Fall 2020

Contacts

Assignment 2

Assignment 2 uses assignment 1 and focuses on a "Contact Management System". This assignment continues to emphasize modularization by compartmentalizing highly cohesive and tightly coupled tasks into modules (*.h and *.c files). This strategy will reduce redundant coding throughout all parts of the application.

Deadlines

This assignment is broken into two (2) parts:

Milestone	Due Date
1-2	Fri. Nov 20 by 23:59 PM (EST)
3-4	Tue. Dec 1 by 23:59 PM (EST) Extension: Fri Dec 4 23:59

Milestone 1 (2%)

(Due with your milestone 2 submission)

Download or clone Assignment 2 (**A2**) from https://github.com/Seneca-144100/IPC-Project

Open the project file **A2MS1** and look inside (expand "Header Files" and "Source Files" folders – see figure 2.1.1). You will find two modules:

Module	Header File	Source File
Contacts	contacts.h	contacts.c
Contact Helpers	contactHelpers.h	contactHelpers.c

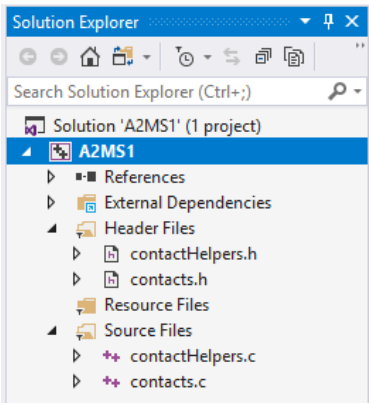


Figure: 2.1.1 – Visual Studio Project Contents

Contacts Header File (.h)

Structures

Open the **contacts.h** file and copy the structures (Name, Address, Numbers, and Contact) from the file **contacts.h** in *Assignment 1*. Be careful not to paste over the helpful comments in the provided **contacts.h** file!

Function Prototypes

1. Copy the function prototypes (**getName**, **getAddress**, and **getNumbers**) from the file **contacts.h** in *Assignment 1 (Milestone 4)*. Again, be careful not to paste over the helpful comments in the provided **contacts.h** file!

2. Declare an additional function prototype:

```
void getContact(struct Contact *contact);
```

   - This function has one parameter that receives a pointer to a Contact.
   - The details on how this function should work is described in Milestone 2.

Contacts Source File (.c)

Open the **contacts.c** source code file.

Libraries

In order to define the functions that were declared as prototypes in the **contacts.h** file, this source file needs visibility of their existence. This is accomplished by including the required header library file. Include the **contacts.h** header file (see source code comments).

Function Definitions

1. Copy the function definitions (**getName**, **getAddress**, and **getNumbers**) from the file **contacts.c** in *Assignment 1 (Milestone 4)*.

2. Add an **empty** definition for the new function **getContact** (see the prototype declared in the **contacts.h** file) – refer to the source comments for placement. For this milestone you don't have to define the details for this function (this will be done in Milestone 2) so for now, specify an empty code block:

```
getContact function header goes here…
{
        // Use an open and close curly brace with a blank line
```

```
}
```

Contact Helper Header File (.h)

The contact helper module contains a group of common functions for use in other source code modules in the application. Other modules will be able to access these functions (without having to recode them) by including this header file.

Function Prototypes

Open the **contactHelpers.h** file. To help you get started, you will notice the prototype for the clearKeyboard function is already done for you.

```
void clearKeyboard(void);
```

- **clearKeyboard** does not return a value and has no parameters.
- The details on how this function should work is described in Milestone 2.

Add the following additional helper function prototypes:

```
void pause(void);
```

- **pause** does not return a value and has no parameters.
- The details on how this function should work is described in Milestone 2.

```
int getInt(void);
```

- **getInt** returns an integer value and has no parameters.
- The details on how this function should work is described in Milestone 2.

```
int getIntInRange(int min, int max);
```

- **getIntInRange** returns an integer value and receives two (2) integer parameters.
- The details on how this function should work is described in Milestone 2.

```
int yes(void);
```

- **yes** returns an integer value and has no parameters.
- The details on how this function should work is described in Milestone 2.

```
int menu(void);
```

- **menu** returns an integer value and has no parameters.
- The details on how this function should work is described in Milestone 2.

```
void contactManagerSystem(void);
```

- **contactManagerSystem** does not return a value and has no parameters.
- The details on how this function should work is described in Milestone 2.

