

Assignment #1

Worth: 10% of final grade

Account Ticketing System

Milestone	Worth	Due Date	Submission Required
1	10%	<i>(Suggested Target: October 29th)</i>	NO
2	40%	November 5th (23:59 EST)	YES
3	10%	<i>(Suggested Target: November 8th / 9th)</i>	NO
4	40%	November 12th (23:59 EST)	YES

Introduction

This is the first of two assignments. Assignment #2 builds upon and extends Assignment #1. Each assignment is 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 assignment #2 (milestone #4), you will have created a basic account and ticketing system. Think of the ticketing component as a tracking system for customer reported problems. Customers will phone or email for support when they need to report a problem. The person handling the support request (agent) will create a ticket for the request that contains the details of the problem. Each ticket also includes a brief message log of the dialogue exchanged between the customer and the agent. The agent is also responsible for maintaining basic account management that includes customer contact information.

The account ticketing system application will be incrementally built (adding more functionality and components) with each assignment milestone. Assignment #1 milestones 1-2 are 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).

Preparation

Download or clone the Assignment 1 (A1) from <https://github.com/Seneca-144100/IPC-Project>

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 (Worth 10%, Target Due Date: October 29th)

Milestone – 1 does not require a submission and does not have a specific deadline, however, you should target to have this part completed no later than **October 29th** to ensure you leave enough time to complete Milestone – 2 which must be submitted and is due **November 5th**.

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. An optional matrix submitter tester is also at your disposal so you can receive additional confirmation that your solution meets the minimum milestone requirements.

Note: Inevitably, all these functions will be tested as part of the Milestone #2 submission

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 these two assignments. 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).

1. Create a module called “commonHelpers”. To do this, you will need to create two files: ***“commonHelpers.h”*** and ***“commonHelpers.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).
 - For each of these files, create a **commented section at the top** containing the following information (you may want to use what was similarly provided to you in the workshops):
 - Assignment #1 Milestone #1
 - Your full name
 - Your student ID number and Seneca email address
 - Your course section code

3. The “**commonHelpers.c**” file will require the usual standard input output system library as well as the new user library “**commonHelper.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 “**commonHelper.h**” header file:

```
int currentYear(void);
void clearStandardInputBuffer(void);
```

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

```
// Uses the time.h library to obtain current year information
// Get the current 4-digit year from the system
int currentYear(void)
{
    time_t currentTime = time(NULL);
    return localtime(&currentTime)->tm_year + 1900;
}
```

***Note:** You will need to **#include <time.h>** system library for the above function to compile.

```
// As demonstrated in the course notes:
https://ict.senecacollege.ca/~ipc144/pages/content/formi.html#buf
// Empty the standard input buffer
void clearStandardInputBuffer(void)
{
    while (getchar() != '\n')
    {
        ; // On purpose: do nothing
    }
}
```

6. Each function briefly described below will require a function prototype to be placed in the “**commonHelpers.h**” file, and their respective function definitions in the “**commonHelpers.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 **seven (7) functions** in total):

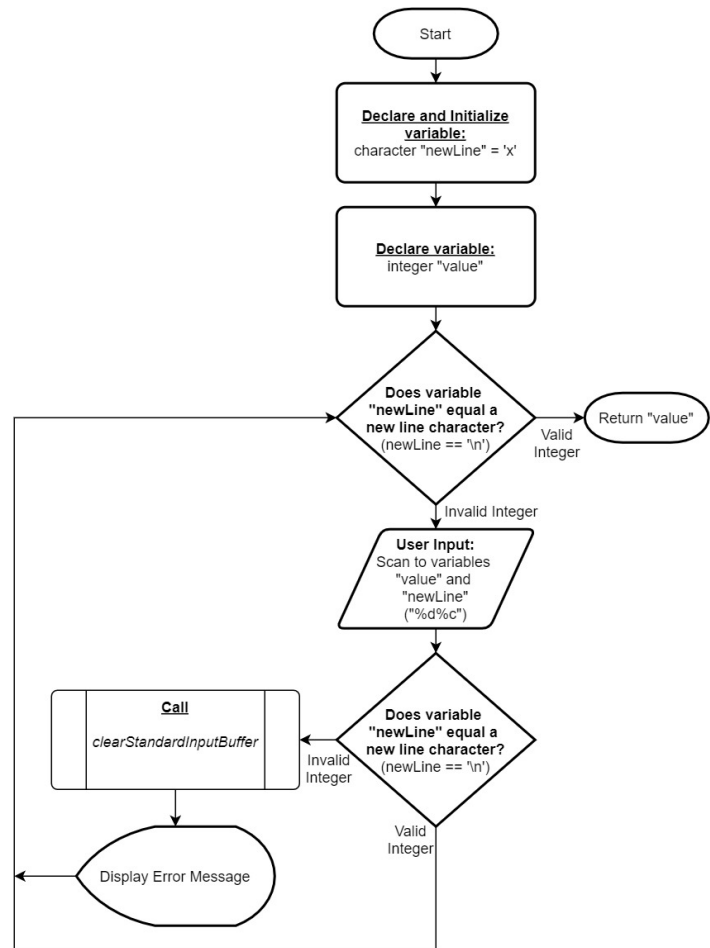
- Function: getInteger

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. **This technique can be used in other “get” functions you need to create for different data types!**

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 ***clearStandardInputBuffer*** function, followed by displaying an error message and continue to prompt for a valid integer. Review the following flowchart that describes this process:



- Function: **getPositiveInteger**

This function must:

- return an integer value and receives no arguments.
- perform the same operations as **getInteger** 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: **getDouble**

This function must:

- return a double value and receives no arguments.
- get a valid double value from the keyboard.
- display an error message if an invalid value is entered (review the sample output for the appropriate error message)
- guarantee a double value is entered and returned.
- **Hint:** Process is the same as described in the flowchart for **getInteger** only this is for a double type

- Function: **getPositiveDouble**

This function must:

- return a double value and receives no arguments.
- perform the same operations as **getDouble** 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 double value is entered and returned.

- Function: **getIntFromRange**

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.
- performs the same operations as **getInteger** 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.

Note: You must provide **meaningful** parameter identifiers (names)

Note

You will need to review the supplemental document “**Introduction to C Strings**”

(<https://github.com/Seneca-144100/IPC-Project/tree/master/A1/Introduction%20to%20C%20Strings.pdf>)

before attempting to do the next two functions

- Function: **getCharOption**

This function must:

- return a single character value and receives one argument:
 - an unmodifiable C string array representing a list of valid characters.
- get a single character value from the keyboard.
- validate the entered character matches any of the characters in the received C string argument.
- 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)
- 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.

