CSE 410: Systems Programming

Recitation 3: Arrays and Pointers

Introduction

This lab will exercise the things you have learned about arrays and pointers, by asking you to implement some simple searches through arrays. You will also use file streams to input a string.

1 Getting Started

Follow the GitHub Classroom link that has been provided to you in Piazza and clone the repository that it creates for you. It will include some C source files and a Makefile. You will have to edit the source files to complete this recitation

2 Requirements

You must implement two functions, described here.

- `const char *cse_strchr(const char *string, char c)`

  The C standard library provides a function called `strchr`, and you will write an implementation of this function in the file `cse_strchr.c`. **You may not use strchr, or any other standard library string function, in your implementation.** This function is given a C string and a single character, and searches through the string for that character. If it is found before the terminating zero byte, a *pointer to the first such instance* is returned. Otherwise, it returns NULL. You can view the documentation for the C library function with the command `man strchr`.

  Note that there are some differences between `cse_strchr` and the C library `strchr`; in particular, the C library version carefully defines what happens on a search for `'0'`, while this version does not (and that will not be tested), and the C library version returns a non-const pointer, while this version does not. Both of these departures are made to simplify the implementation.

- `int arrayfind(const int *array, int size, int value, const int **found)`

  This function, which you must implement in the file `arrayfind.c`, provides similar functionality to the function `cse_strchr`, but on an array of integers. However, since it is desirable for an array of integers to hold any arbitrary values, defining a single value such as zero as the end of the array (such as in C strings) is not practical. Thus this function accepts the argument `size`, which specifies the number of elements in `array`. Your function should access *no more than `size` elements from `array`*.

  Furthermore, it uses an idiom often found in C APIs, which is that its *return value* is an integer indicating success or failure, and the value it computes is returned in a pointer argument. In C programs, it is typical for *a return value of zero to indicate success, and non-zero to indicate failure.*

  This function should search through the given array for an instance of a given value. If the value is found, it should store a *pointer to the value* into the argument `found`, and return zero. If the value is not found, it should leave `found` unchanged, and return a non-zero value.

In addition, you must use `fgets` and your implementation of `cse_strchr` to finish `ampersand.c`.

- This program must read one line of input (it may assume that the line is no more than 80 characters long) and then return 0 if an ampersand was found on that line, and 1 otherwise. It must not wait for any other input (that is, it should read one line, process it, and then exit). You must use `cse_strchr` to detect the presence of the ampersand.

2.1 Example Usages

An example use of cse_strchr (which could also be considered a test of the function!) is:

```
#include <stdio.h>

#include "arrayptr.h"

int main(int argc, char *argv[]) {
    const char str[] = "The quick brown fox jumped over the angry dog's back.";
    const char *c;

    c = cse_strchr(str, 'd');

    if (c != &str[25] || *c != 'd') {
        fprintf(stderr, "cse_strchr returned the wrong character!\n");
        return 1;
    }

    return 0;
}
```

An equivalent usage of arrayfind is only slightly more complicated. Notice the use of &found:

```
#include <stdio.h>

/* Don't worry about this yet */
#define LENGTH_OF_ARRAY(a)(sizeof(a) / sizeof(*a))

#include "arrayptr.h"

int main(int argc, char *argv[]) {
    const int array[] = { 0, 28, 42, 1170, 1337, 1401, 6221023, 9944100 };
    const int *found;
    int result;

    result = arrayfind(array, LENGTH_OF_ARRAY(array), 1170, &found);

    if (result != 0 || found != &array[3] || *found != 1170) {
        fprintf(stderr, "arrayfind returned the wrong element!\n");
        return 1;
    }

    return 0;
}
```

2.2 Testing

Some tests are provided, which will be run by `make test`. If `make test` emits only output regarding successful compilation commands and the following three lines, the tests were successful:

```
testing cse_strchr
testing arrayfind
testing ampersand
```

As there is limited time during recitation, you will probably not have a chance to write your own tests. The autograder does run somewhat more complete tests, but should provide helpful output if it finds an error that these tests did not.

2.3 Submission to Autograder

The command `make submission` will create the file `submission.tar` from the three files that you should edit for this lab. Submit that file, and nothing else, to Autograder for grading. Autograder will not accept any submission containing other files, or any submission for which those three files do not compile.

3 Grading

You will receive one point each for `cse_strchr`, `arrayfind`, and `ampersand`. This recitation will be graded on a partial-credit basis, so that, *e.g.*, completion of only two of the three deliverables will result in 2/3 credit.