TERM ASSIGNMENT

WINTER 2024

Contents

Submission Policy 2
Assignment #1 3
Introduction 3
Preparation 3
Milestone – 1 3
Development Suggestions 4
Specifications 4
Core Module 4
Functions 5
A1-MS1: Sample Output 9
Milestone – 1 Submission 10

Submission Policy

- **Each Milestone is due by the end of day 23:59 EST (UTC – 5), 7-days from your scheduled lab class**
- **Source (.c) and text (.txt) files that are provided with each milestone MUST be used or your work will not be accepted. Resubmission will be required attracting the respective late deduction based on the date/time submitted relative to the deadline date/time.**
- **Late submissions are accepted, but there is a 25% deduction per day late**
- **All work must be submitted by the matrix submitter – no exceptions**
- **All files you create or modify MUST contain the following declaration at the top of all documents or it will require a resubmission attracting the respsective late deduction based on the date/time resubmitted to the deadline date/time**
- **Coding violations (principles, style guidelines, etc.) deemed unacceptable as set by your instructor will not be accepted and will require a resubmission attracting the respsective late deduction based on the date/time resubmitted to the deadline date/time**

Notes

- Due dates are in effect **even during a holiday**
- You are responsible for **backing up your work regularly**
- It is expected and assumed that for each milestone, you will plan your coding solution by using the computational thinking approach to problem solving and that **you will code your solution based on your defined pseudo code algorithm**.

Assignment #1

Worth: 20% of final grade

Veterinarian Clinic System

Introduction

This assignment has been broken down into critical deadlines called milestones. Implementing projects using milestones will help you stay on target with respect to timelines and balancing out the workload.

By the end of milestone #3, you will have created a basic patient (pet) appointment system for a veterinary clinic. Patient contact information will be managed as well as the scheduling and management of appointments.

Each milestone will build upon the previous, adding more functionality and components. Milestone #1 is focused on providing helper functions that will aid you in the development of the overall solution in future milestones. These functions will streamline your logic and simplify the overall readability and maintainability of your program by providing you with established routines that have been thoroughly tested for reliability and eliminate unnecessary code redundancy (so use them whenever possible and don't duplicate logic already done).

Each milestone will be released weekly and can be downloaded or cloned from GitHub:
https://github.com/Seneca-144100/IPC-Project

Reflections will be graded based on the published rubric:
https://github.com/Seneca-144100/IPC-Project/tree/master/Reflection%20Rubric.pdf

Preparation

Download or clone the Assignment 1 from GitHub.

In the directory: A1/MS1 you will find the Visual Studio project files ready to load. Open the project (**a1ms1.vcxproj**) in Visual Studio.

Note: the project will contain only one source code file which is the main tester "**a1ms1.c**".

Milestone – 1

Milestone-1 includes a unit tester (***a1ms1.c***). A unit tester is a program which invokes your functions, passing them known parameter values. It then compares the results returned by your functions with the correct results to determine if your functions are working correctly. The tester should be used to confirm your solution meets the specifications for each "helper" function. The helper functions should be thoroughly tested and fail-proof (100% reliable) as they will be used throughout your assignment milestones.

Development Suggestions

You will be developing several functions for this milestone. The unit tester in the file "***a1ms1.c***" assumes these functions have been created and, until they exist, the program will not compile.

Strategy – 1

You can comment out the lines of code in the "***a1ms1.c***" file where you have not yet created and defined the referenced function. You can locate these lines in the function definitions (after the main function) and for every test function, locate the line that calls the function you have not yet developed and simply comment the line out until you are ready to test it.

Strategy – 2

You can create "empty function shells" to satisfy the existence of the functions but give them no logic until you are ready to program them. These empty functions are often called *stubs*.

Review the specifications below and identify every function you need to develop. Create the necessary function prototypes (placed in the .h header file) and create the matching function definitions (placed in the .c source file), only with empty code blocks (don't code anything yet). In cases where the function MUST return a value, hardcode (temporarily until you code the function later) a return value so your application can compile.

Specifications

Milestone-1 will establish the function "helpers" we will draw from as needed throughout the three milestones. These functions will handle routines that are commonly performed (greatly reduces code redundancy) and provide assurance they accomplish what is expected without fail (must be reliable).

