COMP1511 18s2

Week-08 Tutorial Exercises

1. The tutorial will start with a code review.
   Discuss the good, the bad and the ugly aspects of their code.

   Please be gentle in any criticism - we are all learning!

2. Have you submitted your first assignment?
   Have you learnt anything you think would be useful to share with the tutorial?

3. If your tutorial is on Monday, your tutor will explain the following.

   a. My program works when I run it, **autotest** says I have an uninitialized variable, how can it work with an uninitialized variable.
   b. How does **dcc** help you find errors?
   c. How does **dcc --valgrind** help you find errors?
   d. Why do autotests and automarking run both **dcc** and **dcc --valgrind**?
   e. What is **dcc --leakcheck**?

4. Last week's lab used an array of this struct to store a whale sighting:

```
#define MAX_SPECIES_NAME_LENGTH 128
struct pod {
    struct date when;
    int         how_many;
    char        species[MAX_SPECIES_NAME_LENGTH];
};
```

   This week's lab will use a linked list of this struct to store a whale sighting

```
struct pod {
    struct pod  *next;
    struct date *when;
    int         how_many;
    char        *species;
};
```

   A date is still represented by the same struct:

```
struct date {
    int year;
    int month;
    int day;
};
```

   Sketch out how these two representation would be laid out in memory.
   Discuss the differences and how that will affect this week's lab exercises.

5. a. Assume you have a function with the following prototype:

```
void write_sighting(FILE *f, struct pod *p);
```

   which prints the details of the sighting point to by **p** to stream **f**
   Write a function with this prototype:

```
void write_sightings_file(char filename[], struct pod *first_pod);
```

   which opens the file it is given as argument and calls **write_sighting** to print the details of all the pod structs in the linked list it is given.

   b. (Optional) : Write a function with this prototype:

```
void write_date(FILE *f, struct date *d);
```

   which prints details of a **date** struct to stream **f**.

   c. Write a function with this prototype:

```
void write_sighting(FILE *f, struct pod *p);
```

   which prints details of a sighting (**pod** struct) to stream **f**.

6. a. Assume you have a function with the following prototype:

```
struct pod *read_sighting(FILE *stream)
```

which mallocs a **pod** struct and assigns values to its fields, which are read from a line of the stream that it is given as an argument.
Write a function with this prototype:

```
struct pod *read_sightings_file(char filename[])
```

which opens the file it is given as argument and calls **read_sighting** to create pod structs containing the information in the file.
read_sightings_file should return these pod structs as a linked list.

b. (Optional) : Write a function with this prototype:

```
struct date *read_date(FILE *f);
```

which mallocs a **date** struct and assigns values to its fields, from values read from stream **f**.

c. Write a function with this prototype:

```
struct pod *read_sighting(FILE *f);
```

which mallocs a **pod struct** and assigns values to its fields read from stream **f**.

7. Discuss the remainder code which you are given for the lab exercises and what you have to do for the standard lab exercises.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_SPECIES_NAME_LENGTH 4096

// a struct to represent the date
// a whale pod sighting was made

struct date {
    int year;
    int month;
    int day;
};

// a struct to represent a sighting
// of a pod (group) of whales

struct pod {
    struct pod  *next;
    struct date *when;
    int         how_many;
    char        *species;
};


struct pod *read_sightings_file(char filename[]);
struct pod *read_sighting(FILE *f);
struct date *read_date(FILE *f);

int count_orca_sightings(struct pod *first_pod);

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <file>\n", argv[0]);
        return 1;
    }

    struct pod *first_pod = read_sightings_file(argv[1]);

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

    return 0;
}


// return the number of sightings of Orca

int count_orca_sightings(struct pod *first_pod) {
    // REPLACE THIS COMMENT WITH YOUR CODE
    // YOU ARE NOT PERMITTED TO USE ARRAYS
    // THIS FUNCTION SHOULD NOT CALL SCANF OR PRINTF
    // IT SHOULD JUST RETURN A VALUE
    return 42; // CHANGE ME
}