Contact Helper Source File (.c)

Open the **contactHelpers.c** source code file.

Libraries

Just as the **contacts.c** source code file included the **contacts.h** header file, the **contactHelpers.c** source code file should include the **contactHelpers.h** header file. Including the header file exposes the prototyped functions before they are defined (see source code comments).

Function Definitions

Add an empty definition for each function prototyped in the **contactHelpers.h**. You will notice the clearKeyboard function has been done for you. Follow this example for the remaining functions (refer to the source code comments for placement).

For this milestone, you don't have to define the details for the functions (the details are described in Milestone 2).

Milestone 1 Submission

Milestone 1 does not need to be submitted on matrix. It will be graded along with Milestone 2 when it is submitted.

Milestone 2 (38%)

(Due November 20, by 23:59 EST)

Open the project file **A2MS2** and look inside (expand "Header Files" and "Source Files" folders – see figure 2.2.1). You will notice an additional source code file **a2ms2.c** (do not modify this file). This is the main file used to assess your functions to determine if they work to this milestone's specifications.

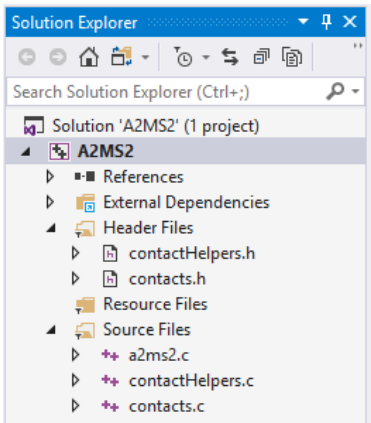


Figure: 2.2.1 – Visual Studio Project Contents

Header Files

contact.h and contactHelpers.h

There are no changes required to these files for this milestone. Consult the comments provided in the header files (.h) for these modules and copy the appropriate contents from Milestone 1.

Contact Helper Source File (.c)

1. Open the **contactHelpers.c** source code file.

   Reminder: Be sure to include the contactHelpers.h header file!

   To help you get started, notice that the clearKeyboard function is done for you.

2. For the remaining functions, copy the empty functions from the **contactHelpers.c** file in *Milestone 1* (consult the source code comments for placement – be careful not to replace the clearKeyboard function that is already provided for you).

3. Complete the full function definition for each function using the descriptions below:

```
void clearKeyboard(void);
```

- **clearKeyboard** does not return a value and has no parameters.
- This function makes sure the keyboard input buffer is clear of any residual character by reading from the input buffer character by character until it reads a new line character.
- This function is provided for you – please consult the IPC144 course notes in the section "Input Functions" to learn about the standard input output library (stdio) getchar() function.

```
void pause(void);
```

- **pause** does not return a value and has no parameters.
- This function pauses the execution of the application by displaying a message and waiting for the user to press the <ENTER> key.
- Display the following line and DO NOT include a newline character:

  >(Press Enter to continue)<

- After displaying the above message, call the clearKeyboard function.

  Note: The clearKeyboard function is used for a foolproof <ENTER> key entry

```
int getInt(void);
```

- **getInt** returns an integer value and has no parameters.
- This function gets a valid integer from the keyboard and returns it. If the value entered is not a valid integer an error message should be displayed:

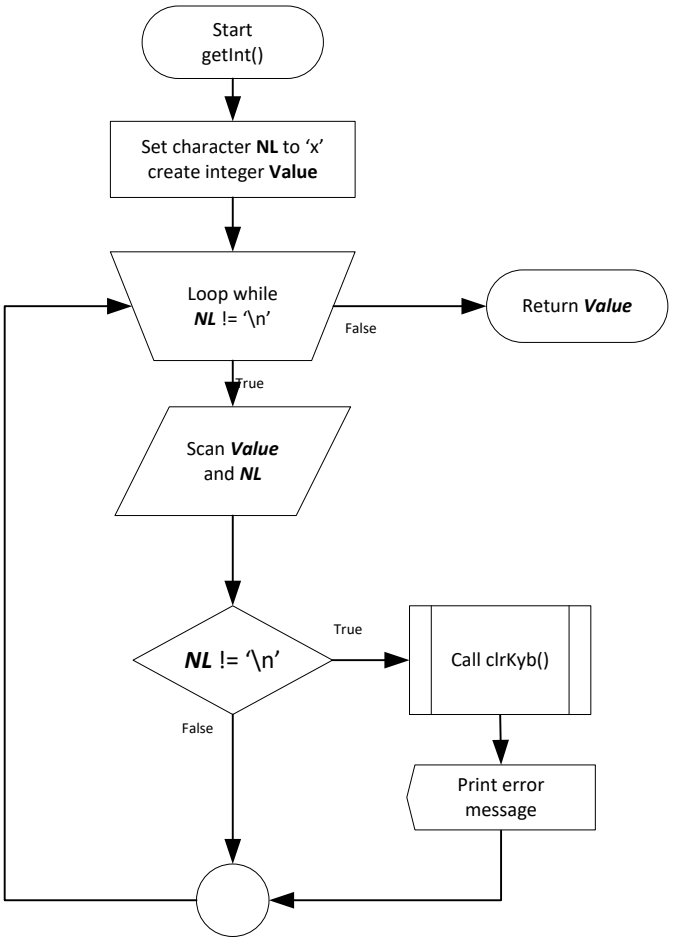
  >*** INVALID INTEGER *** <Please enter an integer>: <

- This function should continue to prompt the user for a valid integer.
- This function must be foolproof and 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 clearKeyboard function, followed by displaying the error message described above and continue to prompt for a valid integer. Review the following flowchart that describes this process:



```
int getIntInRange(int min, int max);
```

- **getIntInRange** returns an integer value and receives two (2) integer parameters.
- This function uses getInt to receive a valid integer and returns it only if the value entered is within the lower-bound (first parameter) and upper-bound (second parameter) range (**inclusive**).
- If the integer entered is not within the valid range, the following error message should be displayed:

  >*** OUT OF RANGE *** <Enter a number between [param-1] and [param-2]>: <

  Note: *Substitute the "[param-1]" and "[param-2]" with the function parameter values*

- This function should be a foolproof call to ensure an integer is entered within a given range. Therefore, it should continue to prompt the user for an entry until a valid integer is within the specified range.

```
int yes(void);
```

- **yes** returns an integer value and has no parameters.
- This function prompts the user for a **single character** entry
- The character entered is only valid if it is a "Y", "y", "N", or "n". Any other value entered (including more than one character) is an error and should display the following message:

  >*** INVALID ENTRY *** <Only (Y)es or (N)o are acceptable>: <

  Hint
  This function is very similar to the getInt function only this function looks for a character as the first value. If the second value is not a newline character it is definitely an error as it won't be a single character entry (be sure to call the clearKeyboard function to clear the input buffer). If the second value is a newline character, then assess if the first value is one of the four (4) valid characters. If it isn't one of the valid four (4) characters, then it is an error.

- This function should continue to prompt the user for an entry until one of the four (4) valid characters is entered.
- When a valid value is entered, the function should return an integer value of 1 when the value is "Y" or "y". Otherwise it should return 0.

```
int menu(void);
```

- **menu** returns an integer value and has no parameters.
- This function should display the following menu:

```
>Contact Management System<
>-------------------------<  (there are 25 dashes)
>1. Display contacts<
>2. Add a contact<
>3. Update a contact<
>4. Delete a contact<
>5. Search contacts by cell phone number<
>6. Sort contacts by cell phone number<
>0. Exit<
><
>Select an option:> <
```

- Prompt for user entry an integer value between the values 0-6 inclusive
- Any other value entered or an integer that is not within the 0-6 range is an error
- The function should continue to prompt for an integer value within the 0-6 range until a valid value is entered.
- When a valid integer value is entered, this value is returned.

