Lab 09: Arrays

CSE 4108

Structured Programming I Lab

October 2022

Lab Tasks

1. **Repeat After Me:**

Write a C program that takes an integer as an input and shows which digits (if any) were repeated.

Sample input:
Enter a number: 939577
Sample output:
Repeated digit(s): 7 9

2. **High Frequency!:**

Modify the **"Repeat After Me"** program so that it prints a table showing how many times each digit appears in the number.

Sample input:
Enter a number: 41271092
Sample output:

Digit:	0	1	2	3	4	5	6	7	8	9
Occurrences:	1	2	2	0	1	0	0	1	0	1

3. **Repeating Again:**

Modify the **"Repeat After Me"** program so that the user can enter more than one number to be tested for repeated digits. The program should terminate when the user enters a number that's less than or equal to 0.

4. **Variable Length Reversing:**

Modify the “reverse.c” program (Can be found in page - 164 of C Programming A Modern Approach book) to use the expression **(int) (sizeof(a) / sizeof(a[0]))** (or a macro with this value) for the array length.

****** What happens when you use a different name (from your array declaration in the main function) for the array in macro declaration?

5. **Hmmm Interesting!:**

Modify the **“interest.c”** program (Can be found in page - 168 of C Programming A Modern Approach book) so that it compounds interest monthly instead of annually. The form of the output shouldn’t change; the balance should still be shown at annual intervals.

6. **B1FF Translator!**:

You made a friend in the famous social network mySpace, named B1FF. He has a unique way of writing messages. The message length doesn’t exceed 100 characters.
Here’s a typical B1FF communiqué:
H3Y DUD3, C 15 C00L!!!!!!!!!!

Write a “B1FF translator” that reads a message entered by the user and translates it into B1ff-speak.

Sample input:
Enter message: Hey dude, C is cool
Sample output:
In B1FF-speak: H3Y DUD3, C 15 C00L!!!!!!!!!!

Your program should convert the message to upper-case letters, substitute digits for certain
letters (A⇒4, B⇒8, E⇒3, I⇒1, O⇒0, S⇒5), and then append 10 or so exclamation marks.

7. **Matrix Sum:**

Write a program that reads a 5 × 5 array of integers and then prints the row sums and the column sums.

Sample input:

Enter row 1: 8 3 9 0 10
Enter row 2: 3 5 17 1 1
Enter row 3: 2 8 6 23 1
Enter row 4: 15 7 3 2 9
Enter row 5: 6 14 2 6 0

Sample output:

Row totals: 30 27 40 36 28
Column totals: 34 37 37 32 21

8. **Quiz Marks:**

Modify the last task so that it prompts for five quiz grades for each of five students, then computes the total score and average score for each student, and the average score, high score, and a low score for each quiz.

9. **Random Walk:**

Write a program that generates a "random walk" across a 10×10 array. The array will contain
characters (all '.' initially). The program must randomly "walk" from element to element, always going up, down, left, or right by one element. The elements visited by the program will be labeled with the letters 'A' through 'Z', in the order visited.

Here's an example of the desired output:

```
A . . . . . . . . .
B C D . . . . . . .
. F E . . . . . . .
H G . . . . . . . .
I . . . . . . . . .
J . . . . . . . Z .
K . . R S T U V Y .
L M P Q . . . W X .
. N O . . . . . . .
. . . . . . . . . .
```

Hint: Use the srand and rand functions (see deal.c in page - 172 of the book) to generate random numbers. After generating a number, look at its remainder when divided by 4. There are four possible values for the remainder - 0, 1, 2, and 3 - indicating the direction of the next move. Before performing a move, check that
(a) it won't go outside the array, and
(b) it doesn't take us to an element that already has a letter assigned. If either condition is violated, try moving in another direction. If all four directions are blocked, the program must terminate.
Here's an example of premature termination:

```
A B G H I . . . . .
. C F . J K . . . .
. D E . M L . . . .
. . . . N O . . . .
. . W X Y P Q . . .
. . V U T S R . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
```

Y is blocked on all four sides, so there's no place to put Z.

10. **Welcome Aboard 3:**

In **Lab 4 - Selection Statements (Welcome Aboard)**, you wrote a program that asks the user to enter a time (expressed in hours and minutes, using the 24-hour clock) and displays the departure and arrival times for the flight whose departure time is closest to that entered by the user.
Modify the program so that the departure times are stored in an array and the arrival times are stored in a second array. (The times are integers, representing the number of minutes since midnight). The program will use a loop to search the array of departure times for the one closest to the time entered by the user.

