인덱스 4에 값
8이 추가되었습니다.

선택(1/2/3/0):3을 입력하세요.
결과 정렬된 링크 목록은 2 3 5 7 8 9입니다.

선택(1/2/3/0)을 입력하세요: 1
연결된 목록에 추가할 정수를 입력합니다: 5 결과 링크된 목록: 2 3 5 7 8 9

원하는 값(1/2/3/0)을 입력하세요: 2 인덱스 -1
에 값 5가 추가되었습니다.

선택(1/2/3/0):3을 입력하세요.
결과 정렬된 링크 목록은 2 3 5 7 8 9입니다.

선택(1/2/3/0)을 입력하세요: 1
연결된 목록에 추가할 정수를 입력합니다: 11 결과 링크된 목록: 2 3 5 7 8 9
11

선택(1/2/3/0)을 입력하세요: 2 인덱스 6에 값
11이 추가되었습니다.

선택(1/2/3/0):3을 입력하세요.
결과 정렬된 링크 목록은 2 3 5 7 8 9 11입니다.

2. **(alternateMergeLL)** 두 번째 목록의 노드를 첫 번째 목록의 대체 위치에 삽입하는 C 함수 `alternateMergeLL()`을 작성합니다. 두 번째 목록의 노드는 첫 번째 목록에 대체 가능한 위치가 있을 때만 삽입해야 합니다.

함수 프로토타입은 다음과 같습니다:

```
void alternateMergeLL(LinkedList *ll1, LinkedList *ll2);
```

예를 들어 두 개의 링크된 목록이 LinkedList1과 LinkedList2라고 가정해 보겠습니다:
LinkedList1: **1, 2, 3**
LinkedList2: **4, 5, 6, 7**

결과 링크 목록은 다음과 같습니다:

LinkedList1: **1, 4, 2, 5, 3, 6**
LinkedList2: **7**

첫 번째 목록이 두 번째 목록보다 크면 두 번째 목록은 비어 있어야 합니다. 예를 들어 두 개의 링크된 목록이 LinkedList1과 LinkedList2라고 가정해 보겠습니다:

LinkedList1: **1, 5, 7, 3, 9, 11**
LinkedList2: **6, 10, 2, 4**

결과 링크된 목록은 다음과 같습니다:

LinkedList1: **1, 6, 5, 10, 7, 2, 3, 4, 9, 11**

LinkedList2: **비어 있음**

입력 및 출력 세션 샘플은 아래와 같습니다(현재 연결된 목록이 연결된 목록 1: **1, 2, 3** 및 연결된 목록 2: **4 , 5, 6, 7인** 경우):

```
연결된 목록 1: 1 2 3
연결된 목록 2: 4 5 6 7

선택(1/2/3/0)을 입력하세요: 3
주어진 링크된 목록을 병합한 후의 결과 링크된 목록은 다음과 같습니다: 연결된 목록 1: 1 4
2 5 3 6
연결된 목록 2: 7
```

3. **(moveOddItemsToBackLL)** 모든 홀수 정수를 연결된 목록의 뒤쪽으로 이동하는 C 함수 `moveOddItemsToBackLL()`을 작성합니다.

number count count temp count

함수 프로토타입은 다음과 같습니다:

```
void moveOddItemsToBackLL(LinkedList *ll);
```

몇 가지 샘플 입력 및 출력 세션이 아래에 나와 있습니다: 연

결된 목록이 **2, 3, 4, 7, 15, 18인** 경우:

```
홀수 정수를 링크된 목록의 뒤쪽으로 이동한 후 결과 링크된 목록은 2 4 18 3 7 15입니다.
```

연결된 목록이 **2, 7, 18, 3, 4, 15인** 경우:

```
홀수 정수를 링크된 목록의 뒤쪽으로 이동한 후 결과 링크된 목록은 2 18 4 7 3 15입니다.
```

현재 연결된 목록이 **1, 3, 5인** 경우:

```
홀수 정수를 링크된 목록의 뒤쪽으로 이동한 후의 결과 링크된 목록은 다음과 같습니다: 1 3
5
```

현재 링크된 목록이 2 4 6인 경우:
홀수 정수를 링크된 목록의 뒤쪽으로 이동한 후 결과 링크된 목록은 2 4 6입니다.

4. **(moveEvenItemsToBackLL)** 모든 짝수 정수를 연결된 목록의 뒤쪽으로 이동하는 C 함수 `moveEvenItemsToBackLL()`을 작성합니다.

함수 프로토타입은 다음과 같습니다:

```
void moveEvenItemsToBackLL(LinkedList *ll);
```

몇 가지 샘플 입력 및 출력 세션이 아래에 나와 있습니다: 연

결된 목록이 **2, 3, 4, 7, 15, 18인** 경우:

짝수 정수를 링크된 목록의 뒤쪽으로 이동한 후의 결과 링크된 목록은 다음과 같습니다: `3 7 15 2 4 18`

연결된 목록이 **2, 7, 18, 3, 4, 15인** 경우:

짝수 정수를 링크된 목록의 뒤쪽으로 이동한 후의 결과 링크된 목록은 다음과 같습니다: `7 3 15 2 18 4`

현재 연결된 목록이 **1, 3, 5인** 경우:

짝수 정수를 링크된 목록의 뒤쪽으로 이동한 후의 결과 링크된 목록은 다음과 같습니다: `1 3 5`

현재 링크된 목록이 `2 4 6`인 경우:

짝수 정수를 링크된 목록의 뒤쪽으로 이동한 후의 결과 링크된 목록은 `2 4 6`입니다.