//
// DO NOT CHANGE THE FUNCTIONS BELOW HERE
//

// return linked list of sightings read from filename
// exit called if there is an error

struct pod *read_sightings_file(char filename[]) {
    FILE *f = fopen(filename, "r");
    if (f == NULL) {
        fprintf(stderr,"error: file '%s' can not open\n", filename);
        exit(1);
    }
```

```
    struct pod *first_sighting = NULL;
    struct pod *last_sighting = NULL;

    struct pod *sighting = read_sighting(f);
    while (sighting != NULL) {
        if (first_sighting == NULL) {
            first_sighting = sighting;
            first_sighting->next = NULL;
        } else {
            last_sighting->next = sighting;
        }
        last_sighting = sighting;
        sighting = read_sighting(f);
    }

    return first_sighting;
}


// read a whale sighting (date, number of whales, whale species)
// return a pointer to a malloced struct containing these details
// return NULL if a sighting can not be read

struct pod *read_sighting(FILE *f) {
    struct pod *p = malloc(sizeof (struct pod));
    if (p == NULL) {
        fprintf(stderr, "out of memory\n");
        exit(1);
    }

    p->next = NULL;

    p->when = read_date(f);
    if (p->when == NULL) {
        free(p);
        return NULL;
    }

    int n_scanned = fscanf(f, "%d", &(p->how_many));
    if (n_scanned != 1) {
        free(p);
        return NULL;
    }

    fgetc(f);
    char species_buffer[MAX_SPECIES_NAME_LENGTH];
    if (fgets(species_buffer, MAX_SPECIES_NAME_LENGTH, f) == NULL) {
        free(p);
        return NULL;
    }
    // finish string at '\n' if there is one
    char *newline_ptr = strchr(species_buffer, '\n');
    if (newline_ptr != NULL) {
        *newline_ptr = '\0';
    }

    // also finish string at '\r' if there is one - files from Windows  will
    newline_ptr = strchr(species_buffer, '\r');
    if (newline_ptr != NULL) {
        *newline_ptr = '\0';
    }

    // malloc a char array long enough to hold species name
    // and copy species to it
    p->species = malloc(strlen(species_buffer) + 1);
    if (p->species == NULL) {
        fprintf(stderr, "out of memory\n");
        exit(1);
    }
    strcpy(p->species, species_buffer);
```

```
    return p;
}


// read a date in year/month/day format from stream f
// return a pointer to a malloced date struct containing them
//  return NULL if a date can not be read

struct date *read_date(FILE *f) {
    struct date *d = malloc(sizeof (struct date));
    if (d == NULL) {
        fprintf(stderr, "out of memory\n");
        exit(1);
    }
    int n_scanned = fscanf(f, "%d/%d/%d", &(d->year), &(d->month), &(d->day));
    if (n_scanned != 3) {
        free(d);
        return NULL;
    }
    return d;
}
```

Revision questions

The remaining tutorial questions are primarily intended for revision - either this week or later in session.
Your tutor may still choose to cover some of the questions time permitting.

8.
- Write a C function `split_string` which takes a string read by fgets containing substrings separated by commas, places the substrings in an char[][] array and returns the number of strings.
  You can assume there are at most 128 words on the line and no word is more than 32 characters long.
- Write a C function `print_substrings` which prints one per line the substrings in the char[][] array created in the previous question.
- Write a C function `substrings_sorted` which given the char[][] array in the previous question returns 1 if the substrings are increasing order and 0 otherwise.
- Write a C function `search_substrings` which given the char[][] array in the previous question returns 1 if the array contains a specified string and 0 otherwise.
- Write a program `substring.c` which reads lines using fgets and calls the above functions.

9. Write a function that is given two `struct date` pointers and compares returns 0 if they are equal, -1 if the first is less than the second and 1 if the first is larger than the second.

```
int compare_date(struct date *d1, struct date *d2);
```