Note: You must provide a **meaningful** parameter identifier (name)

Reminder: A C string will have a **null terminator** character marking the end of the array

Note: Include in the error message the C string permitted characters

- Function: **getCString**

The purpose of this function is to obtain user input for a C string value with a length (number of characters) between the character range specified in 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.

A1-MS1: Sample Output

```
Assignment 1 Milestone 1
=====

TEST #1 - Instructions:
1) Enter the word 'error' [ENTER]
2) Enter the number '-100' [ENTER]
:>error
ERROR: Value must be an integer: -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 a positive integer greater than zero: 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: Value must be an integer: -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 word 'error' [ENTER]
2) Enter the number '-150.75' [ENTER]
:>error
ERROR: Value must be a double floating-point number: -150.75
////////////////////////////////////
TEST #5 RESULT: *** PASS ***
////////////////////////////////////

TEST #6 - Instructions:
1) Enter the number '-22.11' [ENTER]
2) Enter the number '225.55' [ENTER]
:>-22.11
ERROR: Value must be a positive double floating-point number: 225.55
////////////////////////////////////
TEST #6 RESULT: *** PASS ***
////////////////////////////////////

TEST #7 - Instructions:
1) Enter the character 'R' [ENTER]
2) Enter the character 'p' [ENTER]
3) Enter the character 'r' [ENTER]
:>R
ERROR: Character must be one of [qwerty]: p
ERROR: Character must be one of [qwerty]: r
////////////////////////////////////
TEST #7 RESULT: *** PASS ***
////////////////////////////////////
```

```

TEST #8: - Instructions:
1) Enter the word 'horse' [ENTER]
2) Enter the word 'SENECA' [ENTER]
:>horse
ERROR: String length must be exactly 6 chars: SENECA
////////////////////////////////////
TEST #8 RESULT: SENECA (Answer: SENECA)
////////////////////////////////////

TEST #9: - Instructions:
1) Enter the words 'Seneca College' [ENTER]
2) Enter the word 'IPC' [ENTER]
:>Seneca College
ERROR: String length must be no more than 6 chars: IPC
////////////////////////////////////
TEST #9 RESULT: IPC (Answer: IPC)
////////////////////////////////////

TEST #10: - Instructions:
1) Enter the word 'ipc' [ENTER]
2) Enter the word 'SCHOOL' [ENTER]
:>ipc
ERROR: String length must be between 4 and 6 chars: SCHOOL
////////////////////////////////////
TEST #10 RESULT: SCHOOL (Answer: SCHOOL)
////////////////////////////////////

Assignment #1 Milestone #1 completed!

```

Milestone – 1 Submission

1. ***This is a test submission for verifying your work only*** – no files will be submitted to your instructor – this will test your functions and confirm the outputs match to the expected output.
2. Upload (file transfer) all your header and source files:
 - **commonHelpers.h**
 - **commonHelpers.c**
 - **a1ms1.c**

3. Login to matrix in an SSH terminal and change directory to where you placed your source code.
4. Manually compile and run your program to make sure everything works properly:

```
gcc -Wall a1ms1.c commonHelpers.c -o ms1 <ENTER>
```

If there are no error/warnings are generated, execute it: ms1 <ENTER>

5. Run the submission command below (replace **profname.proflastname** with **your professors** Seneca userid and replace **NAA** with your section):

```
~profName.proflastname/submit 144a1ms1/NAA_ms1 <ENTER>
```

6. Follow the on-screen submission instructions.

Milestone – 2 (Worth 20%, Due Date: November 5th)

In Milestone – 2, will expand on what was done from Milestone – 1. You will need to copy the header and source code files for the “**commonHelpers**” module to the Milestone – 2 directory and include them in the Milestone – 2 Visual Studio project.

You will begin this milestone by creating some new data types in an “**account.h**” header file, and the main function creates variable instances of those new types. You will prompt for user input and store the entered values to the appropriate variable members. It is expected you will call functions from the common helper library where appropriate. After data has been entered and stored, you will display the information back to the user in a tabular format (review the sample output section).

Specifications

1. In the “**account.h**” header file, you will create three (3) new data types that will be used to represent an account and related person demographic and user login information. These new data types will be defined in the module “account” and will require you to create another header file called “**account.h**” (add this new file to the Visual Studio project). We will NOT be creating a source code file for this module yet (this will be done in a later milestone). Create the following new structures:

“Person”

- This structure has **four (4) members**. You must provide the appropriate data type and meaningful identifiers/names for each described member:
 - A C string that represents the person's full name (first name, middle name if applicable, and surname) and should be able to store up to thirty (30) displayable characters.
 - An integer type that represents the birth year of a customer.
 - A double floating-point type that represents the household income.
 - A C string that represents the country the customer resides and should be able to store up to thirty (30) displayable characters.

“UserLogin”

- This structure has **two (2) members**. You must provide the appropriate data type and meaningful identifiers/names for each described member:
 - A C string that represents the user login name and should be able to store up to ten (10) displayable characters.
 - A C string that represents the password for the user and should be able to store up to eight (8) displayable characters.

“Account”

- This structure has **two (2) members**. You must provide the appropriate data type and meaningful identifiers/names for each described member:
 - An integer type that represents the account number associated to a customer.
 - A single character type that represents the account type (for example, an ‘A’ would represent a customer service agent, and a ‘C’ would represent a customer).

2. In the “**a1ms2.c**” source code file, you will find three variables declared (“**account**”, “**person**”, and “**login**”) which are instances of the new types you created in the “**account.h**” header file. You need to provide the necessary code that will assign user input to each of the variable members. Use the example output and the source code comments to help guide you in accomplishing this task.

Reminder

You should be utilizing the common helper functions you created in Milestone – 1 as much as possible where appropriate to help you do this task!

- The last task you need to do, is complete the “**displayAccount**” function definition located after the main function. The formatted table header is provided for you. To help you format the data values to properly align with the header, you can use the following format specifiers in your printf statement for the respective fields:

Column Name	Format Specifier
Acct#	%05d
Acct.Type	%-9s
Birth	%5d
Full Name	%-15s
Income	%11.2lf
Country	%-10s
Login	%-10s
Password	%8s

You will need to reference the appropriate arguments received by this function and their respective members to provide your printf function with the required data.

