Album

After helping Rar the Cat code a website in Lab #02, Rar the Cat's website has achieved great success. Now, he wants you to add a new functionality to the website – the ability to create and modify albums.

Each album is a collection of photos where the order is very important. A user will be allowed to insert or delete photos from specific places in the album to create a video montage of photos. In our case, an album is allowed to contain repeated photos.

Rar the Cat wants you to create an album editor that supports the following operations to an album:

1. Insert a photo to the album at a specific position. (**INSERT**)
2. Delete a photo in the album from a specific position. (**DELETE**)
3. Generates a preview of the album by printing the sequence of photos. (**PREVIEW**)
4. Undo the last *change* to the album. **(UNDO)**

To proof that your code can work for multiple users in parallel, Rar the Cat wants you to support editing operations for **two albums**. The two albums will be identified by their **albumID**, which are **"A"** and **"B"** respectively. Similar to Lab #02, photos will be uniquely identified by their **photoID**.

In addition, Rar the Cat also wants you to support the following query:

5. Count the number of **distinct photos** if both albums "A" and "B" were **combined**. **(COUNT)**
   Two photos are considered distinct *if and only if* they have a different **photoID**.

After looking at the scores for Lab #02, Panda boasts that the above is too easy for CS1020 students and they should be able to do it with just ***LinkedList***. Rar the Cat hence challenges you to code the album editor with only ***LinkedList***.

As a reminder, you are **not allowed to use any other classes and methods from the Collections class** *except* the ***LinkedList*** and ***ListIterator*** **class** and methods from these two classes. For more details, please refer to the last page or ask any Lab TA if you are unsure.

In addition, we have also provided an implementation of ***ExtendedLinkedList****.java* that was covered in Lecture 5B. This can be found as classes ***ExtendedLinkedList*** and ***ListNode*** within your skeleton file. You are free to use *and/or modify* this implementation for this task. However, you might want to duplicate a copy of the provided implementation in case you want to revert your changes.

Input

The first line of input will contain a single integer **N**, the number of operations. It is guaranteed that $0 < N \leq 500$.

N lines of input will follow. Each line of input will correspond to one operation.
The details and input format of these operations will be detailed in subsequent pages.

Among these, it is guaranteed that all **albumID** parameters will be either "A" or "B".
Also, all **position** parameters will be an integer between 0 and 1,000,000,000 (1 billion) as well.
In addition, all **photoID** parameters will be an integer between 0 and 1,000,000,000 (1 billion).

Output

For each operation, print the corresponding output denoted in subsequent pages.

1. Insert photo to album operation

Description

This operation will be provided with 3 parameters: **albumID** followed by **position** then **photoID**. This indicates that photo with **photoID** is to be added *after* the **position**-th photo from the *start* of the album identified by **albumID**. If **position** is 0, this means that the photo should be added at the start of the album. Do note that the same photo can be added to the same album multiple times.

Response and Error Handling

- If **position** is greater than the number of photos in the album (**number_of_photos**), ignore the operation and output "Invalid position, album **[albumID]** only has **[number_of_photos]** photos."
- Otherwise, output "Photo **[photoID]** inserted after position **[position]** of album **[albumID]**."

Input/Output Format

In your input, this operation will be in the following format:

`INSERT [albumID] [position] [photoID]`

Do note that output corresponding to alternate lines of input are **bolded** in the table below for clarity.

Sample Input (**album1.in**)	Sample Output (**album1.out**)
8	*[Padded for clarity, do not output this]*
INSERT A 0 1	**Photo 1 inserted after position 0 of album A.**
INSERT B 0 2	Photo 2 inserted after position 0 of album B.
INSERT A 1 3	**Photo 3 inserted after position 1 of album A.**
INSERT A 0 1	Photo 1 inserted after position 0 of album A.
INSERT A 3 2	**Photo 2 inserted after position 3 of album A.**
INSERT B 1000000000 0	Invalid position, album B only has 1 photos.
INSERT A 123456789 17	**Invalid position, album A only has 4 photos.**
INSERT B 1 9	Photo 9 inserted after position 1 of album B.

