Functions with Pointers

Workshop 7 (out of 10 marks - 4% of your final grade)

In this workshop, you will code a C-language program with functions that change the values of variables declared outside the functions.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities:

- to pass an address to a function in a function call
- to refer to an address stored in a function's parameter.
- to implement structured programming principles, including single-entry/single-exit logic
- to describe to your instructor what you have learned in completing this workshop

SUBMISSION POLICY

The "in-lab" section is to be completed during your assigned lab section. It is to be completed and submitted by the end of the workshop period. If you attend the lab period and cannot complete the in-lab portion of the workshop during that period, ask your instructor for permission to complete the in-lab portion after the period. If you do not attend the workshop, you can submit the "in-lab" section along with your "at-home" section (with a penalty; see below). The "at-home" portion of the lab is due on the day that is two days before your next scheduled workshop (23:59).

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to back up your work regularly.

Late submission penalties:

- In-lab portion submitted late, with at-home portion: 0 for in-lab. Maximum of 70/70 for at-home and reflection
- If any of in-lab, at-home or reflection portions is missing the mark will be zero.

IN-LAB: (30%)

Download or clone workshop 7 (**WS07**) from https://github.com/Seneca-144100/IPC-Workshops

Write your code for this segment in the phone_app_in_lab.c file provided with the Visual Studio template project inside the in_lab folder.

In this segment, you will implement Add and Display functionality using a C array and pointer syntax.

OVERVIEW

The phone_app_in_lab.c template file provides a framework for the following functionality:

- Displays a menu list as shown below inside a do-while iteration construct:
    1. Display Phone List
    2. Add a Number
    0. Exit

- Captures user input for the above menu options. Stores the input to an `int` variable named **option**

- Ability to iterate multiple menu choices (until the user enters 0) with a selection construct (switch) that directs process flow to the required logic/functionality. Refer to the comments for each case to identify the specific functionality.

- Displays an error message for invalid menu option selections in the `default` case

- Outputs prompt information for menu options 1 & 2

Your task is to complete the following.

ALLOCATE MEMORY FOR PHONE NUMBER DATA

- Define the maximum number of phone numbers (**SIZE**) to be 3 using the #`define` directive and insert it between the library directive and the **main** function definition
- Define an array of `long long` types named **phoneNumber** that can hold up to **SIZE** elements. Initialize all elements to be 0LL. Note that the suffix LL identifies the constant 0 as a constant of `long long` type.

DEFINE AND IMPLEMENT THE DECOMPOSE FUNCTION

- Prototype a C function named **decompose** that returns nothing and has four parameters. The first parameter is an unmodifiable `long long` that receives a phone number. The remaining parameters receive the addresses of `int` variables declared outside the function.
- Implement the definition of your **decompose** function to break-down the phone number received into its area code (AAA), prefix (PPP) and line number (LLLL) components using the conventional template AAAPPPLLLL. (Hint: you can use integer division and the modulus operator to extract AAA, PPP and LLLL). Assign the component values to the variables stored at the received addresses. **Note:** Assigning a value to a pointer will in fact change the value of the actual variable defined in the caller function

IMPLEMENT EXIT PROGRAM FUNCTIONALITY IN CASE 0

- Print the exiting message.

```
> Exiting Phone Number App. Good Bye!!! <
```

IMPLEMENT DISPLAY FUNCTIONALITY IN CASE 1

- Display the non-empty elements of the **phoneNumber** array. Use the following formatting in a call to the `printf` function:

```
(%3d)-%3d-%4d ←→ (area code, prefix, line number)
```

- After completing the display, enter a newline using a call to the `printf` function.

IMPLEMENT ADD A PHONE NUMBER FUNCTIONALITY IN CASE 2

- Keep track of the number of phone numbers accepted using an `int` variable.
- Check if the **phoneNumber** array is full.
- If the array is full, display the following error message.

```
> ERROR!!! Phone Number List is Full <
```

- If the **phoneNumber** array is not full, accept the number from the user, store it as an element, and increment the number of elements stored by one.

PROGRAM COMPLETION

Your program is complete if your output matches the following output. Red numbers show the user's input.

```
---=== Phone Numbers ===---

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 1

Phone Numbers
==============

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2

Add a Number
============
4164915050

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2

Add a Number
============
9052301212

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2

Add a Number
============
6475551212

1. Display Phone List
2. Add a Number
```

```
0. Exit

Please select from the above options: 1

Phone Numbers
==============
(416)-491-5050
(905)-230-1212
(647)-555-1212

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2

Add a Number
============
ERROR!!! Phone Number List is Full

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 0

Exiting Phone Number App. Good Bye!!!
```

IN_LAB SUBMISSION:

To test and demonstrate execution of your program use the same data as the output example above or any information needed.

If not on matrix already, upload your phone_app_in_lab.c to your matrix account. Compile and run your code and make sure everything works properly.

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

~profname.proflastname/submit 144_w7_lab <ENTER>

and follow the instructions.

Please note that 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.

AT_HOME: (30%)

In this segment, you upgrade your app to include input validation. **Copy** your in-lab source file phone_app_in_lab.c then **rename** it to phone_app_home.c and add the following additional functionality:

DEFINE AND IMPLEMENT THE ISVALID FUNCTION

- Prototype a C function named **isValid** that returns an `int` and has one parameter. The parameter is an unmodifiable `long long`.
- Implement your **isValid** function to return true if the phone number received is valid, false otherwise. Place your function definition below the **main** function. The number received by your function is valid if it contains one of the GTA area codes (416, 647 and 905) and if its prefix is between 100 and 999 inclusive. Include in your design an array of `int`s that contains the valid area codes. Extract the area code from the number received and check it against each element in this array. Your function returns true if the number is valid, false otherwise.

UPGRADE ADD A PHONE NUMBER FUNCTIONALITY IN CASE 2

- Accept a phone number from the user and store it in a temporary variable.
- Check if the number entered is valid.
- Only if the number entered is valid, store it in the **phoneNumber** array and increment the number of elements saved by one.
- If the number entered is invalid, DO NOT save it in the **phoneNumber** array, but display the following error message:

```
> ERROR!!! Phone Number is not Valid <
```

PROGRAM COMPLETION

Your program is complete if your output matches the following output. Numbers in red color shows the user's input.

```
---=== Phone Numbers ===---

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 1

Phone Numbers
```

```
==============

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2

Add a Number
============
6475551212

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2

Add a Number
============
-2345
ERROR!!! Phone Number is not Valid

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 4

ERROR!!!: Incorrect Option: Try Again

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2

Add a Number
============
4164915050

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2

Add a Number
============
6132223333
ERROR!!! Phone Number is not Valid

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2
```

```
Add a Number
============
6470223333
ERROR!!! Phone Number is not Valid

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 2

Add a Number
============
9052301212

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 1

Phone Numbers
==============
(647)-555-1212
(416)-491-5050
(905)-230-1212

1. Display Phone List
2. Add a Number
0. Exit

Please select from the above options: 0

Exiting Phone Number App. Good Bye!!!
```

AT-HOME REFLECTION (40%)

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

1) In 2 or 3 sentences explain the advantage of passing (receiving) values from a function through its parameters rather than through its return value?
2) Define the term dereference and identify an example in your code.
3) Explain what is stored in a pointer variable.

Note: when completing the workshop 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 student.

AT_HOME SUBMISSION:

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload your phone_app_home.c and reflect.txt to your matrix account. Compile and run your code and make sure everything works properly.

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

```
~profname.proflastname/submit 144_w7_home <ENTER>
```

and follow the instructions.

Please note that 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.