Core Module

1. Create a module called "core". To do this, you will need to create two files: "***core.h***" and "***core.c***" and add them to the Visual Studio project.
2. The **header file (.h)** will contain the function prototypes, while the **source file (.c)** will contain the function definitions (the logic and how each function works).
   - Copy and paste the **commented section** provided for you in the ***a1ms1.c*** file (top portion) to all files you create
   - Fill in the information accordingly
3. The "***core.c***" file will require the usual **standard input output** system library as well as the new user library "***core.h***", so be sure to include these.
4. Review the "***a1ms1.c***" tester file and examine each defined tester function (after the main function). Each tester function is designed to test a specific helper function.
5. Two (2) functions are provided for you. Here are the function prototypes you must copy and place into the "***core.h***" header file:

```
// Clear the standard input buffer
void clearInputBuffer(void);

// Wait for user to input the "enter" key to continue
void suspend(void);
```

Functions

The source code file "***core.c***" must contain the function definitions (copy and place the function definitions below in the "***core.c***" file):

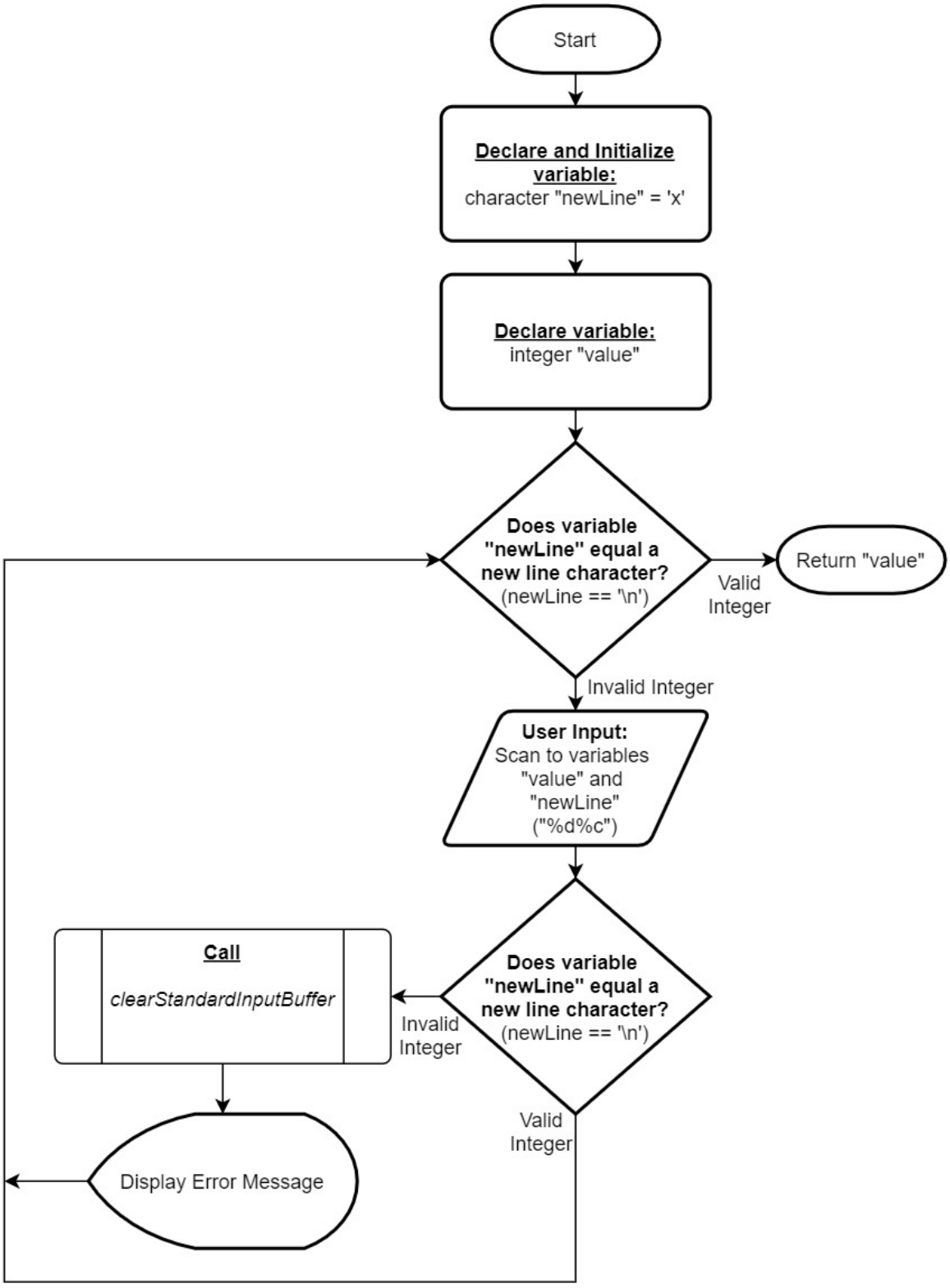
```
// As demonstrated in the course notes:
// https://intro2c.sdds.ca/D-Modularity/input-functions#clearing-the-buffer
// Clear the standard input buffer
void clearInputBuffer(void)
{
    // Discard all remaining char's from the standard input buffer:
    while (getchar() != '\n')
    {
        ; // do nothing!
    }
}
```

```
// Wait for user to input the "enter" key to continue
void suspend(void)
{
    printf("<ENTER> to continue...");
    clearInputBuffer();
    putchar('\n');
}
```

