Fall 2018

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 four (4) parts:

Milestone	Due Date
1	**In-lab** the week it is assigned
2	4 days from the assigned lab class
3	**In-lab** the week it is assigned
4	4 days from the assigned lab class

Milestone 1 (10%)

(Due in your Lab Class the week it is assigned)

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

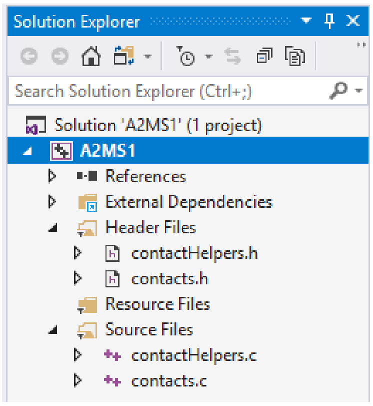


Figure: 2.1.1 – Visual Studio Project Contents

<u>Contacts Header File</u> (.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 *);
```

   - 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.

<u>Contacts Source File</u> (.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, int);
```

- **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).

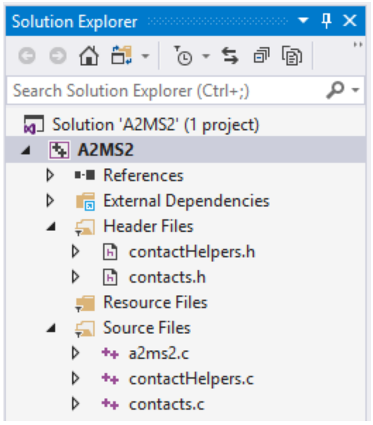
Milestone 1 Submission

Milestone 1 is to be done in the lab and shown to your instructor, there is no need to submit on matrix.

Milestone 2 (20%)

(Due Four (4) days from assigned date)

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.



```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'A2MS2' (1 project)
  A2MS2
    References
    External Dependencies
    Header Files
      contactHelpers.h
      contacts.h
    Resource Files
    Source Files
      a2ms2.c
      contactHelpers.c
      contacts.c
```

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);
```

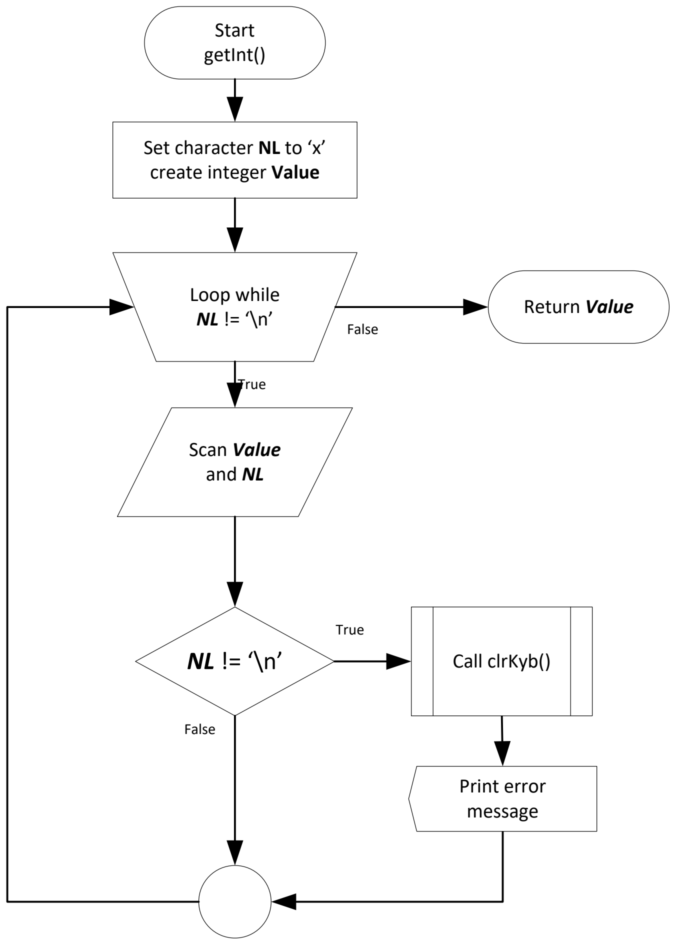
- **getInt** returns an integer value and has no parameters.
- This function gets a valid integer from the keyboard and returns it. If the value entered is not a valid integer an error message should be displayed:

