Lab Practice Problem 4

Filename: `practice4_surname.c`

In this activity, you will practice reading multiple test cases, using loops, and applying functions to solve a problem in C. The focus is on **modular design**, where one function is responsible for reversing a number and another checks if the number is a palindrome.

Try It Out

Write a program that checks if a number is a palindrome. A palindrome is one that reads the same forwards and backwards, such as 121, 10101, or 7.

To accomplish this activity, you will build upon solutions from previous labs and extend them.

- Define a function `get_reverse(num)` that computes and returns the reverse of a number.
- Define a function `is_palindrome(num)` that calls `get_reverse()` and checks if the number is equal to its reverse. It returns a 1 if it is, 0 otherwise.
- In the `main()` function, use a loop to process all test cases and display the results.

A good strategy is to first set up the overall structure of the program in the order described above, without focusing on the detailed logic right away. Begin by writing the `main()` function, as in earlier activities. Include the necessary function prototypes at the top. Then, add the function call to `is_palindrome()`, which depends on `get_reverse()`. Once this structure is in place, you can complete the detailed logic inside each function.

At this stage, it is fine to leave placeholders in the functions so that the program compiles and the structure is ready. You can then fill in the detailed logic later. This incremental approach allows you to focus on one step at a time and gradually build the full program.

Assumption: You may assume that all inputs are positive integers without leading zeros. You do not need to handle negative numbers or extremely large values that might cause overflow.

Input

The first line of input contains an integer $0 \leq \mathrm{T} \leq 100$, the number of test cases. The following T lines each contain a single integer $0 \leq \mathrm{N} < 1{,}000{,}000$ to be evaluated.

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, three values must be printed, each separated by a single space: the original number, the reversed number, and finally a 1 if the number is a palindrome or a 0 otherwise.

Sample Input

```
6
121
10101
7
1234
12
100
```

Sample Output

```
1: 121 121 1
2: 10101 10101 1
3: 7 7 1
4: 1234 4321 0
5: 12 21 0
6: 100 1 0
```

Guide Questions

1. Why is it beneficial to split the program into two separate functions, `get_reverse()` and `is_palindrome()`, instead of writing all the code inside the `main()` function?
2. Suppose one of the test cases is the number `0`. What would the program output for this test case? Explain why this result makes sense based on how the loop in `get_reverse()` works.
3. Suppose one of the test cases is the number `1000`. The reverse of this number would be `0001`, which is stored simply as `1`. What output should the program produce in this case, and why is this the expected behavior?
4. Suppose one of the test cases is a very large number close to the maximum value an `int` can hold in C (for example, `2,147,483,647`). What potential problems might occur when reversing this number, and why? What data type could be used instead to help avoid overflow?
5. The `for` loop in `main()` uses the variable `i` as its loop control variable. Can the variable `i` be accessed outside the loop after it finishes executing? Explain why or why not, based on block scope rules in C.

Sample Solution

The solution will be released after the submission window has closed. You may review it on the activity page at that time.