COMP1511 18s2

Week-07 Laboratory Exercises

Objectives

- Introduction to struct, array of struct and pointers to struct
- Introduction string operations

Topics

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples. **We will only assess lab Exercises 01, 02 and 03 to derive marks for this lab.** However, you will learn a lot by attempting and solving the challenge exercises.

Getting Started

Create a new directory for this lab called `lab07` by typing:

```
$ mkdir lab07
```

Change to this directory by typing:

```
$ cd lab07
```

Exercise-00: Mandelbrot Art Gallery (individual)

This is an individual exercise to complete by yourself.

This exercise is due at the end of **week 8** (two weeks from now)

You've created your Mandelbrot server, you've got the technical side of things working... now it's time to actually **use** your code, and to exercise your aesethetic senses.

Find/create the most beautiful colored tile from the Mandelbrot set that you can. Submit your pixelColor.c file from the Mandelbrot activity AND also the file xanadu.h (sample below) which defines the x y and z coordinates for your tile.

```
/*
 * xanadu.h
 * Defines the centre of your beautiful colored tile image of the
 * Mandelbrot set.
 * Your name here:
 * Your description here:
 */

// Replace these three values with your own chosen numbers
#define CENTER_X 1.0
#define CENTER_Y 0.0
#define ZOOM 9
```

We'll regenerate the tile using your code, and display them in the MandelbrArt Gallery (coming soon to a course website near you!).

Don't forget to also submit your pixelColor.c, so that we can re-generate the tile using your color scheme.

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1511 wk07_mandelbrotBeauty xanadu.h pixelColor.c
```

You must run give before **Sunday 16 September 23:59:59** (end of **week 8**) to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Exercise-01: How many Orca Pods (pair)

This is a pair exercise to complete with your lab partner.

A citizen science project monitoring whale populations has files containing large numbers of whale observations. Each line in these files contains:

- the date the observation was made
- the number of whales in the pod
- the species of whales

Here is a file of example data which you can download to test your program. It looks like this:

```
$ head whales.txt
18/01/18  9 Pygmy right whale
01/09/17 21 Southern right whale
16/02/18  4 Striped dolphin
02/07/17  4 Common dolphin
19/02/18  4 Blue whale
21/02/18 38 Dwarf sperm whale
14/06/17 29 Southern right whale
20/06/17  3 Spinner dolphin
22/07/17 34 Indo-Pacific humpbacked dolphin
20/03/18  7 Long-finned pilot whale
```

Here is the starting code for this exercise.
It already contains the functions which you discussed in your tutorial to read the given file of whale sightings into an array of structs. You do not need need to change these functions.

The main function in **orca.c** has this code:

```
int n_orca_pods = count_orca_sightings(n_sightings, whale_sightings);
printf("%d Orca sightings in %s\n", n_orca_pods, argv[1]);
```

Your task in this exercise is to complete this function:

```
int count_orca_sightings(int n_sightings, struct pod sightings[n_sightings]) {
    // REPLACE THIS COMMENT WITH YOUR CODE
    // THIS FUNCTION SHOULD NOT CALL SCANF OR PRINTF
    // IT SHOULD JUST RETURN A VALUE
    return 42; // CHANGE ME
}
```

Do **not** change any other function.

It should return a count of the number of sightings of Orca pods in the file.

Note, "pod" is the collective number for a group of whales. Your function should return the number of Orca pods. It does not have have to sum the number of individual whales in these pods.

When you have completed the function **count_orca_sightings** this is how **orca.c** should behave:

```
$ dcc -o orca orca.c
$ ./orca whales.txt
53 Orca sightings in whales.txt
```

Hint: hint use `strcmp` from `string.h`
Hint: most of the work for this exercise is figuring out what to do - only a short loop containing an if statement is needed.

Hint: if you are having trouble understanding structs COMP1511 tutors Dean Wunder and Ben Pieters-Hawke take you through the basics in this video:

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest orca
```

When you are finished with this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk07_orca orca.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 09 September 23:59:59** to obtain the marks for this lab exercise.