A1-MS2: Sample Output

Assignment 1 Milestone 2

=====

TEST #1 - Instructions:

1) Enter the word 'error' [ENTER]

2) Enter the number '-100' [ENTER]

:>error

ERROR: Value must be an integer: -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 a positive integer greater than zero: 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: Value must be an integer: -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 word 'error' [ENTER]
2) Enter the number '-150.75' [ENTER]
:>error
ERROR: Value must be a double floating-point number: -150.75
////////////////////////////////
TEST #5 RESULT: *** PASS ***
////////////////////////////////

TEST #6 - Instructions:
1) Enter the number '-22.11' [ENTER]
2) Enter the number '225.55' [ENTER]
:>-22.11
ERROR: Value must be a positive double floating-point number: 225.55
////////////////////////////////
TEST #6 RESULT: *** PASS ***
////////////////////////////////

TEST #7 - Instructions:
1) Enter the character 'R' [ENTER]
2) Enter the character 'p' [ENTER]
3) Enter the character 'r' [ENTER]
:>R
ERROR: Character must be one of [qwerty]: p
ERROR: Character must be one of [qwerty]: r
////////////////////////////////
TEST #7 RESULT: *** PASS ***
////////////////////////////////

TEST #8: - Instructions:
1) Enter the word 'horse' [ENTER]
2) Enter the word 'SENECA' [ENTER]
:>horse
ERROR: String length must be exactly 6 chars: SENECA
////////////////////////////////
TEST #8 RESULT: SENECA (Answer: SENECA)
////////////////////////////////

TEST #9: - Instructions:
1) Enter the words 'Seneca College' [ENTER]
2) Enter the word 'IPC' [ENTER]
:>Seneca College
ERROR: String length must be no more than 6 chars: IPC
////////////////////////////////
```

```
TEST #9 RESULT: IPC (Answer: IPC)
////////////////////////////////////
```

TEST #10: - Instructions:

1) Enter the word 'ipc' [ENTER]

2) Enter the word 'SCHOOL' [ENTER]

:>ipc

ERROR: String length must be between 4 and 6 chars: SCHOOL

```
////////////////////////////////////
```

TEST #10 RESULT: SCHOOL (Answer: SCHOOL)

```
////////////////////////////////////
```

Account Data Input

```
-----
```

Enter the account number: 50001 Account

ERROR: Value must be an integer: 50001

Enter the account type (A=Agent | C=Customer): Agent

ERROR: Character must be one of [AC]: c

ERROR: Character must be one of [AC]: a

ERROR: Character must be one of [AC]: A

Person Data Input

```
-----
```

Enter the person's full name (30 chars max): Will Smith

Enter birth year (current age must be between 18 and 110): 2004

ERROR: Value must be between 1911 and 2003 inclusive: 1910

ERROR: Value must be between 1911 and 2003 inclusive: 1988

Enter the household Income: \$1 million 5 hundred

ERROR: Value must be a double floating-point number: -500.25

ERROR: Value must be a positive double floating-point number: 0.0

ERROR: Value must be a positive double floating-point number: 188222.75

Enter the country (30 chars max.): CANADA

User Login Data Input

```
-----
```

Enter user login (10 chars max): Williamson Willie

ERROR: String length must be no more than 10 chars: MIBAgent-J

Enter the password (must be 8 chars in length): jump

ERROR: String length must be exactly 8 chars: jumping

ERROR: String length must be exactly 8 chars: seventeen

ERROR: String length must be exactly 8 chars: agent007

Acct#	Acct.Type	Full Name	Birth	Income	Country	Login	Password
50001	AGENT	Will Smith	1988	188222.75	CANADA	MIBAgent-J	agent007

Assignment #1 Milestone #2 completed!

Reflection (Worth 20%, Due Date: November 5th)

Academic Integrity

It is a violation of academic policy to copy content from the course notes or any other published source (including websites, work from another student, or sharing your work with others).

Failure to adhere to this policy will result in the filing of a violation report to the Academic Integrity Committee.

Instructions

- Create a text file named “**reflect.txt**” and record your answers to the below questions in this file.
 - Answer each question in sentence/paragraph form unless otherwise instructed.
 - A minimum **300** overall word count is required (does NOT include the question).
 - Whenever possible, be sure to substantiate your answers with a brief example to demonstrate your view(s).
1. From the helper functions library, what function was the most challenging to define and clearly describe the challenge(s) including how you managed to overcome them in the context of the methods used in preparing your logic, debugging, and testing.
 2. Describe how the “helper functions” library contributes toward making the code easier to read and include in your analysis why the library will make your code easier to maintain.
 3. Comment on why the C programming language provides a programmer the ability to create new data types (struct) and what advantages does this have? Are there limitations in the construction of a new data type – if so, what specifically?

Reflections will be graded based on the published rubric:

<https://github.com/Seneca-144100/IPC-Project/tree/master/Reflection%20Rubric.pdf>

Milestone – 2 Submission

1. Upload (file transfer) your all header and source files including your reflection:
commonHelpers.h commonHelpers.c account.h a1ms2.c reflect.txt
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 a1ms2.c commonHelpers.c -o ms2 <ENTER>
```

*If there are no error/warnings are generated, execute it: **ms2 <ENTER>***
4. Run the submission command below (replace **profname.proflastname** with **your professors** Seneca userid and replace **NAA** with your section):

```
~profName.proflastname/submit 144a1ms2/NAA_ms2 <ENTER>
```
5. Follow the on-screen submission instructions.

Milestone – 3 (Worth 10%, Target Due Date: November 8/9th)

In Milestone – 3, the “**account**” module will be refined and expanded on to include **three (3)** functions which will handle user input for account related data. A new “**accountTicketingUI**” (user interface) module will be introduced starting with **four (4)** functions that will be responsible for displaying account records in a tabular format.

Special Instructions Regarding Header Files (.h)

User library header files (such as the "account.h" file) are commonly "included" in many other source files within the same project. This often will cause build errors due to the duplication of the contents injected by including header files. To prevent these compile-time errors, we need to “**safeguard**” the header file. This is a technique used to instruct the compiler to use only one instance of the header contents even when it is referenced more than once in the project by several other files. This is an advanced topic not covered in this course and will be discussed in more depth in the next level course C++ (OOP244/BTP200). However, to successfully build and compile your project, you will need to apply this safeguarding technique to all the (.h) headers files. This will require three (3) extra lines of code to be applied in each header file.

