COE 428 Lab 1

C Review

This is a one week lab. This lab must be submitted at least 48 hours before the beginning of your next lab period.

Prelab preparation

Before coming to the lab you should:

- Read the lab. Try to prepare any questions you may have about the lab.
- Refer to **Lab Guide**.
- Create your lab directory for lab1. (i.e. use **mkdir lab1** within your coe428 directory.)
- Change to your coe428/lab1 and unzip the lab1.zip file with the command:
  unzip /home/courses/coe428/lab1/lab1.zip

Introduction

This lab reviews more basic C programming. You will:

- Learn how to use a program development methodology that emphasizes incremental improvement and frequent testing.
- Use testing and debugging techniques.
- Use command line arguments.

Problem statement (Version 1)

Suppose that the requirements for a lab were:

1. A main() function in a file named sortMain.c uses a hardcoded array of integers, invokes a function called `mySort()` and prints the sorted values to *stdout*.
2. The main() function must invoke the sorting method as:

```
mySort(int data[], unsigned int n);
```

3. The source file sortMain.c must include the file mySort.h which contains:

```
/* DO NOT EDIT */

void mySort(int array[], unsigned int num_elements);
```

Furthermore, you **must** provide a `mySort()` function that conforms to this signature in a file named mySort.c.

Tutorial I (Initial stab at lab)

Clearly, we need three files: mySort.c, mySort.h and sortMain.c. Luckily, mySort.h has been furnished; as well, templates for sortMain.c and mySort.c are given.

There are two general approaches we could now take:

1. Write an implementation of mySort.c and modify sortMain.c to test it.
2. Modify sortMain.c so that it can test any implementation of the mySort.

We opt for the second approach.

What should main()do?

The main routine must:

- Declare and initialize the data to be sorted—i.e. an array of integers.
- Invoke the `mySort` function with the proper parameters: the data array name and the number of items to be sorted.
- Once the sorting function returns, it should print the sorted array to *stdout*.

While these are necessary requirements, the main() could do more; in particular,

- After sorting the array, it could check that the data really is sorted. If it is not correctly sorted, it should inform the user of what problem was encountered and exit with a non-zero exit code.

The initial sortMain.c module

An implementation of the sortMain.c module is shown below:

```
#include <stdio.h>

#include <stdlib.h>

#include "mysort.h"


int main(int argc, char * argv[])

{

    int data[100000]; /* Array of ints to sort */

    int nDataItems;   /* number of actual items in the array */
```

```
    int i;

    /* Test data */
    nDataItems = 4;
    data[0] = 10;
    data[1] = 20;
    data[2] = 30;
    data[3] = 40;

    mySort(data, nDataItems);

    /* Check that the data array is sorted. */
    for(i = 0; i < nDataItems-1; i++) {
        if (data[i] > data[i+1]) {
            fprintf(stderr, "Sort error: data[%d] (= %d)"
              " should be <= data[%d] (= %d)- -aborting\n",
              i, data[i], i+1, data[i+1]);
            exit(1);
        }
    }

    /* Print sorted array to stdout */
    for(i = 0; i < nDataItems; i++) {
        printf("%d\n", data[i]);
    }
    exit(0);
}
```

Edit, Compile, Link and Run the program

When you copied the *needed* files for this lab, you obtained a stub version of the source code file sortMain.c. Edit this file so that the source code is as shown above.

Compile the source code file with the commands:

```
gcc -c sortMain.c

gcc -c mySort.c
```

Now, link the object files and create the executable testSort with:

```
gcc -o testSort mySort.o sortMain.o
```

You can now run the command testSort. It should produce the following output:

```
10

20

30

40
```

Questions and answers

You may well have some questions about what has been done so far.

Question:

> Why write the test and output parts before doing the *real* work—writing the function to sort an array of numbers?

Answer:

It is best to try and develop software one piece at at time focusing your attention on a single problem. Here, although no work has been done on the sorting function, we do have a working main function. We know that it:

1. Invokes the sorting function.
2. Determines if the data really is sorted (and hence detects bugs in the sorting function.)
3. If no errors are detected, the sorted numbers are written to *stdout*.

Question:

It seems weird! The testSort command reports NO errors and the sort function hasn't been written!

Answer:

The reason no errors were detected is that the test data was set up so that it was **already** sorted. In fact, we are quite relieved that no error was reported!

Question:

Aha! I've got you. The test doesn't test anything since the data is fudged. So what's the point?

Answer:

As stated previously, at least we know the error detection code does not find sorting errors when there are none. However, as the question implies, the testing code has not yet been completely tested. This should be done.