  ```
  >*** 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, int);
```

- **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>: <
  ```

  <u>Hint</u>
  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 *);
   ```

   - 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 **RED**. 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 (20%)

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

1. In 3 or 4 sentences explain the term "function" and briefly discuss the need for functions in any language?
2. Briefly explain why you think the "helper" functions are in a different module and why those functions were not included in the "contacts" module?

Note: When completing the reflections it is a violation of academic policy to cut and paste content from the course notes or any other published source, or to copy the work of another

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.

Finally run the following script from your account (replace profname.proflastname with your professors Seneca userid):

```
~profname.proflastname/submit 144_a2ms2 <ENTER>
```

and follow the instructions.

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

Milestone 3 (10%)

(Due in your Lab Class the week it is assigned)

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

Open the project file **A2MS3** and look inside (expand "Header Files" and "Source Files" folders).

This milestone introduces the remaining function prototypes required to complete the contacts management system. You have three tasks to perform for this milestone:

1. Declare new function prototypes in **contactHelpers.h**
2. Define the new functions with empty code blocks in **contactHelpers.c**
3. Update **contacts.h** with safeguard and **contact.c** to use new function getTenDigitPhone

Contact Helper Header File (.h)

Open the **contactHelpers.h** file. Refer to the comments in the file and copy/paste your code from MS2 where directed.

You will need to include the **contacts.h** header file so the new functions (parameters) will be able to use the structures defined in that module (see source comments for placement).

Function Prototypes

You will notice there are two (2) prototypes already prepared:

```
void getTenDigitPhone(char[]);
```

- **getTenDigitPhone** does not return a value and expects a character array sized for 10 characters plus the null terminator.
- The next section describing the **contactHelpers.c** source file explains this function and its use.

```c
int findContactIndex(const struct Contact[], int, const char[]);
```

- **findContactIndex** returns an integer and expects a **Contact** array (marked constant so changes can't be made to it), an integer, and a character array.
- The details on how this function should work is described in Milestone 4.

Add the following additional function prototypes:

```c
void displayContactHeader(void);
```

- **displayContactHeader** does not return a value and has no parameters.
- The details on how this function should work is described in Milestone 4.

```c
void displayContactFooter(int);
```

- **displayContactFooter** does not return a value and receives one integer parameter.
- The details on how this function should work is described in Milestone 4.

```c
void displayContact(const struct Contact*);
```

- **displayContact** does not return a value and receives a constant **Contact** pointer (cannot be updated) parameter.
- The details on how this function should work is described in Milestone 4.

```c
void displayContacts(const struct Contact[], int);
```

- **displayContacts** does not return a value and receives a constant **Contact** array (cannot be updated), and an integer for parameters.
- The details on how this function should work is described in Milestone 4.

```c
void searchContacts(const struct Contact[], int);
```

- **searchContacts** does not return a value and receives a constant **Contact** array (cannot be updated), and an integer for parameters.
- The details on how this function should work is described in Milestone 4.

```c
void addContact(struct Contact[], int);
```

- **addContact** does not return a value and receives a **Contact** array, and an integer for parameters.
- The details on how this function should work is described in Milestone 4.

```c
void updateContact(struct Contact[], int);
```

- **updateContact** does not return a value and receives a **Contact** array, and an integer for parameters.
- The details on how this function should work is described in Milestone 4.

```c
void deleteContact(struct Contact[], int);
```

- **deleteContact** does not return a value and receives a **Contact** array, and an integer for parameters.
- The details on how this function should work is described in Milestone 4.