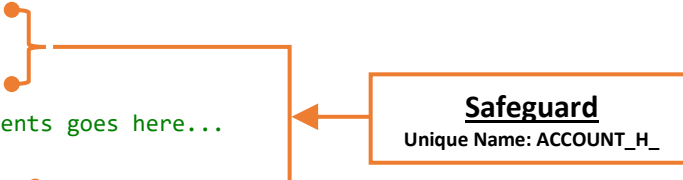
Example safeguard applied to the “**account.h**” header file:

```
//
// Your identification information commented header goes here...
//

#ifndef ACCOUNT_H_
#define ACCOUNT_H_

// Header file contents goes here...

#endif // !ACCOUNT_H_
```



Apply the same technique to all (.h) header files in your project (and any new ones you need to create going forward). It is **IMPORTANT** that you use a **unique name** reflecting the module name (usually derived from the filename) followed by “_H_”. For example, to apply the safeguard to the “**commonHelpers.h**” file, you would do the following:

```
//
// Your identification information commented header goes here...
//

#ifndef COMMON_HELPERS_H_
#define COMMON_HELPERS_H_

// Header file contents goes here...

#endif // !COMMON_HELPERS_H_
```

Specifications

1. In the "**account.h**" header file, you will need to modify the "**Account**" data type to include **two (2) additional members**. You must provide the appropriate data type and meaningful identifiers/names for each described member:
 - A "**Person**" type used to store related details of a person.
 - A "**UserLogin**" type used to store related details of a user login.
2. Each function briefly described below will require a function prototype to be placed in the "**account.h**" file, and the respective function definitions in the "**account.c**" source file.

Note:

- The source file "**account.c**" will need to be created and added to your project. It will also require the appropriate library inclusions to be able to use the helper functions in the "**commonHelpers.h**" file and be able to define the functions prototyped in the "**account.h**" file. Remember to include your commented header that includes your identification information.
 - When coding the function definitions, it is expected you will call functions from the **common helper library where appropriate**.
-

- Function: **getAccount**
 - Receives a modifiable **Account** pointer argument.
 - Does not **return** anything but does return data for an **Account** type via the argument pointer variable.

Functionality

- Displays a title: "*Account Data: New Record*" and is underlined using 40 dashes (-)
 - Prompts the user to enter the account number.
 - Prompts the user to enter the account type.
 - The entered values should be assigned using the pointer argument received by the function.
 - See the sample output for the prompts to be used.
 - Hint: most of this logic and code should be coming from your work done in Milestone 2 found in the a1ms2.c main function
-

- Function: **getPerson**
 - Receives a modifiable **Person** pointer argument.
 - Does not **return** anything but does return data for a **Person** type via the argument pointer variable.

Functionality

- Displays a title: "*Person Data Input*" and is underlined using 40 dashes (-)
- Prompts the user to enter the person's full name.

- Prompts the user to enter the account holder's birth year.
- Prompts the user to enter the household income.
- Prompts the user to enter the country where account holder lives.
- The entered values should be assigned using the pointer argument received by the function.
- See the sample output for the prompts to be used.
- Hint: most of this logic and code should be coming from your work in Milestone 2 found in the a1ms2.c main function

- Function: **getUserLogin**

- Receives a modifiable **UserLogin** pointer argument.
- Does not **return** anything but does return data for a **UserLogin** type via the argument pointer variable.

Functionality

- Displays a title: "User Login Data Input" and is underlined using 40 dashes (-)
- Prompts the user to enter the user login name.
- Prompts the user to enter the user login password.
- The entered values should be assigned using the pointer argument received by the function.
- See the sample output for the prompts to be used.
- Hint: most of this logic and code should be coming from your work in Milestone 2 found in the a1ms2.c main function

3. Create a module called "accountTicketingUI". To do this, you will need to create two files: "**accountTicketingUI.h**" and "**accountTicketingUI.c**" and add them to the Visual Studio project.

- Reminder: You will need to apply a "**safeguard**" to the (.h) header file.

4. Each function briefly described below will require a function prototype to be placed in the "**accountTicketingUI.h**" file, and the respective function definitions in the "**accountTicketingUI.c**" source file.

- Function: **displayAccountDetailHeader**

- Receives no arguments.
- Does not return anything.

Functionality

- Displays a formatted table header (FYI: the related function that produces the data rows under this header is: "displayAccountDetailRecord" described later)
- This function should only display to the screen **eight (8)** column headers with an underline:

Acct#	Acct.Type	Full Name	Birth	Income	Country	Login	Password
-----	-----	-----	-----	-----	-----	-----	-----

- Function: **displayAccountDetailRecord**
 - Receives a unmodifiable **Account** pointer argument.
 - Does not return anything.

Functionality

- Displays a formatted record that aligns to the respective header as defined in function "displayAccountDetailHeader" described earlier)
- Use the following format specifiers in your printf statement for the respective fields:

Column Name	Format Specifier
Acct#	%05d
Acct.Type	%-9s
Disp.Name	%-15s
Birth	%5d
Income	%11.2lf
Country	%-10s
Login	%-10s
Password	%8s

- You will need to reference the respective members of the Account pointer argument variable received by this function to provide your printf function with the required data.

NOTE

- The password must be partially "hidden" with asterisks (*) used for every-other character displayed (**DO NOT** modify the stored password value).

5. Using the comments provided in the "**a1ms3.c**" source file, call the new functions you created in this milestone where appropriate.

Note

- Only the insertion of the function calls is permitted.
 - No other code should be modified or added to the "a1ms3.c" source code file.
 - Some functions require an argument(s); you are limited to using the variable "**account**" already declared for you in the main function – no other variables are to be declared.
6. Review each (.h) header file and supply **brief** but meaningful and concise comments to each function prototype describing what the function does and additional usage information if required.
 - When you are satisfied with your documentation, copy the comments to each function definition in their respective (.c) source code files.

A1-MS3: Sample Output

Assignment 1 Milestone 3 - Tester

=====

Account Data: New Record

Enter the account number: **50001 Account**

ERROR: Value must be an integer: **50001**

Enter the account type (A=Agent | C=Customer): **Agent**

ERROR: Character must be one of [AC]: **c**

ERROR: Character must be one of [AC]: **a**

ERROR: Character must be one of [AC]: **A**

Person Data Input

Enter the person's full name (30 chars max): **Will Smith**

Enter birth year (current age must be between 18 and 110): **2004**

ERROR: Value must be between 1911 and 2003 inclusive: **1910**

ERROR: Value must be between 1911 and 2003 inclusive: **1988**

Enter the household Income: **\$1 million 5 hundred**

ERROR: Value must be a double floating-point number: **-500.25**

ERROR: Value must be a positive double floating-point number: **0.0**

ERROR: Value must be a positive double floating-point number: **188222.75**

Enter the country (30 chars max.): **CANADA**

User Login Data Input

Enter user login (10 chars max): **Williamson Willie**

ERROR: String length must be no more than 10 chars: **MIBAgent-J**

Enter the password (must be 8 chars in length): **jump**

ERROR: String length must be exactly 8 chars: **jumping**

ERROR: String length must be exactly 8 chars: **seventeen**

ERROR: String length must be exactly 8 chars: **agent007**

Acct#	Acct.Type	Full Name	Birth	Income	Country	Login	Password
50001	AGENT	Will Smith	1988	188222.75	CANADA	MIBAgent-J	a*e*t*0*

Account, Person, and User Login test completed!