Hint

This logic sounds familiar… perhaps there is already a function that can help you get an integer value within a specified range…

```
void contactManagerSystem(void);
```

- **contactManagerSystem** does not return a value and has no parameters.
- This function is the heart of the application and will run the whole program.
- This function starts by showing the menu for the system and receives the user's selection
- If the user enters 1, it displays:
  `><<< Feature 1 is unavailable >>><` *(followed by two (2) newlines)*
  If the user enters 2, it displays:
  `><<< Feature 2 is unavailable >>><` *(followed by two (2) newlines)*
  If the user enters 3, it displays:
  `><<< Feature 3 is unavailable >>><` *(followed by two (2) newlines)*
  If the user enters 4, it displays:
  `><<< Feature 4 is unavailable >>><` *(followed by two (2) newlines)*
  If the user enters 5, it displays:
  `><<< Feature 5 is unavailable >>><` *(followed by two (2) newlines)*
  If the user enters 6, it displays:
  `><<< Feature 6 is unavailable >>><` *(followed by two (2) newlines)*
  For selections between 1 and 6, the application should **pause** and then return to displaying the menu.
- If the user enters 0, prompt to exit the application. Display the following:
  `>Exit the program? (Y)es/(N)o: <`

Wait for the user to enter “Y”, ”y”, “N” or “n” (for Yes or No).
If the user replies Yes (“Y”,”y”), it will end the program displaying the following message:

>`Contact Management System: terminated`**<** *(followed by a newline)*

Otherwise, if the user entered No (“N”,”n”), the application continues to display the menu.

- The following is a general pseudo code for a menu driven user interface. Using this pseudo code is optional. You can use any other logic if you like.

```
Menu driver pseudo code:
while it is not done
   display menu
   get selected option from user
   check selection:
      option one selected
         act accordingly
      end option one
      option two selected
         act accordingly
      end option two
      .
      .
      Exit is selected
         program is done
      end exit
   end check
end while
```

Contacts Source File (.c)

Open the **contacts.c** source code file.

1. Open the **contacts.c** source code file.

   Reminder: Be sure to include the **contacts.h** header file!

   The contact helpers module contains additional functions that can be used to streamline some functions previously coded for getting the Name, Address, and Numbers parts of a Contact. To make these helper functions available for use in this source file include the **contactHelpers.h** header file (see the source code comments for placement).

2. Copy the functions from the **contacts.c** file in *Milestone 1* (see the source code comments for placement – be careful not to replace any additional helpful comments)

3. Update the functions getName, getAddress, and getNumbers to use any of the new functions created in the contactHelpers.h library (wherever applicable). See source code comments for some suggestions.

4. Update function getNumbers so that the cell phone number entry is **mandatory** (don't ask if the user wants to enter this value)

5. Define the new function prototyped in *Milestone 1* getContact using the following description:

   ```
   void getContact(struct Contact *contact);
   ```

   - This function does not return a value but has one parameter that receives a pointer to a Contact.
   - The purpose of this function is to set the values for a Contact using the pointer parameter variable (set the Contact it points to).
   - Use the pointer parameter received to this function to supply the appropriate Contact member to the "get" functions (getName, getAddress, and getNumbers) to set the values for the Contact.

