C-Language 4x4 Matrix Keypad & Parallel Arrays

Objective:
Students are introduced to the 4x4 Matrix keypad hardware. They will learn how the keypad is wired to read keypresses, how it is interfaced and memory mapped on our MC9S12 microcontroller and then apply their understanding of the keypad operation and interface to write a program that *simulates* reading a *keypress hardware value*. The actual keypad is not used in this laboratory, but is incorporated in the next lab.

Pre-Lab:
Before you begin to code!

1. Read all instructions completely.
2. Use the Program Development Cycle Worksheet to complete steps 1-3 to design an outline for your program. The outline should be in a word document which you will submit to this assignment. An outline is not a copy of your code, but rather your pre-code design/outline and flow chart as exampled in the Program Development Cycle Worksheet.
3. In-lab points given for students completing this pre-lab during the assigned lab time.

Program:
Write, build and test a C-language program that determines which keypad value is pressed on a 4x4 Matrix keypad. This program will not read the real keypad hardware but instead it will provide logic for a simulate keypress value on a keypad. The program will use a pre-determined hardware keypad keypress value that is passed to two functions. Each function is used to search parallel arrays to find a match to the hardware Hex Character Value passed (simulated keypress value) and returns a value that represents the pre-determined hardware keypad keypress.

The first function receives a value for a detected key press value (simulated key press), searches an array and returns the corresponding "Keypad Key Press" number which represents a numeric value for a detected key Press. The returned values range from 0-9 for the numeric keys. If a numeric "Keypad Key Press" match is not found for the value passed, the function returns a 0xff.

The second function receives a value for a detected key press value (simulated key press), searches an array and returns the corresponding ASCII value for a "Keypad Key Press" and will return the ASCII value for the numbers 0-9, A, B, C, D, * or #. If a "Keypad Key Press" match is not found for the value passed the function returns a 0xff.

An efficient way to search for a match to the "detected key press value" passed to the function is to put "detected key press values" data into an array and search the array for the value passed. A parallel array search can be used to hold the return values "Keypad Key Press". When a match is found, return the value in the parallel at the index of the match.

The following table shows the parallel relationship between a hardware detected keypress and the correlated keypad value.

Values read for detected key press hardware and corresponding Keypad Key Presses:

Hardware Hex Character Value (detected key press value)	Keypad key press value
0xd7	0
0xee	1
0xde	2
0xbe	3
0xed	4
0xdd	5
0xbd	6
0xeb	7
0xdb	8
0xbb	9
0x7e	A
0x7d	B
0x7b	C
0x77	D
0xe7	*
0xb7	#

The *hardware key press value data* array can be declared as below.

```
const unsigned char detectedKeyPress[ ] = {0xd7, 0xee, 0xde, 0xbe,
                                           0xed, 0xdd, 0xbd, 0xeb,
                                           0xdb, 0xbb, 0x7e, 0x7d,
                                           0x7b, 0x77, 0xe7, 0xb7};
```

Be sure to declare the two other arrays in such a way that their corresponding data values are parallel to the "detected key press values"
All three arrays should be declared globally in a file called Keypad.c

Use relevant names for variables and functions. Put the code for the two functions in the Keypad.c file. Add this file to your project.

Put function prototypes in a file named protos.h. (if you have forgotten how to add files to your project, refer to Lab 01).

You can complete this program at home using the demo software.

To test your functions:
Put calls to your functions in main.c. Pass any one of the "detected key press values" (Note: we are simulating a key press, so for now, you will hard code a value from the detected key press table into your function call). Next week we will use this program to complete a real-time key press program.

When debugging: In the data debugger window, change a displayed number base to hexadecimal by right clicking on a variable then select Format | Selected | Hex.

When testing to verify your program: be sure to declare your returned function value as volatile. This will ensure it gets written to.
Example:

```
volatile unsigned char result;
volatile unsigned char valuePassed = 0xee;   //0xee is an example value from the table

result = YourSearchFunction(valuePassed);
```

If using the simulated "detected key press value" 0xee then, depending on which search function is called, result should be either:
0x01 (returned value 1 decimal) for the first function that has the decimal array. Or
0x31 (returned value hex 31) for the second routine that has the ASCII array.

The variable *result* is declared as volatile. This will ensure it is not optimized, meaning, a value is written to every time it is referenced and no optimized.

Before you submit your solution and files

1. Be sure to comment your code and follow good coding practices.
2. Put a commented header in each of the following program files: main.c, Keypad.c, protos.h

Submit:

- Program files: your main.c, Keypad.c, protos.h files and entire zipped project,
- Word document: well organized word document .docx including your fully documented program logic outline and program flow chart design as exampled in the Program Development Cycle worksheet steps 1-3.

Please observe due date as late submissions are not accepted, and files sent via email or email attachments are not accepted.