6. Each function briefly described below will require a function prototype to be placed in the "***core.h***" file, and their respective function definitions in the "***core.c***" file.
   The function identifiers (names) are provided for you however **you are responsible for constructing the full function prototype and definitions** based on the descriptions below (there are **six (6) functions** in total):

   - Function: **inputInt**
     This function must:
     - return an integer value and receives no arguments.
     - get a valid integer from the keyboard.
     - display an error message if an invalid value is entered (review the sample output for the appropriate error message)
     - guarantee an integer value is entered and returned.
     - **Hint:** You can use scanf to read an integer and a character ("%d%c") in one call and then assess if the second value is a newline character. If the second character is a newline (the result of an <ENTER> key press), scanf read the first value successfully as an integer.

- If the second value (character) is not a newline, the value entered was not an integer or included additional non-integer characters. If any invalid entry occurs, your function should call the ***clearInputBuffer*** function, followed by displaying an error message and continue to prompt for a valid integer. Review the flowchart below that describes this process.



- Function: **inputIntPositive**
  This function must:
  - return an integer value and receives no arguments.
  - perform the same operations as **inputInt** but validates the value entered is greater than 0.
  - display an error message if the value is a zero or less (review the sample output for the appropriate error message).
  - continue to prompt for a value until a value is greater than 0.
  - guarantee a positive integer value is entered and returned.

- Function: **inputIntRange**
  This function must:
  - return an integer value and receives two arguments:
    - First argument represents the **lower-bound** of the permitted range.
    - Second argument represents the **upper-bound** of the permitted range.

    Note:
    - A range is a set of numbers that includes the upper and lower limits (bounds)
    - You must provide **meaningful** parameter identifiers (names)
  - performs the same operations as **inputInt** but validates the value entered is between the two arguments received by the function (**inclusive**).
  - display an error message if the value is outside the permitted range (review the sample output for the appropriate error message).
  - continue to prompt for a value until a value is between the permitted range (inclusive)
  - guarantee an integer value is entered within the range (inclusive) and returned.

- Function: **inputCharOption**
  This function must:
  - return a single character value and receives one argument:
    - an unmodifiable C string array representing a list of valid characters.

    Note: You must provide a **meaningful** parameter identifier (name)
  - get a single character value from the keyboard.
  - validate the entered character matches any of the characters in the received C string argument.
    Reminder: A C string will have a ***null terminator*** character marking the end of the array
  - display an error message if the entered character value is not in the list of valid characters (review the sample output for the appropriate error message)
    Note: Include in the error message the C string permitted characters
  - Continue to prompt for a single character value until a valid character is entered.
  - Guarantee a single character value is entered within the list of valid characters (as defined by the C string argument received) and returned.

- Function: **inputCString**
  The purpose of this function is to obtain user input for a C string value with a length (number of characters) in the character range specified by the 2nd and 3rd arguments received (inclusive).

  This function:
  - must receive three (3) arguments and therefore needs three (3) parameters:
    - 1st parameter is a character pointer representing a C string
      ***Note**: Assumes the argument has been sized to accommodate at least the upper-bound limit specified in the 3rd argument received*
    - 2nd parameter represents an integral value of the **minimum** number of characters the user-entered value must be.
    - 3rd parameter represents an integral value of the **maximum** number of characters the user-entered value can be.
  - does not ***return*** a value, but does return a C string via the 1st argument parameter pointer.
  - must validate the entered number of characters is within the specified range. If not, display an error message (review the sample output for the appropriate error message).
    Note: If the 2nd and 3rd arguments are the same value, this means the C string entered must be a specific length.
  - must continue to prompt for a C string value until a valid length is entered.
  - guarantee's a C string value is entered containing the number of characters within the range specified by the 2nd and 3rd arguments (and return via the 1st argument pointer).