2. Delete photo from album operation

Description

This operation will be provided with 2 parameters: **albumID** followed by **position**, indicating that the **position**-th photo from the start of album identified by **albumID** should be deleted from the album.
To clarify, this **does not mean** deleting all photos from the album with the same **photoID** as the **position**-th photo.

Response and Error Handling

- If **position** is greater than the number of photos in the album (**number_of_photos**), ignore the operation and output "Invalid position, album **[albumID]** only has **[number_of_photos]** photos."
- If **position** is **0**, output "Invalid position 0."
- Otherwise, output "Photo deleted from position **[position]** of album **[albumID]**."

Input/Output Format

In your input, this operation will be in the following format: `DELETE [albumID] [position]`

Do note that output corresponding to alternate lines of input are **bolded** in the table below for clarity.

Sample Input (**album2.in**)	Sample Output (**album2.out**)
8	*[Padded for clarity, do not output this]*
INSERT A 0 10	**Photo 10 inserted after position 0 of album A.**
INSERT A 1 11	Photo 11 inserted after position 1 of album A.
INSERT A 2 12	**Photo 12 inserted after position 2 of album A.**
DELETE A 2	Photo deleted from position 2 of album A.
DELETE B 2	**Invalid position, album B only has 0 photos.**
INSERT A 3 13	Invalid position, album A only has 2 photos.
DELETE A 2	**Photo deleted from position 2 of album A.**
DELETE A 0	Invalid position 0.

3. Preview album operation

Description

This operation will be provided with a single parameter: albumID only. In this operation, you are supposed to print the *photoID* of all the photos in the album identified by albumID, in the format described below.

Response and Error Handling

- Firstly, print "Album [albumID]: ", followed by the sequence of *photoIDs* in the album.
- The sequence of photoIDs should be contained within a pair of square brackets "[…]" and printed in the order that they appear in the album.
- Between each photoID, they should be separated with a comma followed by a space. Eg: "2, 3"
- End the line with a full stop.

Input/Output Format

In your input, this operation will be in the following format: `PREVIEW [albumID]`
Do note that output corresponding to alternate lines of input are bolded in the table below for clarity.

Sample Input (album3.in)	Sample Output (album3.out)
8 **INSERT A 0 10** PREVIEW A **PREVIEW B** INSERT A 1 12 **INSERT A 2 11** INSERT A 2 10 **INSERT A 3 13** PREVIEW A	*[Padded for clarity, do not output this]* **Photo 10 inserted after position 0 of album A.** Album A: [10]. **Album B: [].** Photo 12 inserted after position 1 of album A. **Photo 11 inserted after position 2 of album A.** Photo 10 inserted after position 2 of album A. **Photo 13 inserted after position 3 of album A.** Album A: [10, 12, 10, 13, 11].

4. Undo operation

Description

This operation will be provided with a single parameter: albumID and you are supposed to *undo the last change to the album* identified by albumID. A change to the album is defined as either an insert operation or a delete operation. An undo operation is ***not*** considered as a change to the album. In addition, if an insert or delete operation was ***ignored***, it is also not considered as a change to the album.

- In the case of an *insert operation*, the corresponding undo action is to *remove the inserted photo*.
- In the case of a *delete operation*, the corresponding undo action is to *restore (insert back) the deleted photo (in the same position)*.

For example, if photo 1 was inserted, followed by photo 2 and photo 3 into album A resulting in the album having the following: [1, 3, 2]. Subsequently, photo 1 is deleted, resulting in [3, 2].
Five undo operations are then given, which each undo operation should do is listed below:

- The first undo operation should undo the deletion of photo 1 from album A. This means photo 1 should be added back to its original position, resulting in [1, 3, 2].
- The second undo operation should undo the insertion of photo 3 to album A. This means that the album is restored to the state before the insertion of photo 3, which is [1, 2].
- The third undo operation will remove photo 2 to undo its insertion, resulting in [1].
- The fourth undo operation will remove photo 1, resulting in an empty album.
- Lastly, the last undo operation should output "No changes to album A to undo." as all the changes to album A has already been undone.

Do note that *undo* and *changes* are *confined to a single album* meaning that *undo* operation should only undo prior insert or delete operations on the same album.