Question:

How can that be done without writing the sort function?

Answer:

Simple. Just edit the main function and change one of the data items so that they are no longer initialized in sorted order.

Try it! Errors should now be reported and nothing should be written to *stdout*.

Question:

Why are the error messages written to *stderr* instead of *stdout*?

Answer:

There are at least two reasons:

1. The requirements for the program are that the sorted numbers should be written to *stdout*. Nothing else should be written there.
2. If the program is used and *stdout* is redirected, we still want the user to see the error messages. Redirection only affects the *stdout* stream; the *stderr* stream is not redirected and continues to be displayed on the screen.

Question:

OK, what's next?

Answer:

Well, now that we have a working and tested main function, it's time to implement the sorting function. The next section describes the requirements.

Requirement 1: implement mySort

The first requirement for the lab is to implement a sorting algorithm that respects the following specifications:

1. The source code for the implementation must be in a file named mySort.c.
2. A function called mySort (coded in the mySort.c file) that conforms to the signature in mySort.h will perform the sorting.

Warning

The specifications are precise. However, you are free to achieve them in any way you wish.

For example, although you must have a function called mySort in a file called mySort.c, nothing in the specs imply:

- The particular sorting algorithm to use.
- The use of other object modules in solving the problem.
- The inclusion of other functions (apart from mySort(...) in the source file mySort.c) is OK. (Frankly, I cannot see any simple reasons for doing this...but it is permitted...)

Tip

- Your mark for this portion of the lab depends on whether or not it works. This means (for example) that it is up to you to choose an algorithm for sorting that is relatively easy to implement.

You are even allowed to pattern your source code on an existing function that you find in a text book or on the web. However, you will learn more by trying to write the function by yourself from scratch.

Tutorial II: Command line arguments

In this tutorial you will learn how to obtain and use and command line arguments that were entered by the user when a command is invoked.

We now examine how the parameters argc and argv that are passed to the main function can be used. Consider the following standalone program (that is furnished to you in the file cmdlineArgsDemo.c):

```
#include <stdio.h>

#include <stdlib.h>

int main(int argc, char * argv[])

{

    int i;

    fprintf(stderr, "I was invoked with the command: %s\n", argv[0]);

    if (argc > 1) {

        fprintf(stderr, "The command line arguments are:\n");

        for(i = 1; i < argc; i++) {

            fprintf(stderr, "   argv[%d] (as string): %s\n"

                            "              (as int): %d\n"

                            "       (as int in hex): %X\n\n",

                            i, argv[i], atoi(argv[i]), atoi(argv[i]));

        }

    } else {

        fprintf(stderr, "There were no command line arguments.\n");

    }

    exit(0);

}
```

Create an executable from this source code file with the command:

```
gcc -o cmdlineDemo cmdlineArgsDemo.c
```

Invoke the command with:

```
cmdlineDemo
```

The output should be:

```
I was invoked with the command: cmdlineDemo
There were no command line arguments.
```

Now try the command:

```
cmdlineDemo hello 125 22 -6
```

The output should be:

```
I was invoked with the command: cmdlineDemo
The command line arguments are:
   argv[1] (as string): hello
               (as int): 0
            (as int in hex): 0

   argv[2] (as string): 125
               (as int): 125
            (as int in hex): 7D

   argv[3] (as string): 22
```

```
               (as int): 22
        (as int in hex): 16


argv[4] (as string): -6
               (as int): -6
        (as int in hex): FFFFFFFA
```

Requirement 2: Use argc and argv in main function

Copy your sortMain.c to sortMain2.c. Modify the new file so that the command works as follows:

- If no command line arguments are given, sort the hardcoded test data as before.
- If there are command line arguments, convert them to integers and use them values to initialize the data array to be sorted.

Requirements and Lab Submission

Complete both requirements described previously and answer the following questions.

Questions

Answer the following questions in a file named README.

1. Suppose you were given an object module (with no access to source code) that sorted an array of integers very efficiently. However, the sort function in the object module must be invoked with the following signature:

   ```
   betterSort(int data[], first, last);
   ```

   where the array to sort is data and the parameters first and last give the indices of the portion of the array that is to be sorted.

   How could you write a mySort() function with the signature used in this lab that could exploit the better sorting function in the supplied object module?

Submit your lab

1. Go to your **coe428** directory
2. Zip your **lab1** directory by using the following command:
   zip -r lab1.zip lab1/
3. Submit the lab1.zip file using the following command:
   submit coe428 lab1 lab1.zip

by Ken Clowes, revised by Olivia Das