CS 271 and 462

PA 4 - Programming Assignment 4

Goals:

- Become proficient in writing a makefile.
- Become proficient in writing functions in C.
- Become proficient in working with arrays in C.

Grading (40 points total):

1. 5 points - All programs follow documentation and style requirements.

2. makefile - 5 points

3. arrayFunctions.h - 5 points

4. arrayFunctions.c - 20 points (5 points per function)

5. pa4.c - 5 points

Create the makefile

1. Make a new directory (folder) for this assignment called PA4. Place all of the files you create for this assignment in that directory.
2. Open Kate. Start a new file. Save the file with the name makefile (no extension).
3. Type the makefile commands to compile, link, and create an executable named **pa4** (no extension).

   Note: I highly recommend typing this makefile without looking at the previous one. DO NOT use copy and paste. (The "Creating a Makefile" handout is included at the end of this document.)

   You must have an "all" target. The "clean" target is optional.

   The source files are named **arrayFunctions.c** and **pa4.c**.
4. Save the file.

The makefile CANNOT have comments that start with //. Don't write comments in the makefile.

Create the header file: arrayFunctions.h

5. Create a header file named arrayFunctions. In this file, write the preprocessor wrapper:

```
#ifndef ARRAYFUNCTIONS
#define ARRAYFUNCTIONS

     // here is where your function prototypes go

#endif
```

6. For each of the functions below, place a one-line prototype in the header file.

- **Print an integer array.** Write a function called **printIntArray** that will print the elements of an integer array, 10 per line. The values should be neatly right-aligned in columns and you may assume that each integer is no larger than 5 digits.

  1) an array of integers
  2) the size of the array (an integer)

- **Print a character array.** Write a function called **printCharArray** that will print the elements of an char array, 10 per line. The values should be separated by one space.

  1) an array of characters
  2) the size of the array (an integer)

- **Find the minimum value in an array**: Write a function named **arrayMin** that finds and returns the smallest value in an integer array. The parameters to the function are:

  1) an array of integers
  2) the size of the array (an integer)

- **Count the number of letters in a character array.** Write a function named **countLetters** that will count and return the number of letters (either uppercase or lowercase) that appear in the array. The parameters to the function are:

  1) an array of characters
  2) the size of the array (an integer)

  Do not write ASCII code numbers in your program. Use characters 'A', 'Z', etc.

- **Determine the number of elements that are the same in two arrays.** Write a function named **numMatches** that compares the elements in two arrays and counts how many elements are the same. The parameters to the function are:

  1) an array of integers
  2) a second array of integers
  3) the size of the arrays (an integer)

  Both arrays have the same number of elements.

  Example:

  Array 1:

3	7	2	5	9

  Array 2:

3	8	2	5	9

  There are 4 elements that match. The return value of numMatches should be 4.

Create the first source file: arrayFunctions.c

7. Create a source file named arrayFunctions.c   In this file, place the implementations (the full header and body) of the five functions above.   arrayFunctions.c should NOT have a main function.

   This file must have header comments

   Be sure to comment each function thoroughly.

Incremental Programming
Write one function. Write part of the main function. Test and debug that one function.   Then, write the next function.

Create the second source file: pa4.c

8. Create a test program named pa4.c. In this file, place the main function as follows:
   - This file must have header comments
   - Include arrayFunctions.h
   - Be sure to comment each function thoroughly.
   - The only function that can be in this file is the main function.
   - Include statements to completely and thoroughly test the five functions that you wrote in arrayFunctions.c
   - Your output should contain messages that clearly state which function is being tested.

Move files to Linux, make, run

9. FTP your files to Linux. At the Linux command prompt, type
   ```
   make
   ```
10. Correct errors, FTP files again, repeat until there are no errors in the make process.
11. Run the executable.   Test, debug, test, repeat…until everything works correctly.

Submit 4 files:

1) makefile
2) arrayFunctions.h
3) arrayFunctions.c
4) pa4.c

Creating a Makefile

A makefile can simplify the task of compiling and linking multiple files. The default file name can be "makefile" or "Makefile".

Each segment of a makefile has this format:

target: dependencies (file(s) needed to produce this target)
$\xrightarrow[\text{Tab}]{}$ *optional* Linux command to execute

The default target is "all".

You must press the tab key before each Linux command. Spaces won't work!!!

The make command checks to see if any of the dependency files have been updated since the date/time on the target.

You should have only one makefile in a directory.

To execute the entire makefile, all you have to do is type

```
make
```

To execute only one target within a makefile,

make *targetname*

Here's an example makefile:

```
all: myprogram

myprogram: somefunctions.o myprogram.o
───→ gcc somefunctions.o myprogram.o -o myprogram
 Tab
myprogram.o: myprogram.c somefunctions.h
───→ gcc -c myprogram.c
 Tab
somefunctions.o: somefunctions.c somefunctions.h
───→ gcc -c somefunctions.c
 Tab
clean:
───→ rm -rf *.o
 Tab
```

If you just type

make

this makefile will start with the target called "all". "all" tells it to create "myprogram", so it looks for a target called "myprogram" to find the instructions. The target called "myprogram" says it needs "somefunctions.o" and "myprogram.o" so it's going to look for targets to find instructions for those.

The "clean" target is a quick way to remove the object files. They're not needed after the executable has been created.

The clean target is not automatically performed. To use this target, just type the command:

make clean