Module 03 - Unit 3: Lab

ITSC 2181 – Introduction to Computer Systems – Fall 2023

Objectives

- Practice writing small C programs, including identifying and correcting C compiler error messages and warnings.
- Utilize pointers in C.
- Manipulate strings using functions from the string library (`string.h`).
- Implement functions to process and modify data in C.

General Instructions

- **Identification:** Do not include your email or user ID in your program. Use your UNC Charlotte 800# to identify your code if necessary.
- **File Management:** Each program must be in its own source code file with a `.c` extension.
- **Testing:** Thoroughly test your code before submission to ensure correctness.
- **Compilation:**
  - To earn any credit, your program must compile successfully.
  - Comment out lines with errors to obtain partial credit for incomplete work.
  - Programs must compile cleanly without errors or warnings to receive full credit.
- **Integrity:**
  - Complete the work independently.
  - Do not use external resources (Internet, AI, friends, etc.).
  - For assistance, consult the instructor, TA/IA, or the CCI Tutoring Center.

- **Submission:** Upload all your `.c` files to Canvas.

Program 1: String Capitalization (50 points)

Description: Write a C program named `capitalize_string.c` that performs the following tasks:

1. **Function Implementation:**
    - Implement a function with the following prototype:

      ```
      void capitalize(char *str);
      ```

    - This function should capitalize all alphabetic characters in the input string `str`.
    - Only use array indexing to access and modify each character in the string.
    - Utilize the `isalpha` and `toupper` functions from the C Standard Library (`ctype.h`) to check and convert characters.
    - Do **not** use any other C Standard Library functions except for `printf` and `scanf`.
2. **Main Function:**
    - Test the `capitalize` function with multiple strings.
    - Example test cases:

      ```
      char the_str[] = "test";
      capitalize(the_str);
      printf("%s\n", the_str); // Expected Output: TEST

      char the_str2[] = "This IS a tesT!";
      capitalize(the_str2);
      printf("%s\n", the_str2); // Expected Output: THIS IS A TEST!
      ```

3. **Additional Testing:**
    - Ensure your program works with various strings, including those containing non-alphabetic characters.

Sample Output:

```
TEST
THIS IS A TEST!
```

Program 2: Bills Calculation (50 points)

Description: Write a C program named `bills_needed.c` that calculates the minimum number of bills needed to make up a given dollar amount. The program should handle $20, $10, $5, and $1 bills.

1. **User Input:**
    - Prompt the user to enter a whole dollar amount (no cents).

    ```
    Enter dollar amount to pay: 93
    ```

2. **Bill Calculation:**
    - Implement a function with the following prototype:

    ```
    void calc_bills(int dollar_amount, int *twenties, int *tens, int *fives, int *ones);
    ```

    - This function should calculate the number of each bill denomination needed to make up the entered amount using the smallest number of bills possible.
    - Use pointers to return the count of each bill denomination.
    - **Hint:**
        - Start by dividing the amount by 20 to determine the number of $20 bills.

        - Subtract the equivalent amount and repeat the process for $10, $5, and $1 bills.

3. **Output:**

    - Display the number of each bill required.

```
You need:
$20 dollar bills: 4
$10 dollar bills: 1
$5 dollar bills: 0
$1 dollar bills: 3
```

Sample Runs:

- **Input:**

```
Enter dollar amount to pay: 200
```

  Output:

```
You need:
$20 dollar bills: 10
$10 dollar bills: 0
$5 dollar bills: 0
$1 dollar bills: 0
```

- **Input:**

```
Enter dollar amount to pay: 157
```

Output:

```
You need:
$20 dollar bills: 7
$10 dollar bills: 1
$5 dollar bills: 1
$1 dollar bills: 2
```

Grading Rubric

Your submission will be evaluated based on the following criteria:

Logic and Flow of Program - 60%

- **Fully Correct (50 points):** Code compiles without errors and correctly implements all required functionality.
- **Minor Errors (37.5 points):** Code compiles without errors but has minor issues or some required functionality is missing.
- **Major Errors (25 points):** Code has significant issues, compiles with warnings, or lacks major portions of required functionality.
- **No Credit:** Code is completely incorrect, missing, or does not compile.

Output - 30%

- **Fully Correct (30 points):** Output matches the sample formatting and content exactly.
- **Minor Errors (22.5 points):** Output is mostly correct with minor discrepancies.
- **Major Errors (15 points):** Output has significant discrepancies from the expected results.
- **No Credit:** Output is completely incorrect.

Formatting/Organization of Code - 10%

- **Fully Correct (10 points):** Code is well-organized, readable, properly indented, and includes necessary comments.

- **Minor Improvements Needed (7.5 points):** Code is generally readable but may lack some formatting or comments.
- **Major Improvements Needed (5 points):** Code is poorly formatted and difficult to read.
- **No Credit (0 points):** Code lacks organization and readability entirely.

Additional Deductions

- **Code Producing Warnings:** -10% per program.
- **Incorrectly Named Files:** -2 points per file.
- **Academic Integrity Violations:** Any form of cheating will result in a one letter grade reduction, regardless of the assignment's point value.

Remember: Ensure your programs compile without errors or warnings, follow the specified file naming conventions, and adhere to all submission guidelines to maximize your score.

Good luck!