```
void sortContacts(struct Contact[], int);
```

- **sortContacts** does not return a value and receives a **Contact** array, and an integer for parameters.
- The details on how this function should work is described in Milestone 4.

Contact Helper Source File (.c)

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

Libraries

Include the system library **<string.h>** (refer to the source comments for placement). This library contains the needed string related functions you will use in some of the new functions. Be sure to also include the **contactHelpers.h** header file.

Function Definitions

Function getTenDigitPhone is mostly defined for you (see below for the details), but **requires some modifications**:

```
void getTenDigitPhone(char telNum[]);
```

- **getTenDigitPhone** does not return a value and expects a C string character array sized for 10 characters (plus the null terminator).
- This function receives user input for a 10-digit phone number and stores the value to the parameter **telNum**.

Reminder
Arrays by variable name (ex: telNum) is a pointer type (points to the 1st element). This means any changes made using this variable will directly update the source it points to.

-

- If the entered value is not 10 characters in length or if it does not contain all numeric characters, a message is displayed and continues to prompt until a 10 character C string containing all numeric characters is entered:

```
>Enter a 10-digit phone number: <
```

- This function provides a failsafe method in receiving a 10-digit character C string array.
- **NOTE**: You need to modify this function to ensure all values entered are **numerical digits** **(this will be tested in milestone 4)**

Just as you did in MS1, add an empty function definition for all the new functions prototyped in the **contactHelpers.h** header file. To help you get going, findContactIndex is done for you. Use the provided source code comments for placement.

<u>Contacts Header File</u> (.h)

Open the header file **contacts.h**. There are a few additional lines of code added to the top and bottom areas of the file:

```
#ifndef CONTACTS_H_
#define CONTACTS_H_

// Header file contents...

#endif // !CONTACTS_H_
```

<u>Brief Explanation</u>
*These lines of code "**safeguard**" the header file to ensure it is only included once when the application is compiled (when it is included multiple times by other files). This is an advanced topic not covered in this course but will be covered in the next level course C++ (OOP244/BTP200).*

Refer to the comments in the **contacts.h** file and carefully insert your code from MS2 where directed.

There are no further modifications to make to this file for this milestone.

<u>Contacts Source File</u> (.c)

The addition of the new helper function getTenDigitPhone, would benefit the getNumbers function in this module.

Copy your MS2 code where the comments indicate. Update the getNumbers function to use the new function getTenDigitPhone where appropriate to ensure a 10-digit character value is stored for the numbers.

<u>Hint</u>: The getTenDigitPhone function **does not display an initial prompt message**. The function will display a generic prompt only when and if an invalid value is entered. You should first display the prompt message specific to the case, then call the getTenDigitPhone function:

```
char examplePhone[11];
printf("Enter example phone number: ");
getTenDigitPhone(examplePhone);
```

Milestone 3 Submission

Milestone 3 is to be done in the lab and shown to your instructor, there is no need to submit on matrix.

Milestone 4 (20%)

(Due Four (4) days from assigned date)

Open the project file **A2MS4** and look inside (expand "Header Files" and "Source Files" folders – see figure 2.4.1). Do not modify the source code file: **a2ms4.c**. This is the main file used to assess your functions to determine if they work to this milestone's specifications.

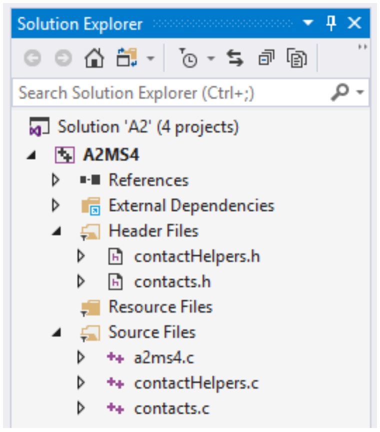


Figure: 2.4.1 – Visual Studio Project Contents

Unchanged Files

contacts.h
contacts.c
contactHelpers.h

Open each of these files in the provided project. Consult the comments provided and copy the appropriate contents from Milestone 3.

Contact Helper Source File (.c)

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

2. Define `MAXCONTACTS 5` where the comments indicate (under the include section). This will be used for sizing the contacts array later in the program

3. Copy the contents from MS3 where the comments indicate.

4. Complete the full function definitions for each function using the descriptions below:

Function Definitions

`void getTenDigitPhone(char telNum[]);`

- **Modify** this function to ensure all values entered are **numerical digits**

`int findContactIndex(const struct Contact[], int, const char[]);`

- **findContactIndex** returns an integer and expects a **Contact** array (marked constant so changes can't be made to it), and integer (intended for representing the size of the Contact array), and a character array (expecting a C string representing a cell phone number).
- This function's purpose is to find a contact based on the provided cell phone number (parameter 3) and return the index (position in the array) otherwise, if the contact is not found, the function should return -1

  Hints
  - *Iterate the Contact array (up to the size specified in parameter-2) and compare the current iteration contact's cell phone number to the parameter-3 cell phone number to determine equality*
  - *Use a **string library function** for string compare*

`void displayContactHeader(void);`

- **displayContactHeader** does not return a value and has no parameters.
- This function display's a centered title "Contacts Listing" header that is surrounded in a border:

```
+-----------------------------------------------------------------------------+ [line-1]
|                              Contacts Listing                               | [line-2]
+-----------------------------------------------------------------------------+ [line-3]
```

Line-1 and Line-3:
`'+'` one (1) plus symbol
`'-'` seventy-seven (77) minus symbols
`'+'` one (1) plus symbol
`Followed by a newline '\n'`

Line-2:
`'|'` one (1) vertical pipe symbol (usually located above the enter key)
`' '` thirty (30) spaces
`'Contacts Listing'` (the title)
`' '` thirty (31) spaces
`'|'` one plus symbol
`Followed by a newline '\n'`

`void displayContactFooter(int);`

- **displayContactFooter** does not return a value and receives one integer parameter.
- This function display's a line and a total contact summary
- The function parameter is the total number of contacts value to display

```
+-----------------------------------------------------------------------------+ [line-1]
Total contacts: 99     [line-2]
```

`Line-1:`
`'+'` one (1) plus symbol
`'-'` seventy-seven (77) minus symbols
`'+'` one (1) plus symbol
`Followed by a newline '\n'`

`Line-2:`
`'Total contacts: '` Note: one (1) space after the colon
Followed by the integer parameter value
Followed by <u>TWO</u> (2) newlines (`'\n'`)

`void displayContact(const struct Contact*);`

- **displayContact** does not return a value and receives a constant **Contact** pointer (cannot be updated) parameter.
- This function displays the details for a single contact (see example below):

Example: With middle initial and an apartment number:
```
 Maggie R. Greene                                        [line-1]
    C: 9051112222   H: 9052223333   B: 9053334444        [line-2]
       55 Hightop House, Apt# 201, Bolton, A9A 3K3       [line-3]
```

Example: <u>No middle initial</u> and <u>no apartment number</u>:
```
 Maggie Greene                                           [line-1]
    C: 9051112222   H: 9052223333   B: 9053334444        [line-2]
       55 Hightop House, Bolton, A9A 3K3                 [line-3]
```

`Line-1 (contact full name):`
`' '` one (1) blank space followed by…
`'Contact full name'` (See below for details) followed by…
One (1) newline '\n'

<u>Contact Full Name</u>:
- Consists of all the name parts put together separated by a space
- If there is no middle initial then there should only be one (1) space separating the first name and the last name

`Line-2 (contact numbers):`
Use the following format string:
`"    C: %-10s   H: %-10s   B: %-10s\n"` (Note: 4 spaces at the beginning)
- The 1st specifier is for the cell phone number
- The 2nd specifier is for the home phone number
- The 3rd specifier is for the business phone number

`Line-3 (contact address):`
Use the following format string to get started:

`"       %d %s, "` (Note: 7 spaces at the beginning and 1 after the comma)

- The 1st specifier is for the street number
- The 2nd specifier is for the street name

IF the apartment field has a value (will be > 0) then include on the same line:

`"Apt# %d, "` (Note: 1 space after the comma)

- The specifier is for the apartment number

Followed by the city and postal code comma separated:

`"%s, %s\n"`

- The 1st specifier is for the city
- The 2nd specifier is for the postal code

`void displayContacts(const struct Contact[], int);`

- **displayContacts** does not return a value and receives a constant **Contact** array (cannot be updated), and an integer (intended for representing the size of the Contact array).
- This function needs to display a title header, a detail listing of each **VALID** contact in the array and a footer section showing the total number of contacts.
- To produce the title header, call the displayHeader function described earlier in this milestone
- You will need to iterate all items in the **Contact** array (parameter 1) up to the number of items it holds (parameter 2).
- **IF** the current iterator **Contact** is **VALID** (will have a cell number > 0), display the details (call the displayContact function described earlier in this milestone).
- After iterating all the contacts, finish with displaying the footer summary section by calling the displayContactFooter function.

Hint: Keep track of the number of valid contacts displayed and pass this count to the displayContactFooter function

`void searchContacts(const struct Contact[], int);`

- **searchContacts** does not return a value and receives a constant **Contact** array (cannot be updated), and an integer (intended for representing the size of the Contact array).
- This function should prompt the user to search the contact listings based on a cell phone number. If there is a match, display the details for the contact.
- Prompt the user for a cell phone number to search:
  >`Enter the cell number for the contact: `<
  - Do not allow the application to continue until a 10-digit phone is entered. (**Hint**: call the getTenDigitPhone function to get the value)
- Search the contact array to find a contact with the cell phone number previously entered (**Hint**: call function findContactIndex)

- If the contact is found, display the details (**Hint**: call function displayContact)
  Note: follow with one (1) newline ('\n')
- Otherwise, show the following message:
  `>*** Contact NOT FOUND ***< (add a newline '\n')`

