Lab Practice Problem 6

Filename: `practice6_surname.c`

In this activity, you will implement a program that compresses a list of positive integers in-place by removing any consecutive duplicate values. After compression, only the first occurrence of each sequence of identical numbers should remain in the array.

Try It Out

Write a program that removes all **consecutive duplicate values** from an array, leaving only one instance of each repeated number. The following function prototypes are provided that define the **interfaces** you need to implement:

`void print_array(int arr[], int size);`
: Print all elements of `arr` consecutively (no spaces) on one line, then output a newline.

`int compress_array(int arr[], int size);`
: Removes consecutive duplicate elements from the array `arr` in place. It returns the new logical size of the array after duplicates have been removed.

`int shift_array(int arr[], int size, int index);`
: Shifts all elements of the array `arr` one position to the left, starting from the specified `index`. This operation effectively overwrites the element at `index` with the element immediately to its right, and so on, until the second-to-last element. The function returns the updated logical size of the array, which is one less than the original. One element is effectively "*removed*" from `arr`.

In the `main()` function, use a loop to process all test cases and display the results.

Input

The first line of input contains an integer $0 \leq \mathsf{T} \leq 100$, the number of test cases. The following T lines each contain a sequence of up to 10 positive integers, followed by a sentinel value `0`. The value `0` marks the end of the list for that test case and is not included as part of the list of numbers or in the count of the 10 possible integers.

Output

For each test case, the program should print one line of output. The line begins with the test case number, followed by a colon and a space. After this, print the contents of the array after it has been compressed. Show only the values that remain in the array after removing all consecutive duplicates. Each remaining number should be separated by a single space.

Sample Input

```
5
1 2 3 4 5 0
1 1 2 2 3 4 5 5 4 0
1 1 1 1 1 1 1 1 1 1 0
1 11 111 1111 111 11 1 0
0
```

Sample Output

```
1: 1 2 3 4 5
2: 1 2 3 4 5 4
3: 1
4: 1 11 111 1111 111 11 1
5:
```

Guide Questions

1. In this program, we keep track of the array's size (i.e., logical size) separately from its declared capacity. Why is it important to distinguish between the array's logical size and its fixed capacity? How does this distinction affect how the program processes and prints the array after compression?
2. What changes would you need to make in your program if the array's capacity increased from 10 to 100 elements? Would this affect only the macro definition, or would other parts of your code or design need adjustment as well?
3. Why do you think the functions `compress_array()` and `shift_array()` were designed to return an `int` value? What might happen if these functions were instead declared as `void`? Would it still be possible to solve the problem and produce the same output? If yes, what would the function prototype look like? **Hint:** Pointers!
4. We worked with numbers stored in an array of integers. Now, imagine that instead of numbers, we are dealing with letters. In this new context, the data is stored in an array of characters. Assuming that a correct `main()` function is already provided, how would you update the other functions to handle this change? Specifically, what modifications would you make to their prototypes and their definitions?

Sample Solution

The source code can be accessed here.