CS 24000: Programming in C
Final Exam
Fall 2018

Name:

Username:

Read all instructions before beginning the exam.

- This is a closed book examination. No material other than those provided for you are allowed.
- You need only a pencil and eraser for this examination. If you use ink, use either black or blue ink. If you use pencil, your writing must be dark and clearly visible.
- This examination contains an amount of material that a well-prepared student should be able to complete in two hours.
- This examination is worth a total of 165 points. Not all questions are worth the same amount. Plan your time accordingly.
- Write legibly. You should try to adhere to the course code standard when writing your solution(s). Egregious violations may result in point deductions.
- You may leave after you have turned in all pages of the examination booklet. You will not be able to change any answers after turning in your examination booklet.
- Read each question *carefully* and *only do what is specifically asked for* in that problem.
- Some problems require several steps. Show all your work. Partial credit can only be rewarded to work shown.
- Do not attempt to look at other students' work. Keep your answers to yourself. Any violation will be considered academic dishonesty.
- Write your username on *EVERY* page where indicated. Any page without a username will receive a zero for the material on that page.
- Use appropriate assertion checks for *all* problems.
- Read and sign the statement below. Wait for instructions to start the examination before continuing to the next page.

"I signify that the answers provided for this examination are my own and that I have not received any assistance from other students nor given any assistance to other students."

Signature:

- Do not open the examination booklet until instructed.

1. (35 points) Provide a short answer to each of the following questions.

   (a) (5 points) Write a function called `serial_number()` that returns 5000 the first time it is called, 5001 the second time it is called, 5002 the third time it is called, and so on. For example:

   ```
   x = serial_number();   /* x set to 5000 */
   x = serial_number();   /* x set to 5001 */
   x = serial_number();   /* x set to 5002 */
   x = serial_number();   /* x set to 5003 */
   x = serial_number();   /* x set to 5004 */
   ```

   ...and so on.

   `serial_number()` has no arguments. **Note:** Do not use any global variables.

   (b) (3 points) Suppose you are writing a C file that refers to a global variable defined as `int balloons`. This variable, however, is defined in another module. How should you declare this variable in *your* module so that you can refer to it but not duplicate its definition?

   (c) (3 points) Assuming that `slgp` is a pointer to `struct lawn_gnome` and `sgp` is a pointer to `struct gnome`, assign `slgp` to `sgp` so that there is no compile warning. (i.e., use a cast).

   (d) (5 points) Write a macro `MIN()` that takes two variable arguments (of any first class type) and determines the minimum of the two. Use the ternary operator. Do not forget to use parentheses correctly.

(e) (3 points) Given three object files: `myobj1.o`, `myobj2.o`, and `myobj3.o`, write the command to generate a library named "`my_lib`" (the filename should be `libmy_lib.a` or `libmy_lib.so` depending on your approach).

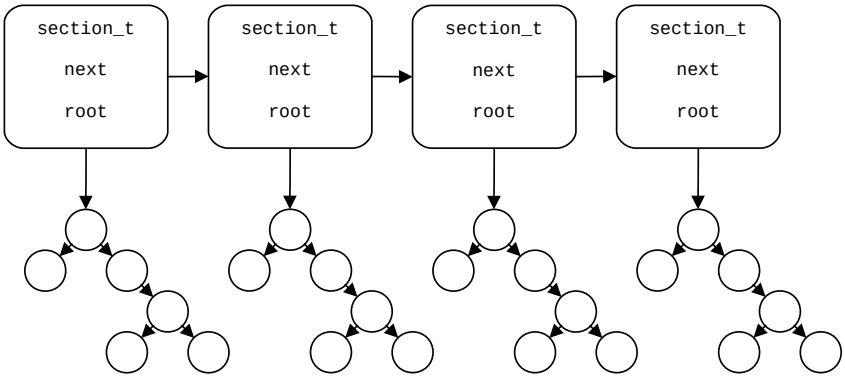
(f) (3 points) What are the two additional flags that must be passed to `gcc` to link against the library created above, assuming that it is located in the current working directory and that the current working directory is not in the library search path?

(g) (12 points) Write a function, `nco()` that accepts an `unsigned int` and returns an `int` indicating the maximum number of consecutive ones in the binary representation of the argument. So, for instance, if the value 6,521 (110010**1111**001 in binary) were passed to the function, it would return 4.

The remaining parts of this exam are based on a particular project: a telephone database. You will be writing functions that are needed to build, access, and delete this database. We provide the following definitions:

```
/* constant definitions */
#define MAX_NAME_SIZE    (512)   /* max name length */
#define MAX_ADDRESS_SIZE (1024)  /* maximum address length */
```

The telephone database is laid out in the following manner:



As you can see, each section is a node in a singly-linked list. For a given section, there is a binary tree containing all telephone entries that belong in that section.

2. (20 points) Structures

   The following questions deal with structures for this telephone database. Once declared, these structures are assumed to exist for the remainder of the exam. Assume for all string arrays that the string (including the `'\0'` will not be longer than its corresponding `MAX...SIZE`.

