Fall 2019

Contacts

Assignment 1

The two (2) assignments this semester will revolve around building a Contact system that is searchable and efficient.

In the first assignment, you will build a Contact structure that contains other structures representing the contact Name, Address and phone Numbers.

Introduction to C Strings

The section of the notes regarding C strings is not completely covered until week 11, however you will need to know some very simple facts about string handling in C for this assignment. If you wish, you can read more about string handling here: http://scs.senecacollege.ca/~ipc144/pages/content/strin.html

A C string is an array of type char with a special terminator character called the null byte. When declaring a C string array it is necessary to always make the array one character larger than the maximum number of characters it needs to be able to store.

If we need to be able to store up to **30 characters**, this is the declaration of the array:

```
char firstName[31]; // the size is 31!
```

To read a C string (user input), code the following:

```
scanf("%30s", firstName);
```

***Note**: There is no ampersand (&) before firstName, the name of an array is its address. 30 specifies the max number of characters to be read*

To print a C string, code the following:

```
printf("%s\n", firstName);
```

Here is a code sample that reads and writes a C string:

```
#include<stdio.h>

#define NAME_SIZE 30
int main(void)
{
	char firstName[NAME_SIZE + 1];

	printf("Enter the contact's first name, maximum 30 characters: ");
	scanf("%30s", firstName); //user enters: Fred

	printf("You entered: %s.\n", firstName); //outputs: You entered: Fred.

	return 0;
}
```

Milestone 1 (10%)

(Due: By the end of your lab class)

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

Open the project file for MS1 and look inside. You will find a file named contacts.h. The .h extension identifies this file as a header file. Header files contain declarations of structs and function prototypes. In this header file, the struct Name is already declared.

Structure Name has three (3) members: **firstName**, a C string that can hold up to 30 characters **middleInitial**, a C string that can hold up to 6 characters, and **lastName**, a C string that can hold up to 35 characters.

```
struct Name
{
	char firstName[31];
	char middleInitial[7];
	char lastName[36];
};
```

Declare two (2) more structures named Address, and Numbers, in the header file, **contacts.h**

Structure Address has five (5) members: **streetNumber, street**, **apartmentNumber**, **postalCode**, and **city**.

streetNumber:	int, (**logic** for this field should enforce values greater than 0)
street:	C string, up to 40 characters
apartmentNumber:	int, (**logic** for this field should enforce values greater than 0)
postalCode:	C string, up to 7 characters
city:	C string, up to 40 characters

Structure Numbers has three (3) members: **cell**, **home**, and **business**. These are all C strings that can hold up to 10 characters.

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)

Application Logic

Open the project file for MS2 and look inside. You will find a source file named **a1ms2.c**. Use this file and in it code the main() function by implementing the following. Make sure that your project contains the **contacts.h** header file from milestone 1:

- Declare a variable of type Name (use a self-describing variable name) to be used for storing a contact's full name
  - Initialize each member to an empty C string
- Declare a variable of type Address (use a self-describing variable name) to be used for storing a contact's address information
  - Initialize the members so numeric values are set to zero and char arrays are set to an empty C string
- Declare a variable of type Numbers (use a self-describing variable name) to be used for storing a contact's phone(s) information
  - Initialize each member to an empty C string
- Display to the screen a welcome message:

```
>Contact Management System<
>-------------------------<  (25 dashes followed by a single newline)
```

Note:
Structure member's **middleInitial**, **apartmentNumber**, **cell**, **home**, and **business** are all **optional values**. Not all people have a middle initial, live in an apartment, or have multiple phone numbers. When asking the user for this information you should first ask if they wish to enter it:

Member **middleInitial** prompt (member of the Name type):

```
>Do you want to enter a middle initial(s)? (y or n): <
```

Member **apartmentNumber** prompt (member of the Address type):

```
>Do you want to enter an apartment number? (y or n): <
```

Member **cell** prompt (member of the Numbers type):

```
>Do you want to enter a cell phone number? (y or n): <
```

Member **home** prompt (member of the Numbers type):

```
>Do you want to enter a home phone number? (y or n): <
```

Member **business** prompt (member of the Numbers type):

```
>Do you want to enter a business phone number? (y or n): <
```

Figure: 1.2.1 - Prompts

Contact Name

- Prompt the user to enter the required member data for the Name type. First, ask for the first name:

  >`Please enter the contact's first name:` <
  - Read and store the C string value the user enters into the appropriate Name member

- Prompt the user if a middle initial(s) value needs to be entered (*see **figure: 1.2.1** above regarding optional values for example prompt*)
  - Evaluate the entered value; if an 'n' is entered proceed with the next member entry otherwise prompt for the middle initial(s):
  - >`Please enter the contact's middle initial(s):` <
  - Read and store the C string value the user enters into the appropriate Name member

- Prompt the user to enter the last name:
  >`Please enter the contact's last name:` <
  - Read and store the C string value the user enters into the appropriate Name member

Contact Address

- Prompt the user to enter the required member data for the Address type. First, ask for the street number:
  >`Please enter the contact's street number:` <
  - Read and store the number entered by the user into the appropriate Address member
  - Hint: Review the specifications for this structure member's valid values

- Prompt the user to enter the street name:
  >`Please enter the contact's street name:` <
  - Read and store the C string value the user enters into the appropriate Address member

- Prompt the user if an apartment number needs to be entered (*see **figure: 1.2.1** above regarding optional values for example prompt*)
  - Evaluate the entered value; if an 'n' is entered proceed with the next member entry otherwise prompt for the apartment number:
  - >`Please enter the contact's apartment number:` <
  - Read and store the number entered by the user into the appropriate Address member
  - Hint: Review the specifications for this structure member's valid values

- Prompt the user to enter the postal code:
  >`Please enter the contact's postal code:` <
  - Read and store the C string value the user enters into the appropriate Address member

- Prompt the user to enter the city:
  >`Please enter the contact's city:` <
  - Read and store the C string value the user enters into the appropriate Address member

Contact Numbers

- Prompt the user to enter the required member data for the Numbers type. First, prompt the user if a **cell phone number** needs to be entered (*see **figure: 1.2.1** above regarding optional values for example prompt*)
  - Evaluate the entered value; if an 'n' is entered proceed with the next member entry otherwise prompt for the cell phone number:
  - >Please enter the contact's cell phone number: <
  - Read and store the number entered by the user into the appropriate Numbers member

- Prompt the user if a **home phone number** needs to be entered (*see **figure: 1.2.1** above regarding optional values for example prompt*)
  - Evaluate the entered value; if an 'n' is entered proceed with the next member entry otherwise prompt for the home phone number:
  - >Please enter the contact's home phone number: <
  - Read and store the number entered by the user into the appropriate Numbers member

- Prompt the user if a **business phone number** needs to be entered (*see **figure: 1.2.1** above regarding optional values for example prompt*)
  - Evaluate the entered value; if an 'n' is entered don't prompt for this value otherwise prompt for the business phone number:
  - >Please enter the contact's business phone number: <
  - Read and store the number entered by the user into the appropriate Numbers member

Test the above logic and your 3 structures by referring to the below sample output (input's are identified in **RED**). Your output should match exactly:

Sample Output

```
Contact Management System
-------------------------
Please enter the contact's first name: Tom
Do you want to enter a middle initial(s)? (y or n): y
Please enter the contact's middle initial(s): L. A.
Please enter the contact's last name: Wong Song
Please enter the contact's street number: 20
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: 40
Please enter the contact's postal code: A8A 4J4
Please enter the contact's city: North York
Do you want to enter a cell phone number? (y or n): Y
Please enter the contact's cell phone number: 9051116666
Do you want to enter a home phone number? (y or n): Y
Please enter the contact's home phone number: 7052227777
```

```
Do you want to enter a business phone number? (y or n): Y
Please enter the contact's business phone number: 4163338888

Contact Details
---------------
Name Details
First name: Tom
Middle initial(s): L. A.
Last name: Wong Song

Address Details
Street number: 20
Street name: Keele Street
Apartment: 40
Postal code: A8A 4J4
City: North York

Phone Numbers:
Cell phone number: 9051116666
Home phone number: 7052227777
Business phone number: 4163338888

Structure test for Name, Address, and Numbers Done!
```

Milestone 2 Reflection (20%)

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**:

- Can you think of a more efficient way to ask a user to add the required information to each data field? Justify your thoughts with an example.
- Discuss the differences between a C string and a primitive character array. What would happen if you attempt to display the contents of a primitive character array (not a C string) using the printf specifier "%s"?

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**, **a1ms2.c**, and **reflect.txt** files to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms2 a1ms2.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 144a1ms2/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.