```
void addContact(struct Contact[], int);
```

- **addContact** does not return a value and receives a **Contact** array, and an integer (intended for representing the size of the Contact array).
- This function should first determine if there is an available slot (empty index) in the contact array. This requires iterating the contact array and looking for the first available element having a cell number string length equal to zero (0). **Hint**: use the string library's string length function.
- If there are no available slots, the contact listing is full. Notify the user displaying the following message:
  `>*** ERROR: The contact list is full! ***< (add a newline '\n')`
- Otherwise, prompt the user to enter the details for the new contact (store to the contact array at the determined empty slot index). **Hint**: use the getContact function
- After adding the new contact, display the following message:
  `>--- New contact added! ---< (add a newline '\n')`

```
void updateContact(struct Contact[], int);
```

- **updateContact** does not return a value and receives a **Contact** array, and an integer (intended for representing the size of the Contact array).
- This function should prompt the user for the cell phone number of the contact to update.
- Search the contact array to find a contact with the cell phone number previously entered. **Hint**: call the findContactIndex function
- If the contact could NOT be located, display the following message:
  `>*** Contact NOT FOUND ***< (add a newline '\n')`
- Otherwise, if the contact is located, display the following message:
  Print a newline ('\n') followed by:
  `>Contact found:<`
  Follow with another newline ('\n')
- Then display the details for the contact. **Hint**: call the displayContact function – followed by a newline ('\n')
- Prompt the user with the following message:
  `>Do you want to update the name? (y or n): <` **Hint**: call the yes function
  - If the user responds yes ('y'|'Y') call the function getName to update the name member of the contact
- Follow with a prompt displaying the following message:
  `>Do you want to update the address? (y or n): <` **Hint**: call the yes function
  - If the user responds yes ('y'|'Y') call the function getAddress to update the address member of the contact

- Follow with a prompt displaying the following message:
  >`Do you want to update the numbers? (y or n): `<   **Hint**: call the yes function
  - If the user responds yes ('y'|'Y') call the function getNumbers to update the numbers member of the contact
- Lastly display a confirmation message to indicate the contact information was updated:
  >`--- Contact Updated! ---`< `(add a newline` '\n')

`void deleteContact(struct Contact[], int);`

- **deleteContact** does not return a value and receives a **Contact** array, and an integer (intended for representing the size of the Contact array).
- This function should prompt the user for the cell phone number of the contact to delete from the list.
- Search the contact array to find a contact with the cell phone number previously entered. **Hint**: call the findContactIndex function
- If the contact could NOT be located, display the following message:
  >`*** Contact NOT FOUND ***`< `(add a` newline '\n')
- Otherwise, if the contact is located, display the following message:
  Print a newline ('\n') followed by:
  >`Contact found:`<
  Follow with another newline ('\n')
- Then display the details for the contact. **Hint**: call the displayContact function – followed by a newline ('\n')
- Prompt the user with the following message:
  >`CONFIRM: Delete this contact? (y or n): `<   **Hint**: call the yes function
  - If the user responds yes, set the contact's cell number to an empty string (**Hint**: assign a null terminator character to the first element)
- Lastly display a confirmation message to indicate the contact was deleted:
  >`--- Contact deleted! ---`< `(add a newline` '\n')

NOTE
*The **sortContacts** function is **OPTIONAL** (**unless otherwise specified by your instructor**)*

- *If you successfully define this function (and if your instructor does not require you to do this function), bonus marks will be given to compensate your extra effort.*

If you chose NOT to do this function, display the following message:

>`<<< Feature to sort is unavailable >>>`< (add a newline '\n')

`void sortContacts(struct Contact[], int);`

- **sortContacts** does not return a value and receives a **Contact** array, and an integer (intended for representing the size of the Contact array).
- This function reorders the elements in the contacts array from lowest to highest based on the cell phone numbers.

- After sorting is done (the array is reordered), display the following message:
  `>--- Contacts sorted! ---<` (add a newline '\n')
- **Hint**: Review the course notes on Algorithms (specifically the sections on "**Sorting**")

5. Lastly, revise the ContactManagementSystem function.
   - Create a **Contact** array sized to `MAXCONTACTS` and provide it a meaningful name.
     - Initialize the array to hold the following:

```
{ { { "Rick", {'\0'}, "Grimes" },
    { 11, "Trailer Park", 0, "A7A 2J2", "King City" },
    { "4161112222", "4162223333", "4163334444" } },
  {
     { "Maggie", "R.", "Greene" },
     { 55, "Hightop House", 0, "A9A 3K3", "Bolton" },
     { "9051112222", "9052223333", "9053334444" } },
  {
     { "Morgan", "A.", "Jones" },
     { 77, "Cottage Lane", 0, "C7C 9Q9", "Peterborough" },
     { "7051112222", "7052223333", "7053334444" } },
  {
     { "Sasha", {'\0'}, "Williams" },
     { 55, "Hightop House", 0, "A9A 3K3", "Bolton" },
     { "9052223333", "9052223333", "9054445555" } },
 };
```

   - Next, refer to the below for what function should be called for each menu selection:

```
Contact Management System
-------------------------
1. Display contacts                       Call function: displayContacts
2. Add a contact                          Call function: addContact
3. Update a contact                       Call function: updateContact
4. Delete a contact                       Call function: deleteContact
5. Search contacts by cell phone number   Call function: searchContacts
6. Sort contacts by cell phone number     Call function: sortContacts
0. Exit

Select an option:>
```

Sample Output (NO Sorting Option)

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

```
<<< TO BE DETERMINED >>>
```

Sample Output (With Sorting Option)

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

```
<<< TO BE DETERMINED >>>
```

Milestone 4 Reflection (20%)

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

3. Briefly explain how the two functions **findContactIndex** and **getTenDigitPhone** benefit your source code with respect to overall maintenance and readability.

4. What did you learn most from doing Assignment 2?

Note: When completing the reflections it is a violation of academic policy to cut and paste content from the course notes or any other published source, or to copy the work of another

Milestone 4 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 ms4 contacts.c contactHelpers.c a2ms4.c <ENTER>
```

This command will compile your code and name your executable "**ms4**". Execute **ms4** and make sure everything works properly.

Finally run the appropriate script from your account: (replace profname.proflastname with your professors Seneca userid):

```
[NO Sorting Option]
~profname.proflastname/submit 144_a2ms4 <ENTER>
```

<<< OR >>>

```
[WITH Sorting Option]
~profname.proflastname/submit 144_a2ms4Sort <ENTER>
```

and follow the instructions.

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