  [IMPORTANT]
  *You are NOT to use any of the **string library functions**; you must manually determine the entered C string length using a conventional iteration construct.*

- Function: **displayFormattedPhone**
  The purpose of this function is to display an array of 10-character digits as a formatted phone number.
  This function:
  - must receive one (1) argument and therefore requires one (1) parameter:
    - 1st parameter is an unmodifiable character pointer representing a C string..
  - does not **return** a value..
  - should not assume a valid C string array, and therefore, should carefully validate the argument char array to determine:
    - it is exactly 10 characters long
    - only contains digits (0-9)
  - should display "`(___)___-____`" when the argument C string char array is not a 10-character all digit value.
  - should display the phone number in the following format when it is a valid C string phone number: "`(###)###-####`" (where each # is the character digit from the C string argument char array).
  - NOTE: Do not add a newline character at the beginning or end of the displayed value.

A1-MS1: Sample Output

```
Assignment 1 Milestone 1: Tester
================================

TEST #1 - Instructions:
1) Enter the word 'error'  [ENTER]
2) Enter the number '-100' [ENTER]
:>error
Error! Input a whole number: -100
////////////////////////////////////////
TEST #1 RESULT: *** PASS ***
////////////////////////////////////////

TEST #2 - Instructions:
1) Enter the number '-100' [ENTER]
2) Enter the number '200'  [ENTER]
:>-100
ERROR! Value must be > 0: 200
////////////////////////////////////////
TEST #2 RESULT: *** PASS ***
////////////////////////////////////////

TEST #3 - Instructions:
1) Enter the word 'error' [ENTER]
2) Enter the number '-4'  [ENTER]
3) Enter the number '12'  [ENTER]
4) Enter the number '-3'  [ENTER]
:>error
Error! Input a whole number: -4
ERROR! Value must be between -3 and 11 inclusive: 12
ERROR! Value must be between -3 and 11 inclusive: -3
////////////////////////////////////////
TEST #3 RESULT: *** PASS ***
////////////////////////////////////////

TEST #4 - Instructions:
1) Enter the number '14' [ENTER]
:>14
////////////////////////////////////////
TEST #4 RESULT: *** PASS ***
////////////////////////////////////////

TEST #5 - Instructions:
1) Enter the character 'R' [ENTER]
2) Enter the character 'e' [ENTER]
3) Enter the character 'p' [ENTER]
4) Enter the character 'r' [ENTER]
:>R
ERROR: Character must be one of [qwErty]: e
ERROR: Character must be one of [qwErty]: p
ERROR: Character must be one of [qwErty]: r
////////////////////////////////////////
TEST #5 RESULT: *** PASS ***
////////////////////////////////////////

TEST #6: - Instructions:
1) Enter the word 'horse'   [ENTER]
```

```
2) Enter the word 'chicken' [ENTER]
3) Enter the word 'SENECA'  [ENTER]
:>horse
ERROR: String length must be exactly 6 chars: chicken
ERROR: String length must be exactly 6 chars: SENECA
//////////////////////////////////////
TEST #6 RESULT: SENECA (expected result: SENECA)
//////////////////////////////////////

TEST #7: - Instructions:
1) Enter the words 'Seneca College' [ENTER]
2) Enter the word 'CATS'            [ENTER]
:>Seneca College
ERROR: String length must be no more than 6 chars: CATS
//////////////////////////////////////
TEST #7 RESULT: CATS (expected result: CATS)
//////////////////////////////////////

TEST #8: - Instructions:
1) Enter the word 'dogs'  [ENTER]
2) Enter the word 'HORSES' [ENTER]
:>dogs
ERROR: String length must be between 5 and 6 chars: HORSES
//////////////////////////////////////
TEST #8 RESULT: HORSES (expected result: HORSES)
//////////////////////////////////////

//////////////////////////////////////
TEST #9 RESULT:
Expecting (___)___-____ => Your result: (___)___-____
Expecting (___)___-____ => Your result: (___)___-____
Expecting (___)___-____ => Your result: (___)___-____
Expecting (___)___-____ => Your result: (___)___-____
Expecting (___)___-____ => Your result: (___)___-____
Expecting (___)___-____ => Your result: (___)___-____
Expecting (416)111-4444 => Your result: (416)111-4444
//////////////////////////////////////

Assignment #1 Milestone #1 testing completed!
```

Milestone – 1 Submission

1. Upload (file transfer) your all header and source files:
   `core.h  core.c  a1ms1.c`
2. Login to matrix in an SSH terminal and change directory to where you placed your source code.
3. Manually compile and run your program to make sure everything works properly:
   ```
   gcc -Wall a1ms1.c core.c -o ms1 <ENTER>
   ```
   If there are no error/warnings are generated, execute it: ***`ms1`*** *`<ENTER>`*
4. Run the submission command below (replace **profname.proflastname** with **<u>your professors</u>** Seneca userid and replace **NAA** with your section):
   ```
   ~profName.proflastname/submit 144a1ms1/NAA_ms1 <ENTER>
   ```
5. Follow the on-screen submission instructions.