5. **(frontBackSplitLL)** 단일 링크된 목록을 앞쪽 절반과 뒤쪽 절반의 두 개의 하위 목록으로 분할하는 C 함수 `frontBackSplitLL()`을 작성합니다. 요소 수가 홀수인 경우 여분의 요소는 앞쪽 목록에 들어가야 합니다. `frontBackSplitLL()`은 앞 리스트와 뒷 리스트를 포함하는 두 개의 리스트를 출력합니다.

함수 프로토타입은 다음과 같습니다:

```
void frontBackSplitLL(LinkedList *ll, LinkedList
*결과 프론트 리스트, 링크된 리스트 *결과 백 리스트);
```

fixsize = ll->size

예를 들어, 주어진 링크 목록이 **2, 3, 5, 6, 7이라고** 가정합니다.

결과 링크된 목록인 frontList와 backList는 다음과 같습니다:

frontList: **2, 3, 5**
backList: **6, 7**

입력 및 출력 세션의 샘플은 아래와 같습니다:

1: 연결된 목록에 정수를 삽입합니다:

2: 링크된 목록을 인쇄합니다:

3: 연결된 목록을 앞 목록과 뒷 목록의 두 개의 연결된 목록으로 분할합니다: 0: 종료합니다:

선택(1/2/3/0)을 입력하세요: 1

링크된 목록에 추가할 정수를 입력합니다: 2 결과 링크된 목록: 2

선택(1/2/3/0)을 입력하세요: 1

링크된 목록에 추가할 정수를 입력합니다: 3 결과 링크된 목록: 2 3

선택(1/2/3/0)을 입력하세요: 1

링크된 목록에 추가할 정수를 입력합니다: 5 결과 링크된 목록: 2 3 5

선택(1/2/3/0)을 입력하세요: 1

연결된 목록에 추가할 정수를 입력합니다: 6 결과 링크된 목록: 2 3 5 6

선택(1/2/3/0)을 입력하세요: 1

연결된 목록에 추가할 정수를 입력합니다: 7 결과 링크된 목록: 2 3 5 6 7

선택(1/2/3/0)을 입력하세요: 2 결과 링크 목록:
2 3 5 6 7

선택(1/2/3/0)을 입력하세요: 3

주어진 링크 목록을 분할한 후 결과 링크 목록은 다음과 같습니다: 앞쪽 링크 목록: 2 3 5

뒤로 링크 목록: 6 7

6. **(moveMaxToFront)** 연결된 정수 목록을 최대 한 번만 순회한 다음 저장된 값이 가장 큰 노드를 앞쪽으로 이동하는 C 함수 `moveMaxToFront()`를 작성합니다.

목록 [illegible]term[illegible]else [illegible]listnode[illegible]
front->next == max->next; max->next = *ptrHead; *ptrHead = max;

함수 프로토타입은 다음과 같습니다:

```
int moveMaxToFront(ListNode **ptrHead);
```

[illegible]1.[illegible]Insertnode[illegible] 2.ListNode *temp[illegible]temp = *ptrHead[illegible]

예를 들어 링크된 목록이 **(30, 20, 40, 70, 50)**인 경우 결과 링크된 목록은 **(70, 30, 20, 40)**이 됩니다, **50)**.

1: 연결된 목록에 정수를 삽입합니다:

2: 저장된 값이 가장 큰 노드를 목록의 앞쪽으로 이동합니다:

0: 종료합니다:

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 30 연결된 목록은 30

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 20 연결된 목록은 다음과 같습니다:
30 20

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 40 연결된 목록은 다음과 같습니다:

30 20 40

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 70 연결된 목록은 다음과 같습니다:
30 20 40 70

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 50 연결된 목록은 다음과 같습니다:
30 20 40 70 50

선택 사항을 입력하세요(1/2/0): 2

결과 링크된 목록은 다음과 같습니다: 70 30 20 40 50 원

하는 것을 입력하세요(1/2/0): 0

7. **(recursiveReverse)** 다음 포인터와 그 헤드 포인터를 변경하여 주어진 링크된 목록을 재귀적으로 역순으로 되돌리는 C 함수 `recursiveReverse()`를 작성합니다.

함수 프로토타입은 다음과 같습니다:

```
void recursiveReverse(ListNode **ptrHead);
```

예를 들어 연결된 목록이 **(1, 2, 3, 4, 5)**인 경우 결과 연결된 목록은 **(5, 4, 3, 2, 1)**이 됩니다.

입력 및 출력 세션의 샘플은 다음과 같습니다: 1: 연결된 목록에 정수를 삽입합니다: 2: 역연결 리스트:
0: 종료합니다:

□, ʈʂ목록노드⌐*첫⌐*휴식⌐
ऱNULLI௨return5 재귀
적ς&rest≭; first-next-
next = first;
첫 번째 다음 = NULL;
*ptrHead = rest;

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 1 결과 링크된 목록은 다음과 같습니다
: 1

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 2 결과 링크된 목록은 다음과 같습니다
: 1 2

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 3 결과 링크된 목록은 다음과 같습니다
: 1 2 3

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 4 결과 링크된 목록은 다음과 같습니다
: 1 2 3 4

선택 사항을 입력하세요(1/2/0): 1

링크된 목록에 추가할 정수를 입력합니다: 5 결과 링크된 목록은 다음과 같습니다
: 1 2 3 4 5

선택 사항을 입력하세요(1/2/0): 2

요소를 반전시킨 결과 링크된 목록은 다음과 같습니다: 5 4 3 2 1 원하는 것을 입력하세요

(1/2/0): 0