섹션 C - 스택 및 대기열

1. **(createQueueFromLinkedList)** 링크된 목록에 저장된 모든 정수를 큐에 대기시켜 큐(링크된 목록 기반)를 생성하는 C 함수 `createQueueFromLinkedList()`를 작성합니다. 연결된 목록의 첫 번째 노드를 먼저 큐에 넣은 다음 두 번째 노드를 큐에 넣는 식으로 진행됩니다. 큐가 비어 있지 않은 경우 처음에 큐를 비워야 한다는 점을 잊지 마세요.

   함수 프로토타입은 다음과 같습니다:

   ```
   void createQueueFromLinkedList(LinkedList *ll , Queue *q);
   ```

   현재 연결된 목록이 **1, 2, 3, 4, 5인** 경우 입력 및 출력 세션의 샘플이 아래에 나와 있습니다(현재 연결된 목록이 **1, 2, 3, 4, 5인** 경우):

   결과 링크 목록은 다음과 같습니다: 1 2 3 4 5 선택 항목 (1/2/3/0)을 입력하세요: 2 결과 대기열: 1 2 3 4 5

   ```
   ListNode *temp= ll->head; while(temp != NULL) {
       //원하는 대로 하세요// 임시= 임시->다음;
   }
   ```

2. **(createStackFromLinkedList) 링크된** 리스트에 저장되어 있는 모든 정수를 푸시하여 스택(링크된 리스트 기반)을 생성하는 C 함수 `createStackFromLinkedList()`를 작성합니다. 링크된 목록의 첫 번째 노드를 먼저 푸시하고, 그 다음 두 번째 노드를 푸시하는 식으로 푸시합니다. 스택이 비어 있지 않은 경우 처음에 스택을 비워야 한다는 점을 잊지 마세요.

   함수 프로토타입은 다음과 같습니다:

   ```
   void createStackFromLinkedList(LinkedList *ll , Stack *stack);
   ```

   현재 연결된 목록이 **1, 3, 5, 6, 7인** 경우 입력 및 출력 세션의 샘플이 아래에 나와 :

   결과 링크 목록은 다음과 같습니다: 1 3 5 6 7 선택(1/2/3/0)을 입력하세요: 2 결과 스택: 7 6 5 3 1

   ```
   ListNode *temp= ll->head; while(temp != NULL) {
       //원하는 대로 하세요// 임시= 임시->다음;
   }
   ```

3. **(isStackPairwiseConsecutive)** C 함수 작성 스택의 숫자가 쌍으로 연속되는지 여부를 검사하는 함수 `isStackPairwiseConsecutive()`를 작성합니다. `isStackPairwiseConsecutive()` 함수는 스택에서 정수를 추가하거나 제거할 때만 `push()` 및 `pop()` 함수를 사용한다는 점에 유의하세요.

   함수 프로토타입은 다음과 같습니다:

   ```
   int isStackPairwiseConsecutive(Stack *s);
   ```

   샘플 테스트 사례는 아래에 나와 있습니다:

   예를 들어 스택이 **(16, 15, 11, 10, 5)**인 경우

   예를 들어 스택이 **(16, 15, 11, 10, 5, 4)**인 경우입니다:
   다음과 같습니다: 16 15 11 10 5 4 스택은 쌍으로 연속됩니다.

   예를 들어 스택이 **(16, 15, 11, 10, 5, 1)** 인 경우
   스택은 다음과 같습니다: 16 15 11 10 5 1
   스택이 쌍으로 연속되지 않습니다.

크기가 홀수이면 0을 반환합니다.
그렇지 않으면 모든 쌍을 검사하고, 만족하지 않으면 0을 반환하고 모든
검사를 통과하면 1을 반환할 수 있습니다.

스택은 다음과 같습니다: 16 15 11 10 5
스택이 쌍으로 연속되지 않습니다.

