Lab Practice Problem 3

Filename: `practice3_surname.c`

In this activity, you will practice reading input with loops and applying character operations in C. Although your program does not directly open or read files, you will simulate this process using input redirection. With redirection, the contents of a file are treated as if they were typed by the user at the keyboard. This new format will also be used for grading, since it allows a prepared set of test cases to be fed automatically into your program.

Try It Out

Write a program that accepts a single character as input and determines its position in the English alphabet (e.g., 'A' or 'a' = 1, 'B' or 'b' = 2, ..., 'Z' or 'z' = 26). If the input is not a valid letter, your program should output `"Invalid"`.

Important Do not write 26 separate `if` statements. Do not use a `switch` statement. Instead, rely on character arithmetic and existing library functions.

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 character.

Output

For each test case, output the test case number followed by a colon and a single space. Then, print the position of the letter if valid, or `"Invalid"` otherwise.

Sample Input

```
5
a
Y
@
V
i
```

Sample Output

```
1: 1
2: 25
3: Invalid
4: 22
5: 9
```

Running Your Program on the Eustis Server

The following instructions are specific to the Eustis server. After writing and saving your C file, compile it as usual:

```
gcc practice3_surname.c -o prac3
```

Assuming the compiled executable is named `prac3`, there are two common ways to provide input for testing:

Option 1: Typing Input Directly (EOF Approach)

You can simulate file input without creating a separate file by typing the test cases directly:

```
./prac3 <<EOF
contents go here
EOF
```

Replace `contents go here` with your test cases. You may use newlines to separate multiple inputs. Press `Enter` after the last line. The program will run until it encounters `EOF`.

Option 2: Redirecting Input from a File (Preferred)

The file you create here should contain exactly what is shown in the **Sample Input** section of the instructions. Save your test cases in a text file (e.g., `input.txt`) and run:

```
./prac3 < input.txt
```

This approach is especially useful for automated grading on Eustis, since it allows your program's output to be compared directly against an expected output file.

Comparing Your Output to an Expected File

After running your program, you should verify that its output exactly matches the expected results. The contents of this expected file are the same as what is specified in the **Sample Output** section of the instructions. On the Eustis server, this can be done using the `diff` command.

The following checks whether your program's output is identical to the expected output file:

```
./prac3 < input.txt | diff - output.txt
```

If the files are identical, no message is shown. If there are differences, `diff` will display the mismatched lines. The grading system on Eustis relies on this type of comparison to check your program's output against the official expected output. Practicing with `diff` yourself ensures you will catch formatting errors (such as missing colons, extra spaces, or misplaced newlines) before submitting.

Guide Questions

1. Why do we need to include a leading space before %c when using `scanf` to read characters? What problem does this solve? What happens if this space is removed?
2. Why is it important to check whether the input character is a letter before performing arithmetic on it? What would happen if we skipped this step?
3. What happens to the `for` loop if you instead initialize `i = 0` and use `i < T` as the condition? How would it affect the number before the `:` in each case? Rewrite the loop in **four** different ways by combining: (1) initializing `i` to either `0` or `1`; and (2) using either `<` or `<=` as the condition.
4. What happens if you enter more than one character (e.g., "AB") on a single test case line? How does `scanf(" %c", &letter)` handle it compared to entering one character per line? Enter this after providing the value for `T`.
5. Rewrite the program using a `while` loop instead of a `for` loop. What changes are needed? Does it affect the program's behavior? What are the pros and cons of each style?

Sample Solution

```c
#include <stdio.h>
#include <ctype.h>

int main(void) {
    int T;      // number of test cases
    int i;      // used for the loop
    int pos;    // for the solution
    char letter;

    // read the number of test cases
    scanf("%d", &T);

    // counter-controlled loop where there are T iterations
    for(i = 1; i <= T; i++) {
        // read user input, notice the leading space before %c
        scanf(" %c", &letter);

        // common output, so print first
        printf("%d: ", i);

        // check if alpha is a letter
        if( isalpha(letter) ) {
            // compute the position, but transform to common text case
            pos = toupper(letter)-'A'+1;

            printf("%d", pos);
        }
        else {
            printf("Invalid");
        }

        // common output, so print last
        printf("\n");
    }

    return 0;
}
```