Response and Error Handling

- If there are no changes to the album, or all changes have already been undone, output "No changes in album [albumID] to undo."

- Otherwise, perform the corresponding changes and then output "Album [albumID] has been undone."

Input/Output Format

In your input, this operation will be a single word on a single line: UNDO [albumID]
Do note that output corresponding to alternate lines of input are bolded in the table below for clarity.

Sample Input (album4.in)	Sample Output (album4.out)
17	*[Padded for clarity, do not output this]*
INSERT A 0 1	**Photo 1 inserted after position 0 of album A.**
INSERT A 0 2	Photo 2 inserted after position 0 of album A.
INSERT A 1 3	**Photo 3 inserted after position 1 of album A.**
DELETE A 1	Photo deleted from position 1 of album A.
PREVIEW A	**Album A: [3, 1].**
UNDO A	Album A has been undone.
PREVIEW A	**Album A: [2, 3, 1].**
UNDO B	No changes in album B to undo.
INSERT B 0 0	**Photo 0 inserted after position 0 of album B.**
UNDO A	Album A has been undone.
UNDO A	**Album A has been undone.**
PREVIEW A	Album A: [1].
INSERT A 0 10	**Photo 10 inserted after position 0 of album A.**
UNDO A	Album A has been undone.
UNDO A	**Album A has been undone.**
PREVIEW A	Album A: [].
UNDO A	**No changes in album A to undo.**

Errata in red.

5. Count distinct photos query operation

Description

This operation will be provided with no parameters. You are to count and output the number of distinct photos in both the albums, combined.
For example, if album A contains photos {3, 3, 7} and album B contains photos {6, 7, 7, 8}, this counts only as 4 distinct photos – {3, 6, 7, 8}.

Response and Error Handling

- Compute how many *distinct photos* there are in both the albums combined as [distinct_photos].
- Output the result in a single line as such: "Number of distinct photos: [distinct_photos]."

Input/Output Format

In your input, this operation will be a single word on a single line: COUNT
Do note that output corresponding to alternate lines of input are bolded in the table below for clarity.

Sample Input (album5.in)	Sample Output (album5.out)
13	*[Padded for clarity, do not output this]*
COUNT	**Number of distinct photos: 0.**
INSERT A 0 3	Photo 3 inserted after position 0 of album A.
INSERT A 1 7	**Photo 7 inserted after position 1 of album A.**
INSERT B 0 6	Photo 6 inserted after position 0 of album B.
INSERT B 1 8	**Photo 8 inserted after position 1 of album B.**
COUNT	Number of distinct photos: 4.
INSERT A 2 3	**Photo 3 inserted after position 2 of album A.**
COUNT	Number of distinct photos: 4.
DELETE A 3	**Photo deleted from position 3 of album A.**
COUNT	Number of distinct photos: 4.
INSERT B 1 7	**Photo 7 inserted after position 1 of album B.**
INSERT B 2 7	Photo 7 inserted after position 2 of album B.
COUNT	**Number of distinct photos: 4.**

Sample Testcase

We have also provided a longer sample testcase that includes all operations and *most* of their error handling cases, with explanation. Do note that output corresponding to alternate lines of input are **bolded** in the table below for clarity. (You do not need to bold your output.)
You are reminded that you can find *all* the input/output in this statement in your working directory.

Sample Input (**album6.in**)	Sample Output (**album6.out**)
20	*[Padded for clarity, do not output this]*
INSERT A 0 17	**Photo 17 inserted after position 0 of album A.**
INSERT A 1 13	Photo 13 inserted after position 1 of album A.
INSERT B 0 6	**Photo 6 inserted after position 0 of album B.**
INSERT B 1 8	Photo 8 inserted after position 1 of album B.
INSERT B 2 13	**Photo 13 inserted after position 2 of album B.**
COUNT	Number of distinct photos: 4.
UNDO A	**Album A has been undone.**
COUNT	Number of distinct photos: 4.
UNDO B	**Album B has been undone.**
DELETE A 1	Photo deleted from position 1 of album A.
PREVIEW A	**Album A: [].**
INSERT B 3 13	Invalid position, album B only has 2 photos.
PREVIEW B	**Album B: [6, 8].**
UNDO A	Album A has been undone.
UNDO B	**Album B has been undone.**
PREVIEW B	Album B: [6].
INSERT A 1 17	**Photo 17 inserted after position 1 of album A.**
INSERT A 1 5	Photo 5 inserted after position 1 of album A.
PREVIEW A	**Album A: [17, 5, 17].**
COUNT	Number of distinct photos: 3.

