Lab Practice Problem 2

Filename: `practice2_surname.c`

In this activity, you will apply modular arithmetic to solve a problem in C. Write a program that reads a single non-negative three-digit integer $(100 \leq \texttt{num} \leq 999)$ and computes its reverse. This activity can be completed without using any conditional or looping statements. By the end, the process will reveal a hidden text code.

Try It Out

Start by extracting each digit and constructing the reversed number as you go. When done, print the reversed number. Ensure that any leading zeros are not displayed. For example:

$$123 \rightarrow 321 \qquad\qquad 300 \rightarrow 3$$

Next, compute and print the sum of the original number and its reverse. For example:

$$123 + 321 = 444 \qquad\qquad 300 + 3 = 303$$

Finally, translate each digit of the reversed number into a letter, where 0 corresponds to `A`, 1 to `B`, 2 to `C`, and so on. You must always print three letters. If the reversed number has fewer than three digits, left-pad it with zeros before conversion. The resulting text is the hidden code. For example:

$321 \rightarrow$ `DCB` $\qquad\qquad$ 3 (becomes 003 after left-padding) $\rightarrow$ `AAD`

Two sample runs are shown below. Anything that is underlined represents user input.

Sample Run 1

```
Enter Number: 123
Reversed: 321
Sum: 444
Hidden Code: DCB
```

Sample Run 2

```
Enter Number: 300
Reversed: 3
Sum: 303
Hidden Code: AAD
```

Guide Questions

1. Test your program with the input `387`. What 3-letter code is produced?
2. Consider reverse-engineering the process: what input would be needed to produce the code `BIG`? Is that possible within the given rules? What about the code `DAY`?
3. The program assumes that the input is a number between 100 and 999, inclusive. Why is this range helpful? What happens if the user enters a value outside of it? Try entering the following values: `5`, `10`, `1234`, or `12345`. Are the outputs you observe in these cases expected?
4. Extend your program so that the hidden code is printed in lowercase if the sum is odd and uppercase if the sum is even. Try to solve this without using explicit `if`-statements. For example, the hidden code for Sample Run 2 would have been `aad`.
5. In ASCII, there is a fixed number you can add to an uppercase letter to obtain its lowercase equivalent (i.e., offset). Determine this number experimentally using code (without looking up a chart). Is this difference the same for all letters?
6. How would you extend your program to handle inputs with 4 digits or 5 digits? What changes would be needed in your approach?

Sample Solution

```c
#include <stdio.h>

int main(void) {
    int num;
    int tmp, rem;       // Temporary variable and for Remainder
    int acc = 0;        // An accumulator needs to be initialized

    printf("Enter Number: ");
    scanf("%d", &num);

    // assign to a temporary variable to preserve user input
    tmp = num;

    // Round 1: Ones Digit  (Notice the repetition)
    rem = tmp % 10;
    //printf("%d\n", rem);
    tmp = tmp / 10;
    acc = (acc * 10) + rem;

    // Round 2: Tens Digit
    rem = tmp % 10;
    //printf("%d\n", rem);
    tmp = tmp / 10;
    acc = (acc * 10) + rem;

    // Round 3: Hundreds Digit
    rem = tmp % 10;
    //printf("%d\n", rem);
    tmp = tmp / 10;
    acc = (acc * 10) + rem;

    // Print Reversed
    printf("Reversed: %d\n", acc);

    int sum = num + acc;
    // Print Sum
    printf("Sum: %d\n", sum);

    // Extract Letters
    char alpha = 'A';
    //int offset = 'a' - 'A';   // GQ#5 offset +32
    //alpha = (char)( alpha + (offset*(sum % 2)) ); // Even/Odd Logic

    // Implicitly Takes Care of the Left-Padding (with type-casting)
    char ch3 = (char)(acc / 1 % 10) + alpha;        // Ones
    char ch2 = (char)(acc / 10 % 10) + alpha;       // Tens
    char ch1 = (char)(acc / 100 % 10) + alpha;      // Hundreds

    // Print the Hidden Code
    printf("Hidden Code: %c%c%c\n", ch1, ch2, ch3);

    return 0;
}
```