Milestone – 3 Submission

7. ***This is a test submission for verifying your work only*** – no files will be submitted to your instructor – this will test your functions and confirm the outputs match to the expected output.
8. Upload (file transfer) your all header and source files:
 - **commonHelpers.h & commonHelpers.c**
 - **account.h & account.c**
 - **accountTicketingUI.h & accountTicketingUI.c**
 - **a1ms3.c**
9. Login to matrix in an SSH terminal and change directory to where you placed your source code.

10. Manually compile and run your program to make sure everything works properly:

```
gcc -Wall a1ms3.c commonHelpers.c account.c accountTicketingUI.c -o ms3 <ENTER>
```

If there are no error/warnings are generated, execute it: **ms3** <ENTER>

11. Run the submission command below (replace **profname.proflastname** with your professors Seneca userid and replace **NAA** with your section):

```
~profName.proflastname/submit 144a1ms3/NAA_ms3 <ENTER>
```

12. Follow the on-screen submission instructions.
-

Milestone – 4 (Worth 20%, Due Date: November 12th)

This module will conclude Assignment 1! In Milestone-4, you will be adding more functionality to the "**account**" module that will support the updating of data whereby the user can select a specific field member to change for each of the user-defined data types (**Account**, **Person**, **UserLogin**).

You will expand the "**accountTicketingUI**" module to include a menu-driven model for navigating the application features. The application will provide an "agent" user-type menu for managing account records.

Review the sample output section to get a feel for how the application will flow and what menu options are available.

IMPORTANT

- It is expected you will minimize code redundancy as much as possible by calling appropriate ready-to-use functions that you have already coded (including those coded for this milestone if applicable)

Note: The application will continue to be enhanced and expanded on in Assignment 2.

Specifications

1. The "**a1ms4.c**" source file should not be modified. Examine the "**a1ms4.c**" file.
 - You will notice the main function does not do much. It assigns data to an accounts array and then calls a function "**applicationStartup**" providing it with the array of accounts.
 - The **main** function is the entry point to the application, but you will now create an entry point to the application logic which will be the "**applicationStartup**" function to be coded in the accountTicketingUI module.
 2. Each function briefly described below will require a function prototype to be placed in the "**accountTicketingUI.h**" file, and the respective function definitions in the "**accountTicketingUI.c**" source file.
-

- Function: **applicationStartup**
 - Receives as an argument in **parameter 1**, a modifiable array of type **Account**.
 - Receives as an argument in **parameter 2**, an integer specifying the maximum number of elements in the first argument's array.
 - Does not return anything.

Functionality

- The purpose of this function is to be the entry point to the application logic.
- This function will be responsible for a main loop that will call a **menuLogin** function (see next function description for details) until the user specifies the intention to exit the system.
- The **menuLogin** function will return the index number of the **account array** for the user that is logged in (or -1 if the user wishes to exit the application).
- Using the returned index number, you should check the logged in user account type.
- If the logged in user is an "agent" account type, load the agent main menu by calling the function "**menuAgent**" (described later)
- Only an agent type account should be permitted to login; any other login attempt should display:
`"ERROR: login failed!"`
- When the user wishes to exit the application, the main loop should end and then display the exit message (see example output section for message)

-
- Function: **menuLogin**
 - Receives as an argument in **parameter 1**, an unmodifiable array of type **Account**.
 - Receives as an argument in **parameter 2**, an integer specifying the maximum number of elements in the first argument's array.
 - Returns an integer representing the array index of the **Account** that matches the entered account number or -1 if the user wishes to exit.

Functionality

- Displays a login menu:

```
=====
Account Ticketing System - Login
=====
1) Login to the system
0) Exit application
-----
Selection:
```

- The user must select an option number (1 or 0).
- Selecting "1" (to login to the system), should prompt the user for an integer account number. The accounts array should be searched for a match on the entered account number.
 - If a match is found, the array index position should be returned to the caller of this function.
 - If no match is found, an error message should be displayed:

```
"ERROR: login failed!"
```

- **NOTE:** The single-entry, single-exit principle must be honoured and therefore a function can have only **ONE** *return* statement near the end of the function.
- This process will repeat until either the user selects 0 signifying the user wishes to exit the application, or the entered account number matched to an account in the account array.
- When the user enters 0 to exit the application, you must get user confirmation.

- Function: **menuAgent**

- Receives as an argument in **parameter 1**, a modifiable array of type **Account**.
- Receives as an argument in **parameter 2**, an integer specifying the maximum number of elements in the first argument's array.
- Receives as an argument in **parameter 3**, an unmodifiable pointer to type **Account**. This argument represents the logged-in agent's account details.
- Does not return anything.

Functionality

- Display's the agent's main menu options until the user wishes to logout.
- Display's the logged in agent's "full name" and "account number" (in parenthesis)
- Followed by the main menu options:

```
=====
Account Ticketing System - Agent Menu
=====
1) Add a new account
2) Modify an existing account
3) Remove an account
4) List accounts: detailed view
-----
0) Logout

Selection:
```

- This is the main menu for an agent who has authorization to manage the accounts for the system.
- Option 1: New accounts can be added to the array.
 - Your logic should include finding an available index in the accounts array where a new Account record can be added (**the account number at that index should be zero (0) which indicates an empty record**).
 - If there are no remaining elements available for a new record (all account numbers have values > 0), then an error message should be displayed:


```
ERROR: Account listing is FULL, call ITS Support!
```
 - Your logic should at some point be calling the function "**getAccount**"
- Option 2: Modification to existing accounts.
 - Your logic should prompt the user for an account number and if not found in the array of accounts, display an error (see sample output for appropriate message)

- If the account is found, then at some point you must call the "**updateAccount**" function.
- Option 3: Removal of an account.
 - Like option 2 above, you must prompt for an account number used to lookup the specific account record and display an appropriate error message if not found.
 - If the account number entered is the same as the **logged-in user's account**, you must deny the removal (see sample output for appropriate message)
 - If the account is found and is different from the logged-in user's account, then you must obtain confirmation for the removal (see sample output for prompt message where only an uppercase "Y" or "N" is permitted)
 - If the user confirms the removal, you must set the **account number member to a zero (0) value** – no other members should be modified.
 - Based on the confirmation of removal response, display the appropriate message (see sample output for possible messages)
- Option 4: Displays a tabular view of all the accounts in a "detailed" format.
 - Your code should at some point call the function "**displayAllAccountDetailRecords**"
- Option – 0: Should exit the menu function (essentially logging out of the system) and return to the login menu (Note: You should **NOT** be calling the menuLogin function to achieve this).