Exercise-02: Counting A Whale Species (pair)

This is a pair exercise to complete with your lab partner.

Here is the starting code for this exercise.
It already contains the functions which you discussed in your tutorial to read the file of whale sightings into an array of structs. You do not need need to change these functions.

The main function in **species_count.c** has this code:

```
int pod_count;
int whale_count;
species_count(species, n_sightings, whale_sightings, &pod_count, &whale_count);
printf("%d %s pods containing %d whales in %s\n", pod_count, species, whale_count, filename);
```

Your task in this exercise is to complete this function:

```
void species_count(char species[], int n_sightings, struct pod sightings[n_sightings], int *n_pods, int *n_whales) {
    // REPLACE THIS COMMENT WITH YOUR CODE
    // THIS FUNCTION SHOULD NOT CALL SCANF OR PRINTF
    // IT SHOULD JUST ASSIGN VALUES to N_PODS AND N_WHALES
    *n_pods = 24; // CHANGE ME
    *n_whales = 42; // CHANGE ME
}
```

Do **not** change any other function.
species_count() should assign the number of pods of the given species to `*n_pods` and the total number of whales of the given species to `*n_whales`.

When you have completed the function **species_count** this is how **species_count.c** should behave:

```
$ dcc -o species_count species_count.c
$ ./species_count whales.txt "Blue whale"
51 Blue whale pods containing 1171 whales in whales.txt
$ ./species_count whales.txt "Indo-Pacific humpbacked dolphin"
43 Indo-Pacific humpbacked dolphin pods containing 897 whales in whale
s.txt
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest species_count
```

When you are finished with this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk07_species_count species_count.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 09 September 23:59:59** to obtain the marks for this lab exercise.

Exercise-03: Printing A summary of All Whale Species (pair)

This is a pair exercise to complete with your lab partner.

Here is the starting code for this exercise.
It already contains the functions which you discussed in your tutorial to read the file of whale sightings into an array of structs. You do not need need to change these functions.

The main function in **whale_summary.c** has this code:

```
whale_summary(n_sightings, whale_sightings);
```

Your task in this exercise is to complete this function:

```
void whale_summary(int n_sightings, struct pod sightings[n_sightings]) {

    // PUT YOUR CODE HERE

}
```

Do **not** change any other function.

whale_summary should **print** for every whale species how many pods were seen and how many total whales were seen of that species.

Match the example output form EXACTLY, except you may print the lines in any order.

When you have completed the function **whale_summary** this is how **whale_summary.c** should behave:

```
$ dcc -o whale_summary whale_summary.c
$ ./whale_summary whales.txt
59 Pygmy right whale pods containing 1260 whales
55 Southern right whale pods containing 1252 whales
68 Striped dolphin pods containing 1337 whales
62 Common dolphin pods containing 1232 whales
51 Blue whale pods containing 1171 whales
58 Dwarf sperm whale pods containing 1272 whales
52 Spinner dolphin pods containing 1118 whales
43 Indo-Pacific humpbacked dolphin pods containing 897 whales
50 Long-finned pilot whale pods containing 996 whales
68 Short-finned pilot whale pods containing 1344 whales
49 Dwarf minke whale pods containing 1080 whales
53 Orca pods containing 1245 whales
57 Pygmy sperm whale pods containing 1346 whales
55 Humpback whale pods containing 1071 whales
65 Fin whale pods containing 1251 whales
53 Coastal bottlenose dolphin pods containing 1169 whales
55 Sei whale pods containing 929 whales
47 Bryde's whale pods containing 1023 whales
```

You may assume there are at most 256 different whale species.

If **autotest** gives you an uninitialized variable error, you can reproduce this by compiling and running with `dcc --valgrind`, e.g.:

```
$ dcc --valgrind -o whale_summary whale_summary.c
$ ./whale_summary whales.txt
Runtime error: uninitialized variable accessed.

Execution stopped here in whale_summary(n_sightings=1000, sightings=0x1
ffee9fbf0) in whale_summary.c at line 113:
....
```

Beware the error may reported later than you expect.
Look back through your code for a variable, such as an array element, that has not been assigned an initial value before it is used.