11. **That Sounds Phoney 2:**

In Lab 6 - Basic Types (That Sounds Phoney), you wrote a program that converted phonewords to phone numbers.
Modify the program so that the program labels its output.

Sample input:

Enter phone number: 1-800-DOM-INOS

Sample output:

1-800-366-4667

The program will need to store the phone number (either in its original form or in its numeric form) in an array of characters until it can be printed. You may assume that the phone number is no more than 15 characters long.

12. **GINGERS:**

In **Lab 6 - Basic Types (Scrabbler)**, you wrote a program that calculated the face value of a word by summing the values of its letters. Modify the program so that the Scrabble values of the letters are stored in an array. The array will have 26 elements, corresponding to the 26 letters of the alphabet. For example, element 0 of the array will store 1 (because the Scrabble value of the letter A is 1), element 1 of the array will store 3 (because the Scrabble value of the letter B is 3), and so forth.
As each character of the input word is read, the program will use the array to determine the Scrabble value of that character. Use an array initializer to set up the array.

13. **Harvard Citation Style:**

In Lab 6 - Basic Types (APA Citation Style), you wrote a program that takes a first name and last name entered by the user and displays the last name, a comma, and the first initial, followed by a period. Modify the program so that it labels its output.

Sample input:

Enter a first and last name: Daniel Atlas

Sample output:

Atlas, D.

The program will need to store the last name (but not the first name) in an array of characters until it can be printed. You may assume that the last name is no more than 20 characters long.

14. **Word Unit Palindromes:**

Write a program that reverses the words in a sentence.

Sample input:

Enter a sentence: you can cage a swallow can't you?

Sample output:

Reversal of sentence: you can't swallow a cage can you?

You can assume that the sentence will contain at most 80 characters long. If there's no terminating character, put a full stop at the end of your output.

15. **Caesar Cypher:**

One of the oldest known encryption techniques is the Caesar cipher, attributed to Julius Caesar. It involves replacing each letter in a message with another letter that is a fixed number of positions later in the alphabet. If the replacement would go past the letter Z, the cipher "wraps around" to the beginning of the alphabet. For example, if each letter is replaced by the letter two positions after it, then Y would be replaced by A, and Z would be replaced by B. Write a program that encrypts a message using a Caesar cipher. The user will enter the message to be encrypted and the shift amount (the number of positions by which letters should be shifted).

Sample input:
Enter a message to be encrypted: Smile, it's Sunnah
Enter shift amount (1-25): 3
Sample output:
Encrypted message: Vploh, lw'v Vxqqdk

Notice that the program can decrypt a message if the user enters 26 minus the original key.

Sample input:
Enter message to be encrypted: Vploh, lw'v Vxqqdk
Enter shift amount (1-25): 23
Sample output:
Encrypted message: Smile, it's Sunnah

You may assume that the message does not exceed 80 characters. Characters other than letters should be left unchanged. Lower-case letters remain lower-case when encrypted, and uppercase letters remain upper-case.

16. **Anagrams:**

Write a program that tests whether two words are anagrams (permutations of the same letters).

Sample input:
Enter first word: smartest
Enter second word: mattress
Sample output:
The words are anagrams.

Sample input:
Enter first word: dumbest
Enter second word: stumble
Sample output:
The words are not anagrams.

17. **Magic Square:**

Write a program that prints an n × n magic square (a square arrangement of the numbers 1,2, .. ., n2 in which the sums of the rows, columns, and diagonals are all the same). The user will specify the value of n.

Sample input:

This program creates a magic square of a specified size.
The size must be an odd number between 1 and 99.
Enter size of magic square: 5

Sample output:

The Magic Square is:

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

Store the magic square in a two-dimensional array. Declare the array to have n rows and n columns. Start by placing the number 1 in the middle of row 0. Place each of the remaining numbers 2,3, .. ., n2 by moving up one row and over one column. Any attempt to go outside the bounds of the array should "wrap around" to the opposite side of the array.

For example,
instead of storing the next number in row: -1, we would store it in row: n − 1 (the last row). Instead of storing the next number in column: n, we would store it in column 0. If a particular array element is already occupied, put the number directly below the previously
stored number.