-
- Function: **findAccountIndexByAcctNum**
 - Receives an argument in **parameter 1**, an integer value representing the account number to find a match on in the account array (parameter 2)
 - Receives as an argument in **parameter 2**, an unmodifiable array of type **Account**.
 - Receives as an argument in **parameter 3**, an integer specifying the maximum number of elements in the second argument's array.
 - Receives as an argument in **parameter 4**, an integer type representing a zero or non-zero value (indicates if this function should prompt the user for the account number)
 - Returns an integer representing the array index of the **Account** that matches the desired account number or -1 if the record was not found.

Functionality

- This function's purpose is to search the array received in argument 2 for a match on a desired account number.
 - HINT**: This function can be used in many places in your application
- If the 4th argument is a zero (0) value, then the 1st argument account number value should be used in the search routine, otherwise, the function will need to first prompt the user for the invoice number to search on (and will NOT use the 1st argument value).
- If prompting is required (4th argument will be a non-zero value), use the following prompt message:

Enter the account#:

 - Use the entered account number value in the search routine.
- The function should return either -1 (no record match found) or the index position where the matched record was found within the argument 2 array.

- Function: **displayAllAccountDetailRecords**
 - Receives as an argument in **parameter 1**, an unmodifiable array of type **Account**.
 - Receives as an argument in **parameter 2**, an integer specifying the maximum number of elements in the first argument's array.
 - Does not return anything.

Functionality

- This function displays a detailed view of **all** the **valid** account records (where the account number is greater than zero (0)).
- The appropriate tabular header should be displayed (call the appropriate function)
- The corresponding record detail should be displayed (call the appropriate function) as many times as required to show all the valid records.

-
- Function: **pauseExecution** (**Provided for you – see below**)
 - This function is provided for you (the logic and a sample of this is already made available to you in the course notes)
 - Often, we want to be able to "pause" the application and await the user's confirmation to continue by pressing the "enter" key. Be sure to call this function where appropriate (see sample output for where this needs to be implemented)

Function Prototype

```
// Pause execution until user enters the enter key
void pauseExecution(void)
```

Function Definition

```
// Pause execution until user enters the enter key
void pauseExecution(void)
{
    printf("<< ENTER key to Continue... >>");
    clearStandardInputBuffer();
    putchar('\n');
}
```

Account Module

3. The existing function "getAccount" will require some enhancements to the logic and will only affect the source definition file ("**account.c**").

-
- Function: **getAccount**
 - Modify the logic for this function so that it will prompt the user for the **User Login** information ONLY IF the new account is an "**Agent**" type
 - If the new account is not an "Agent" type, **DO NOT** prompt the user for the **User Login** information (only agents may login to the system). In such cases, set the **User Login** member variable to a **safe empty state**

4. Each function briefly described below will require a new function prototype to be placed in the "**account.h**" file, and the respective function definitions in the "**account.c**" source file.

- Function: **updateAccount**

- Receives a modifiable **Account** pointer argument.
- Does not **return** anything but does return data for an **Account** type via the argument pointer variable.

Functionality

- Display's the update menu options for the account until the user wishes to exit.
- Display's a menu header title that **includes the account number** being modified (with 40 dashed characters on the next line). Example:

```
Update Account: 00001 ( ? )
-----
```

- The highlighted green **?** should be replaced with the **full name** associated to the desired account
- Followed by a menu with options to modifying specific members of an **Account**:

```
1) Update account type (current value: ?)
2) Person
3) Login
0) Done
Selection:
```

- The highlighted green **?** should be replaced with the **account type** member (single character **C** or **A**) associated to the desired account
- Note: The account number member is not modifiable and therefore is not an option in this menu
- Option – 1: Prompt the user for the modified value and assign the entered value to the account accordingly (see sample output for prompt message and valid values)

Note:

- If the modified account type is being set to an "**Agent**" type, the user must immediately be prompted to enter the User Login information lead by the message:

```
Agent type accounts require a user login. Please enter this information now:
```

- Otherwise, if being changed to a "**Customer**" type, set the User Login information to a **safe empty state** since this information is not required.

- Option – 2: Should call at some point the function "**updatePerson**"

- Option – 3:

- This option should display an error message if the account is a "**Customer**" type since non-Agent accounts will not have login information (and return to the menu):

```
ERROR: Customer account types don't have user logins!
```

- Otherwise, at some point call the function "**updateUserLogin**"

- Option – 0: Should exit the function and return to the caller (the agent menu; Note: You should **NOT** be calling the menuAgent function to achieve this)

- Function: **updatePerson**
 - Receives a modifiable **Person** pointer argument.
 - Does not **return** anything but does return data for a **Person** type via the argument pointer variable.

Functionality

- Display's a menu header title followed by 40 dashed characters on the next line, followed by a menu with options to modifying specific members of a **Person**:

```
Person Update Options
```

```
-----
```

```
1) Full name (current value: ?)
2) Household Income (current value: $?)
3) Country (current value: ?)
0) Done
```

```
Selection:
```

- The highlighted green **?** should be replaced with the current value for the respective members
- Note: The account holders birth year member is not modifiable and therefore is not an option in this menu
- Option – 1: Prompt the user for the modified value and assign the entered value to the appropriate **Person** member accordingly (see sample output for prompt message)
- Option – 2: Prompt the user for the modified value and assign the entered value to the appropriate **Person** member accordingly (see sample output for prompt message)
- Option – 3: Prompt the user for the modified value and assign the entered value to the appropriate **Person** member accordingly (see sample output for prompt)
- Option – 0: Should exit the function and return to the caller (the update account menu; Note: You should **NOT** be calling the updateAccount function to achieve this).

- Function: **updateUserLogin**
 - Receives a modifiable **UserLogin** pointer argument.
 - Does not **return** anything but does return data for an **UserLogin** type via the argument pointer variable.

Functionality

- Display's the update menu options for the user login until the user wishes to exit.
- Display's a menu header title that **includes the login name** being modified (with 40 dashed characters on the next line). Example:

```
User Login: ? - Update Options
```

```
-----
```

- The highlighted green **?** should be replaced with the **Login Name** associated to the desired account's user login.
- Followed by a menu with options to modifying specific members of a **UserLogin**:


```
1) Password
0) Done
Selection:
```
- Note: The login name member is not modifiable and therefore is not an option in this menu

- Option – 1: Prompt the user for the modified value and assign the entered value to the appropriate **UserLogin** member accordingly (see sample output for prompt message)
- Option – 0: Should exit the function and return to the caller (the update account menu;
Note: You should **NOT** be calling the updateAccount function to achieve this).