Hint: if you are having trouble debugging your code watch COMP1511 tutor Peter Kydd explain how to add debugging printfs:

Hint: if you are having trouble with an uninitialized variable errors maybe Peter's explanation here will help:

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest whale_summary
```

When you are finished with this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk07_whale_summary whale_summary.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 09 September 23:59:59** to obtain the marks for this lab exercise.

Exercise-04: Correcting Typos in Whale Names (individual)[Challenge]

This is an individual exercise to complete by yourself.

The citizen scientists making whale observations have been careless entering whale names. They have often entered extra non-alphabetic characters or swapped upper case and lower case.

Here is an example of data they have entered you can download for this exercise.

We need a program to automatically correct the whale names they have entered.

Here is the starting code for this exercise.

It already contains the functions which you discussed in your tutorial which reads the file of whale sightings into an array of structs. You do not need need to change these functions.

You will notice the main function in **sanitize_whales.c** calls this function. The main function in **sanitize_whales.c** has this code:

```
sanitize_whales(species_names_file, n_sightings, whale_sightings);
```

Your task in this exercise is to complete this function:

```
void sanitize_whales(char species_names_file[], int n_sightings, struct pod sightings[n_sightings]) {
    // REPLACE THIS COMMENT WITH YOUR CODE
    // THIS FUNCTION SHOULD NOT CALL SCANF OR PRINTF
    // IT SHOULD CHANGE THE STRUCTS IN THE ARRAY SIGHTINGS
}
```

The **species_names_file** will be the name of a file with the correct whale name one per line.
Here is an example whale names file you can download for this exercise.

sanitize_whales should delete any extra non-alphabetic characters in the whale name and it should correct the case of the whale name.

When you have completed the function **sanitize_whales** this is how **sanitize_whales.c** should behave:

```
$ dcc -o sanitize_whales sanitize_whales.c
$ cat unsanitized_whales.txt
18/01/18  9 Pygmy  right whale
01/09/17 21  Southern       right whale
16/02/18  4 Striped dolphin
02/07/17  4 common dolphin
19/02/18  4 BLUE WHALE
21/02/18 38 Dwarf-sperm-whale
14/06/17 29    Southern64376537653right76532765353whale
20/06/17  3 Spinner35879875dolphin84372979324832978
22/07/17 34 indoPacifichumpbackeddolphin
20/03/18  7 ~`,.?<>:;"'[]Long-finned~`,.?<>:;"'[]pilot~`,.?<>:;"'[]whal
e
11/03/18 19        ____short       ____finned       ____PILOT       ___
_WHALE
24/09/17  5 S   outh----ern ri'ght wH78ale
$ ./sanitize_whales whale_names.txt unsanitized_whales.txt corrected.txt
$ cat corrected.txt
18/01/18  9 Pygmy right whale
01/09/17 21 Southern right whale
16/02/18  4 Striped dolphin
02/07/17  4 Common dolphin
19/02/18  4 Blue whale
21/02/18 38 Dwarf sperm whale
14/06/17 29 Southern right whale
20/06/17  3 Spinner dolphin
22/07/17 34 Indo-Pacific humpbacked dolphin
20/03/18  7 Long-finned pilot whale
11/03/18 19 Short-finned pilot whale
24/09/17  5 Southern right whale
```

You may assume there are at most 256 different whale species.

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest sanitize_whales
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1511 wk07_sanitize_whales sanitize_whales.c
```

Submission

When you are finished with each exercise make sure you submit your work by running **give**.
You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Sunday 09 September 23:59:59** to submit your work.

COMP1511 18s2: Programming Fundamentals is brought to you by
the School of Computer Science and Engineering at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs1511@cse.unsw.edu.au
CRICOS Provider 00098G