Explanation

There are a total of 20 operations in this input.

#	Description	Album A	Album B
1	*Insert* photo **17** after the **0**-th photo of album **A**.	[17]	[]
2	*Insert* photo **13** after the **1**-th photo of album **A**.	[17, 13]	[]
3	*Insert* photo **6** after the **0**-th photo of album **B**.	[17, 13]	[6]
4	*Insert* photo **8** after the **1**-th photo of album **B**.	[17, 13]	[6, 8]
5	*Insert* photo **13** after the **2**-th photo of album **B**.	[17, 13]	[6, 8, 13]
6	*Count* number of distinct photos.	*Result*: **4** – {6, 8, 13, 17}	
7	*Undo* the last change to album **A**, which is #2.	[17]	[6, 8, 13]
8	*Count* number of distinct photos.	*Result*: **4** – {6, 8, 13, 17}	
9	*Undo* the last change to album **B**, which is #5.	[17]	[6, 8]
10	*Delete* the **1**-th photo of album **A**.	[]	[6, 8]
11	Print a *preview* of album **A**.	**[]**	[6, 8]
12	*Insert* photo **13** after the **3**-th photo of album **B**. (Invalid position since album **B** only has 2 photos).	[]	[6, 8]
13	Print a *preview* of album **B**.	[]	**[6, 8]**
14	*Undo* the last change to album **A**, which is #10.	[17]	[6, 8]
15	*Undo* the last change to album **B**, which is #4.	[17]	[6]
16	Print a *preview* of album **B**.	[17]	**[6]**
17	*Insert* photo **17** after the **1**-th photo of album **A**.	[17, 17]	[6]
18	*Insert* photo **5** after the **1**-th photo of album **A**.	[17, 5, 17]	[6]
19	Print a *preview* of album **A**.	**[17, 5, 17]**	[6]
20	*Count* number of distinct photos.	*Result*: **3** – {5, 6, 17}	

Skeleton

You are given the skeleton file **`Album.java`**. You should see a non-empty file when you open the skeleton file. Otherwise, you might be in the wrong working directory.

Notes:

1. You should develop your program in the subdirectory **`ex1`** and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. If your algorithm is different from the given skeleton, you are free to write a solution according to your own algorithm. However, **your algorithm must use linked list** to solve this problem.
   You are not allowed to use arrays, ArrayList, HashMap, etc. for this problem. You are also **not allowed to use any methods from the Collections class (such as Collections.sort)**.
   Solution that does not use linked list or uses any other data structures and methods from the Collections class will receive 0 marks.
   However, you are allowed to use Java API **LinkedList**, **ListIterator** *and/or* the provided *ExtendedLinkedList* and *ListNode* implementations.
   You are allowed to use a mix of Java API's *LinkedList* and the *ExtendedLinkedList* in your solution.
3. You are **free to define your own classes (or remove existing ones)** if it is suitable.
4. Please be reminded that the marking scheme is:

   Input : 10%
   Output : 10%
   Correctness : 50%
   Programming Style : 30% (awarded if you score **at least 20% from the above**):
   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity (modular programming, proper modifiers [public / private]): 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

   Compilation Error : Deduction of **50% of the total marks obtained**.

Skeleton File – Album.java

You should see the following contents when you open the skeleton file:

```
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.Scanner;
import java.util.NoSuchElementException;


public class Album {

    public Album() {
        //constructor
    }

    private void run() {
        //implement your "main" method here
    }

    public static void main(String[] args) {
        Album newAlbum = new Album();
        newAlbum.run();
    }
}

public class ListNode<E> {  …  }

public class ExtendedLinkedList<E> {  …  }
```