A1-MS4: Sample Output

```

=====
Account Ticketing System - Login
=====
1) Login to the system
0) Exit application
-----

Selection: 2
ERROR: Value must be between 0 and 1 inclusive: -1
ERROR: Value must be between 0 and 1 inclusive: login
ERROR: Value must be an integer: 0

Are you sure you want to exit? ([Y]es|[N]o): no
ERROR: Character must be one of [yYnN]: n

=====
Account Ticketing System - Login
=====
1) Login to the system
0) Exit application
-----

Selection: 1

Enter your account#: 12345
ERROR: Login failed!

<< ENTER key to Continue... >> [ENTER]

=====
Account Ticketing System - Login
=====
1) Login to the system
0) Exit application
-----

Selection: 1

Enter your account#: 50001
ERROR: Login failed!

<< ENTER key to Continue... >> [ENTER]

=====
Account Ticketing System - Login
=====

```

```
1) Login to the system
0) Exit application
-----

Selection: 1

Enter your account#: 50008

AGENT: Will Smith (50008)
=====
Account Ticketing System - Agent Menu
=====
1) Add a new account
2) Modify an existing account
3) Remove an account
4) List accounts: detailed view
-----
0) Logout

Selection: 1

Account Data: New Record
-----
Enter the account number: 91111

Enter the account type (A=Agent | C=Customer): Agent
ERROR: Character must be one of [AC]: a
ERROR: Character must be one of [AC]: c
ERROR: Character must be one of [AC]: C

Person Data Input
-----
Enter the person's full name (30 chars max): Tesla Tommy
Enter birth year (current age must be between 18 and 110): 2004
ERROR: Value must be between 1911 and 2003 inclusive: 1910
ERROR: Value must be between 1911 and 2003 inclusive: 2003
Enter the household Income: $0.00
ERROR: Value must be a positive double floating-point number: 1.99
Enter the country (30 chars max.): FINLAND

*** New account added! ***

<< ENTER key to Continue... >> [ENTER]

AGENT: Will Smith (50008)
=====
Account Ticketing System - Agent Menu
=====
1) Add a new account
2) Modify an existing account
3) Remove an account
4) List accounts: detailed view
-----
0) Logout

Selection: 1

Account Data: New Record
```

 Enter the account number: **92222**

Enter the account type (A=Agent | C=Customer): **A**

Person Data Input

Enter the person's full name (30 chars max): **Tania Ticket**

Enter birth year (current age must be between 18 and 110): **2001**

Enter the household Income: \$**88123.45**

Enter the country (30 chars max.): **IRELAND**

User Login Data Input

Enter user login (10 chars max): **agent123**

Enter the password (must be 8 chars in length): **diamonds**

*** New account added! ***

<< ENTER key to Continue... >> **[ENTER]**

AGENT: Will Smith (50008)

=====

Account Ticketing System - Agent Menu

=====

- 1) Add a new account
 - 2) Modify an existing account
 - 3) Remove an account
 - 4) List accounts: detailed view
-

0) Logout

Selection: **4**

Acct#	Acct.Type	Full Name	Birth	Income	Country	Login	Password
50001	CUSTOMER	Silly Sally	1990	150000.10	CANADA		
50028	AGENT	Fred Flintstone	1972	2250400.22	AFRICA	Bedrock-10	y*b*d*b*
50004	CUSTOMER	Betty Boop	1978	250800.74	INDIA		
50008	AGENT	Will Smith	1952	2350600.82	U.S.A.	MIBAgent-J	t*e*o*s*
50020	CUSTOMER	Shrimpy Shrimp	2000	350500.35	KOREA		
91111	CUSTOMER	Tesla Tommy	2003	1.99	FINLAND		
92222	AGENT	Tania Ticket	2001	88123.45	IRELAND	agent123	d*a*o*d*

<< ENTER key to Continue... >> **[ENTER]**

AGENT: Will Smith (50008)

=====

Account Ticketing System - Agent Menu

=====

- 1) Add a new account
 - 2) Modify an existing account
 - 3) Remove an account
 - 4) List accounts: detailed view
-

0) Logout

Selection: **2**

Enter the account#: **91111**

Update Account: 91111 (Tesla Tommy)

1) Update account type (current value: C)
2) Person
3) Login
0) Done
Selection: **3**

ERROR: Customer account types don't have user logins!

Update Account: 91111 (Tesla Tommy)

1) Update account type (current value: C)
2) Person
3) Login
0) Done
Selection: **1**

Enter the account type (A=Agent | C=Customer): **A**

Agent type accounts require a user login. Please enter this information now:

User Login Data Input

Enter user login (10 chars max): **Bond James Bond**
ERROR: String length must be no more than 10 chars: **Agent-Bond**
Enter the password (must be 8 chars in length): **qweapons**

Update Account: 91111 (Tesla Tommy)

1) Update account type (current value: A)
2) Person
3) Login
0) Done
Selection: **2**

Person Update Options

1) Full name (current value: Tesla Tommy)
2) Household Income (current value: \$1.99)
3) Country (current value: FINLAND)
0) Done
Selection: **1**

Enter the person's full name (30 chars max): **James Bond**

Person Update Options

1) Full name (current value: James Bond)
2) Household Income (current value: \$1.99)
3) Country (current value: FINLAND)
0) Done
Selection: **2**

Enter the household Income: **\$2123456.75**

Person Update Options

```
-----
1) Full name (current value: James Bond)
2) Household Income (current value: $2123456.75)
3) Country (current value: FINLAND)
0) Done
Selection: 3
```

Enter the country (30 chars max.): ENGLAND

Person Update Options

```
-----
1) Full name (current value: James Bond)
2) Household Income (current value: $2123456.75)
3) Country (current value: ENGLAND)
0) Done
Selection: 0
```

Update Account: 91111 (James Bond)

```
-----
1) Update account type (current value: A)
2) Person
3) Login
0) Done
Selection: 3
```

User Login: Agent-Bond - Update Options

```
-----
1) Password
0) Done
Selection: 1
```

Enter the password (must be 8 chars in length): spygames

User Login: Agent-Bond - Update Options

```
-----
1) Password
0) Done
Selection: 0
```

Update Account: 91111 (James Bond)

```
-----
1) Update account type (current value: A)
2) Person
3) Login
0) Done
Selection: 4
ERROR: Value must be between 0 and 3 inclusive: 0
```

AGENT: Will Smith (50008)

Account Ticketing System - Agent Menu

```
=====
1) Add a new account
2) Modify an existing account
3) Remove an account
4) List accounts: detailed view
```