Sample Output

Below is a sample output. User input values are identified in **BOLD**. Your output should match exactly:

```
----------------------------------------
Testing: Yes()
----------------------------------------
Please enter 'Y' > Y
    Result: 1
Please enter 'y' > y
    Result: 1
Please enter 'N' > N
    Result: 0
Please enter 'yes', then 'no', then 'n' > yes
*** INVALID ENTRY *** <Only (Y)es or (N)o are acceptable>: no
*** INVALID ENTRY *** <Only (Y)es or (N)o are acceptable>: n
    Result: 0

----------------------------------------
Testing: pause()
----------------------------------------
(Press Enter to Continue)     <Enter>

----------------------------------------
Testing: getInt()
----------------------------------------
Enter 'ipc', then '144' > ipc
```

```
*** INVALID INTEGER *** <Please enter an integer>: 144
    Integer entered: 144

----------------------------------------
Testing: getIntInRange(int,int)
----------------------------------------
Enter 'seneca', then '99', then '501', then '250' > seneca
*** INVALID INTEGER *** <Please enter an integer>: 99
*** OUT OF RANGE *** <Enter a number between 100 and 500>: 501
*** OUT OF RANGE *** <Enter a number between 100 and 500>: 250
    Integer entered: 250

Enter '100' > 100
    Integer entered: 100
Enter '500' > 500
    Integer entered: 500

----------------------------------------
Testing: getContact(struct Contact *)
----------------------------------------
Please enter the contact's first name: Tom See John
Do you want to enter a middle initial(s)? (y or n): yes
*** INVALID ENTRY *** <Only (Y)es or (N)o are acceptable>: y
Please enter the contact's middle initial(s): How Wong R. U.
Please enter the contact's last name: Song Sing
Please enter the contact's street number: one two
*** INVALID INTEGER *** <Please enter an integer>: -99
*** INVALID STREET NUMBER *** <must be a positive number>: 99
Please enter the contact's street name: Keele Street
Do you want to enter an apartment number? (y or n): y
Please enter the contact's apartment number: -1920
*** INVALID APARTMENT NUMBER *** <must be a positive number>: 1920
Please enter the contact's postal code: A8A 3J3 R1W
Please enter the contact's city: North Bay
Please enter the contact's cell phone number: 9051116666
Do you want to enter a home phone number? (y or n): n y
*** INVALID ENTRY *** <Only (Y)es or (N)o are acceptable>: n
Do you want to enter a business phone number? (y or n): n

Values Entered:
Name: Tom See John How Wo Song Sing
Address: 99|Keele Street|1920|A8A 3J3|North Bay
Numbers: 9051116666||

----------------------------------------
Testing: contactManagerSystem()
----------------------------------------
Contact Management System
-------------------------
1. Display contacts
2. Add a contact
```

```
3. Update a contact
4. Delete a contact
5. Search contacts by cell phone number
6. Sort contacts by cell phone number
0. Exit

Select an option:> 9
*** OUT OF RANGE *** <Enter a number between 0 and 6>: 1

<<< Feature 1 is unavailable >>>

(Press Enter to Continue)     <Enter>

Contact Management System
-------------------------
1. Display contacts
2. Add a contact
3. Update a contact
4. Delete a contact
5. Search contacts by cell phone number
6. Sort contacts by cell phone number
0. Exit

Select an option:> 4

<<< Feature 4 is unavailable >>>

(Press Enter to Continue)     <Enter>

Contact Management System
-------------------------
1. Display contacts
2. Add a contact
3. Update a contact
4. Delete a contact
5. Search contacts by cell phone number
6. Sort contacts by cell phone number
0. Exit

Select an option:> 6

<<< Feature 6 is unavailable >>>

(Press Enter to Continue)     <Enter>

Contact Management System
-------------------------
1. Display contacts
2. Add a contact
3. Update a contact
4. Delete a contact
```

```
5. Search contacts by cell phone number
6. Sort contacts by cell phone number
0. Exit

Select an option:> 0

Exit the program? (Y)es/(N)o: n

Contact Management System
-------------------------
1. Display contacts
2. Add a contact
3. Update a contact
4. Delete a contact
5. Search contacts by cell phone number
6. Sort contacts by cell phone number
0. Exit

Select an option:> 0

Exit the program? (Y)es/(N)o: y

Contact Management System: terminated

----------------------------------------
Testing: Assign#2 - MS #2 test completed
----------------------------------------
```

Milestone 2 Reflection (60%)

Please provide answers to the following in a text file named **reflect.txt.**

In three or more paragraphs and a **minimum of 150 words**, explain what you learned while doing these first two milestones. In **addition to what you learned**, **your reflection should also include the following**:

- An explanation of the term “function” and briefly discuss the need for functions in any language. Refer to functions from this project to support your explanation.
- An explanation why you think the "helper" functions are in a different module and why those functions were not included in the "contacts" module. Refer to functions from this project to support your explanation.

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

Example:
An example reflection answer for Workshop #2 is available demonstrating the minimum criteria:
https://github.com/Seneca-144100/IPC-Project/tree/master/Example%20Reflection-WS_2.pdf

Milestone 2 Submission

If not on matrix already, upload your **contacts.h**, **contacts.c**, **contactHelpers.h**, **contactHelpers.c**, **a2ms2.c**, and **reflect.txt** files to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms2 contacts.c contactHelpers.c a2ms2.c <ENTER>
```

This command will compile your code and name your executable "**ms2**". Execute **ms2** and make sure everything works properly.

Then run the following script from your account and follow the instructions (replace profname.proflastname with your professors Seneca userid and replace **NAA** with your section):

~profname.proflastname/submit 100a2ms2/NAA_ms2 <ENTER>

Please Note

- A successful submission does not guarantee full credit for this workshop.
- If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.