Lab 3 (Oct. 26-Nov. 6)

Pointers. Arrays of Strings. Dynamic Memory Allocation.

This lab is worth 3% of the course mark. The number in square brackets at the beginning of each question shows the number of points the question is worth. The total number of points is 30.

You will receive a mark only for the codes that are demonstrated in front of a TA by the end of the lab. With that in mind, please complete as many problems as you can before you come to lab. Work not completed by the end of the lab will not be marked. TAs will leave the lab at 5:20pm sharp.

General Requirements on Your Programs

A. Your programs should be written in a good programming style, including instructive comments and well-formatted, well-indented code. Use self-explanatory names for variables as much as possible. (~5% of the mark)
B. When outputting the results, include in the output an explanatory message. When inputting values display a message prompting the user to input the appropriate values. (~10% of the mark)

Lab Questions

1. [5] Write a program to read (from the keyboard) a positive integer n followed by a sequence of n floating point numbers. Store the numbers in an array allocated by invoking function calloc(). Then compute the average of all numbers, and the difference between the largest and smallest numbers by calling the function process(). The function's prototype has to be
   - void process(double y[ ], int m, double *avPtr, double *difPtr).

   The following statement can be used to call the function: process(x, n, &mean, &dif);
   where x is a pointer to the array (i.e., a pointer to the first element of the array), mean is the variable to store the average and dif is the variable to store the difference. After that print the values of mean and dif on the screen. **Note:** When using C standard library memory allocation functions use #include <stdlib.h>.

2. [5] Write a function with the prototype
   - int *sort(const int a[], int n, int *ptr)

   This function is passed an array of integers (a), its size (n) and a pointer to an integer variable. The function must allocate a new array and store in it all elements of a without repetitions and in increasing order. Finally, the function returns the address of the first element in the new array. Additionally, the function must store the size of the new array in the variable pointed to by ptr. The new array must be no larger than necessary in order to fulfill the requirements. On the other hand, during execution, the function is allowed to allocate a temporary array, if necessary. Write a program to test the function.

3. [5] Implement a string processing function with the prototype
   - char *my_strcat(const char * const str1, const char * const str2);

   The function creates a new string by concatenating str1 and str2. The function has to call malloc() or calloc() to allocate memory for the new string, i.e., for the total number of characters, plus the null character. The function returns the new string (i.e., the value of a pointer to the first array element, in other words, the address of the first array element). You are not allowed to call functions declared in the standard library header <string.h>, except for strlen() (strlen(s) returns the size of string s). After executing the following call to printf(): printf ("%s\n", my_strcat("Hello", "world!"));
   the printout on the screen has to be

Helloworld!
Write a program to test the function.

4. [5] Write a function with prototype
   - int largest_prefix(const char *str, const char **list, int n)

   where list represents an array of n strings (thus, list is a pointer to the first element in the array of strings), and str is another string. The function must find the largest string in list which is a prefix of str, and return the index of this string in list. If no string in list is a prefix of str, then the function must return -1. We make the convention that, for a string to be a prefix of another the case of letters must match. Write a program to test the function.
   Example: Assume that str is "university" and list consists of the following strings: "abc", "!$rt", "un", "city", "univ", "a", "Univer". Then "univ" (list[4]) is the largest prefix and the function has to return 4. (Note: "Univer" is not a prefix because of the uppercase "U").

5. [10] Write a program that reads a sequence of words from an input file and then writes them in **reversed order** in another file. For this exercise a word is a sequence of characters without white spaces. To read the data from the input file your program invokes the function read_words() with prototype
   - char **read_words(const char *input_filename, int *nPtr).

   Note that input_filename is a string representing the name of the input file. This function has to store the words into an array of strings. The memory for the array of strings and for each string has to be allocated dynamically (i.e., using malloc() or calloc()). Do not allocate more memory **then necessary for the array of strings and for the individual words. The input file contains a positive integer representing the number of words, on the first line.** Then the words follow one per line. Function read_words() has to store the number of words in the variable pointed to by nPtr. Additionally, the function returns a pointer to the beginning of the array of strings that was dynamically allocated.

   After invoking the function read_words() your program has to invoke the function output_words() to write the words in reversed order into another file. This file must contain the number of words on the first line, then the words one per line. This function's prototype is
   - void output_words(const char *output_filename, char ** sArray, int size),

   where output_filename is a string representing the name of the output file, sArray is a pointer to the array of strings (in other words, a pointer to the first element in the array) and size is the size of this array (i.e., the number of words).

   For this exercise you **are allowed** to use functions strlen() and strcpy() from the standard string processing library. Function strcpy() has the prototype
   - char * strcpy(char *s1, const char *s2)

   It copies the characters of string s2 into array s1 and appends '\0' at the end. It returns the value of s1. Pay attention to the fact that s1 must point to an array already existing in memory. strlen(s) returns the size of string s. **Note:** See Lab 2 notes for info on how to open a file, read and write in a file. See lecture notes for Topic 8 and 9 for info on arrays of strings and dynamic memory allocation. See Chapter 8 for more info on function strcpy().