0) Logout

Selection: **4**

Acct#	Acct.Type	Full Name	Birth	Income	Country	Login	Password
50001	CUSTOMER	Silly Sally	1990	150000.10	CANADA		
50028	AGENT	Fred Flintstone	1972	2250400.22	AFRICA	Bedrock-10	y*b*d*b*
50004	CUSTOMER	Betty Boop	1978	250800.74	INDIA		
50008	AGENT	Will Smith	1952	2350600.82	U.S.A.	MIBAgent-J	t*e*o*s*
50020	CUSTOMER	Shrimpy Shrimp	2000	350500.35	KOREA		
91111	AGENT	James Bond	2003	2123456.75	ENGLAND	Agent-Bond	s*y*a*e*
92222	AGENT	Tania Ticket	2001	88123.45	IRELAND	agent123	d*a*o*d*

<< ENTER key to Continue... >> **[ENTER]**

AGENT: Will Smith (50008)

=====
Account Ticketing System - Agent Menu
=====

- 1) Add a new account
- 2) Modify an existing account
- 3) Remove an account
- 4) List accounts: detailed view

0) Logout

Selection: **2**

Enter the account#: **92222**

Update Account: 92222 (Tania Ticket)

-
- 1) Update account type (current value: A)
 - 2) Person
 - 3) Login
 - 0) Done

Selection: **1**

Enter the account type (A=Agent | C=Customer): **C**

Update Account: 92222 (Tania Ticket)

-
- 1) Update account type (current value: C)
 - 2) Person
 - 3) Login
 - 0) Done

Selection: **0**

AGENT: Will Smith (50008)

=====
Account Ticketing System - Agent Menu
=====

- 1) Add a new account
- 2) Modify an existing account
- 3) Remove an account
- 4) List accounts: detailed view

 0) Logout

Selection: **4**

Acct#	Acct.Type	Full Name	Birth	Income	Country	Login	Password
50001	CUSTOMER	Silly Sally	1990	150000.10	CANADA		
50028	AGENT	Fred Flintstone	1972	2250400.22	AFRICA	Bedrock-10	y*b*d*b*
50004	CUSTOMER	Betty Boop	1978	250800.74	INDIA		
50008	AGENT	Will Smith	1952	2350600.82	U.S.A.	MIBAgent-J	t*e*o*s*
50020	CUSTOMER	Shrimpy Shrimp	2000	350500.35	KOREA		
91111	AGENT	James Bond	2003	2123456.75	ENGLAND	Agent-Bond	s*y*a*e*
92222	CUSTOMER	Tania Ticket	2001	88123.45	IRELAND		

<< ENTER key to Continue... >> **[ENTER]**

AGENT: Will Smith (50008)

=====
 Account Ticketing System - Agent Menu
 =====

- 1) Add a new account
- 2) Modify an existing account
- 3) Remove an account
- 4) List accounts: detailed view

 0) Logout

Selection: **3**

Enter the account#: **50008**

ERROR: You can't remove your own account!

<< ENTER key to Continue... >> **[ENTER]**

AGENT: Will Smith (50008)

=====
 Account Ticketing System - Agent Menu
 =====

- 1) Add a new account
- 2) Modify an existing account
- 3) Remove an account
- 4) List accounts: detailed view

 0) Logout

Selection: **3**

Enter the account#: **50001**

Acct#	Acct.Type	Full Name	Birth	Income	Country	Login	Password
50001	CUSTOMER	Silly Sally	1990	150000.10	CANADA		

Are you sure you want to remove this record? ([Y]es|[N]o): **Y**

*** Account Removed! ***


```
<< ENTER key to Continue... >> [ENTER]

AGENT: Will Smith (50008)
=====
Account Ticketing System - Agent Menu
=====
1) Add a new account
2) Modify an existing account
3) Remove an account
4) List accounts: detailed view
-----
0) Logout

Selection: 4

Acct# Acct.Type Full Name      Birth Income      Country  Login      Password
-----
50028 AGENT      Fred Flintstone  1972  2250400.22 AFRICA   Bedrock-10 y*b*d*b*
50004 CUSTOMER   Betty Boop      1978   250800.74 INDIA
50008 AGENT      Will Smith      1952  2350600.82 U.S.A.   MIBAgent-J t*e*o*s*
50020 CUSTOMER   Shrimpy Shrimp  2000   350500.35 KOREA
91111 AGENT      James Bond      2003  2123456.75 ENGLAND  Agent-Bond s*y*a*e*
92222 CUSTOMER   Tania Ticket    2001   88123.45 IRELAND

<< ENTER key to Continue... >> [ENTER]

AGENT: Will Smith (50008)
=====
Account Ticketing System - Agent Menu
=====
1) Add a new account
2) Modify an existing account
3) Remove an account
4) List accounts: detailed view
-----
0) Logout

Selection: 0

### LOGGED OUT ###

=====
Account Ticketing System - Login
=====
1) Login to the system
0) Exit application
-----

Selection: 0

Are you sure you want to exit? ([Y]es|[N]o): Y

=====
Account Ticketing System - Terminated
=====
```

Reflection (Worth 20%, Due Date: November 12th)

Academic Integrity

It is a violation of academic policy to copy content from the course notes or any other published source (including websites, work from another student, or sharing your work with others).

Failure to adhere to this policy will result in the filing of a violation report to the Academic Integrity Committee.

Instructions

- Create a text file named “**reflect.txt**” and record your answers to the below questions in this file.
 - Answer each question in sentence/paragraph form unless otherwise instructed.
 - A minimum **300** overall word count is required (does NOT include the question).
 - Whenever possible, be sure to substantiate your answers with a brief example to demonstrate your view(s).
1. As painful and frustrating as it may be to match the expected output exactly, explain why you think you must be put through this challenge and expected to meet this minimum expectation.
 2. What factors must you consider when naming a module or library? Why do you think it is a suggested best practice to identify a module/library header and source code files using the same name? Give an example to support your argument.
 3. This application applies a cascading menu system framework. What does this mean and explain it using elements from this application to support your understanding of this logical concept?

Reflections will be graded based on the published rubric:

<https://github.com/Seneca-144100/IPC-Project/tree/master/Reflection%20Rubric.pdf>

Milestone – 4 Submission

1. Upload (file transfer) your all header and source files including your reflection:
 - **commonHelpers.h & commonHelpers.c**
 - **account.h & account.c**
 - **accountTicketingUI.h & accountTicketing.c**
 - **a1ms4.c**
 - **reflect.txt**
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 a1ms4.c commonHelpers.c account.c accountTicketingUI.c -o ms4 <ENTER>
```

*If there are no error/warnings are generated, execute it: **ms4 <ENTER>***
4. Run the submission command below (replace **profname.proflastname** with **your professors** Seneca userid and replace **NAA** with your section):

```
~profName.proflastname/submit 144a1ms4/NAA_ms4 <ENTER>
```
5. Follow the on-screen submission instructions.