4. **(reverseQueue) 스택을** 사용하여 큐를 되돌리기 위한 C 함수 `reverseQueue()`를 작성합니다. `reverseQueue()` 함수는 스택에서 정수를 추가하거나 제거할 때는 `push()` 및 `pop()`만 사용하고 큐에서 정수를 추가하거나 제거할 때는 `enqueue()` 및 `dequeue()`만 사용한다는 점에 유의하세요. 스택이 비어 있지 않은 경우 처음에 스택을 비워야 한다는 점을 잊지 마세요.

스택을 초기화하는 것을 잊지 마세요.

함수 프로토타입은 다음과 같습니다:

```
void reverseQueue(큐 *q);
```

예를 들어 대기열이 **(1, 2, 3, 4, 5)**인 경우 결과 대기열은 **(5, 4, 3, 2, 1)**이 됩니다.

5. **(recursiveReverseQueue) 재귀적** C 함수 `recursiveReverseQueue()`를 작성합니다. 를 사용하면 정수 큐에 저장된 항목의 순서를 반대로 바꿀 수 있습니다.

int temp;
if(q->ll.head== NULL) return; temp =
dequeue(q); recursiveReverseQueue(q);
enqueue(q,temp);

함수 프로토타입은 다음과 같습니다:

```
void recursiveReverseQueue(큐 *q);
```

예를 들어 대기열이 **(1, 2, 3, 4, 5)**인 경우 결과 대기열은 **(5, 4, 3, 2, 1)**이 됩니다.

6. **(removeUntilStack)** 선택한 값이 처음 나타날 때까지 정수 스택에서 모든 값을 꺼내는 C 함수 `removeUntilStack()`을 작성합니다.

동안 루프는 다음과 같이 끊어질 수 있습니다 s->ll.size == 0을 고려해야 함을 기억하세요.

함수 프로토타입은 다음과 같습니다:

```
void removeUntilStack(Stack *s, int value);
```

왼쪽에 맨 위 숫자가 표시된 스택**(1, 2, 3, 4, 5, 6, 7)**이 주어지면 다음을 호출합니다.
=4 값을 가진 `removeUntilStack()`은 스택**(4, 5, 6, 7)**을 생성합니다.

왼쪽에 맨 위 숫자가 표시된 스택**(10, 20, 15, 25, 5)**이 주어지면 다음을 호출합니다.
=15 값을 가진 `removeUntilStack()`은 스택**(15, 25, 5)**을 생성합니다.

7. **(balanced)** 문자 `()[]{}`로 구성된 표현식이 균형이 확인하는 C 함수 `balanced()` 작성합니다. `balanced()` 함수의 프로토타입은 아래와 같습니다:

함수 프로토타입은 다음과 같습니다:

```
int balanced(char *expression);
```

예를 들어 다음 표현식은 괄호 안의 순서와 수량이 일치하므로 균형이 맞습니다:

```
()
([])
{[]()[]}
```

스택 사용
if {[(스택에 밀어 넣기 else {
if }]) 스택의 엿보기와 일치하면 팝
}
마지막으로 스택이 비어 있는지 확인합니다.

입력 및 출력 세션의 샘플은 아래와 같습니다:
1: 문자열을 입력합니다:

2: 문자 ()[]{}로 구성된 표현식이 균형이 맞는지 확인합니다:
0: 종료합니다:

선택 사항을 입력하세요(1/0): 1
공백 없이 표현식을 입력하여 균형이 맞는지 확인합니다: {[]()[]}
{[]()[]}
균형 잡힌!

선택 사항을 입력하세요(1/0): 0

다음 표현식은 균형이 맞지 않습니다:
{()]
[({()])

아래에 입력 및 출력 세션 샘플이 나와 있습니다:
1: 문자열을 입력합니다:
2: 문자 ()[]{}로 구성된 표현식이 균형이 맞는지 확인합니다:
0: 종료합니다:

선택 사항을 입력하세요(1/0): 1
공백 없이 입력하여 균형 여부를 확인합니다: [({()])
[({()])
균형이 맞지 않습니다!

선택 사항을 입력하세요(1/0): 0