   (a) (10 points) (Tree) Write the declaration for a structure type called `entry_t` that contains (in this order) two pointers to `char` called `name` and `address`, a variable named `phone` of type `int`, and two pointers to the structure being declared—named `right` and `left`. Use `typedef` for this type declaration.

   (b) (10 points) (Singly-linked list) Write the declaration for a structure type called `section_t` that contains (in this order) a pointer to `char` called `name`, two integers named `start_page` and `end_page`, a pointer to the structure being declared—named `next`, and a pointer to `entry_t` named `root`. Use `typedef` for this type declaration.

3. (60 points) Dynamic memory

   (a) (10 points) Define a function, `create_entry()`, that accepts three arguments, two a pointer to `char`: a `name` and an `address`, and one an `int`: the `phone_number`. It should return a pointer to a newly allocated `entry_t` containing a copy of all data. Be sure to properly initialize *all* structure fields!

(b) (15 points) Define a function, `insert_section()`, that accepts two arguments: a pointer to a pointer to `section_t` (the `head` of a singly-linked list), and a pointer to `section_t` (a `new` element to be added). Insert the second argument into the potentially empty singly-linked list in *alphabetical order* by the `name` field. The function's return type should be `void`.

(c) (15 points) Define a function, `insert_entry()`, that accepts three arguments: a pointer to a pointer to `entry_t` (the `root` of a sorted binary tree), a pointer to `entry_t` (a `new` node to be added), and a pointer to a `compare` function that returns an `int` and accepts two arguments—each a pointer to an `entry_t`. Insert the second argument into the potentially empty binary tree using the `compare` function pointed to by the third argument. The compare function is defined as returning -1 if the first argument comes before the second, 0 if they are equal, and 1 if the first falls after the second. `insert_entry()`'s return type should be `void`.

(d) (10 points) Define a function, `add_entry()`, that accepts three arguments and returns an `int` indicating success or failure. The three arguments consist of a pointer to `section_t` (the `head` of the section list, a pointer to `entry_t` (the `new` telephone entry to be added), and an `int` indicating on which page the entry is found. Locate the correct section in the list and insert the new entry into the corresponding binary tree. Use the `insert_entry()` function described above. Return -1 if there is no section containing the page range in which the new entry falls, otherwise return 0. Assume that there is a previously defined comparison function named `my_compare()` to pass to `insert_entry()`.

(e) (10 points) Define a function, `free_entry()` that accepts a single argument—a pointer to a pointer to `entry_t` and returns `void`. If the pointer to `entry_t` is not `NULL`, you should de-allocate all associated memory and set the pointer to `NULL`.

4. (50 points) File I/O

   (a) (25 points) Write a function, `create_book()`, that accepts a single argument—a pointer to `char` (the `name` of the input file), and returns a pointer to `section_t` (the `head` of a newly allocated phone book). The file should contain data in the following format:

```
<Book Title>
<Section Name> pp. <StartPage>-<EndPage>
...
<PageNumber> <Name>; <Address>; <PhoneNumber>
...
```

   A "real" example follows:

```
TurkeyLand Phonebook
Mechanics pp. 1-10
Randomness pp. 11-45
More_Crap pp. 46-75
23 Joseph Smith; 1838 Bob Ave., Honolulu, HI  90210; 4088425993
54 Mike Williams; 465 Northwestern Ave., West Lafayette, IN  47906; 7654942344
7 Jeff Turkstra; 27 Hilltop Dr. Apt. 7, West Lafayette, IN  47906; 7654959258
```

   Section names may *not* contain spaces.

   Open the data file and read the data into the previously described data structures. For each section, allocate a new linked list node and insert it into the (initially empty) singly-linked list. For each entry, allocate a new tree node and insert it into the appropriate binary tree. Upon completion, return a pointer to the head of the newly created linked list.

   HINT: To detect the transition between the list of sections and the list of entries, check the return value of `fscanf()`. Also, use `ftell()` to keep track of the beginning of the previous line. When you encounter the transition, `fseek()` back to the beginning of the line and use the correct `fscanf()` call to start reading the entries.

   Return `NULL` in the event of any error (improper file format, unreadable file, etc).

   Use the `create_()` and `insert_()` functions described on previous pages.

   Begin writing the function on the next page...

Additional space on next page...

Work area for problem 4.a...

(b) (15 points) Define a function, `write_entries()`, that accepts two arguments—a pointer to `entry_t` (the `root` of a binary tree), and a file pointer that points to a "file" that *has already been opened*. It should return an `int` indicating the number of entries written or -1 in the event of an error. Traverse the tree "least to greatest" and write each entry to the file in the following format:

```
<Name> <Address> <Phone Number>
```

For example:

```
Joseph Smith 1838 Bob Ave., Honolulu, HI  90210 408-842-5993
Jeff Turkstra 27 Hilltop Dr. Apt. 7, West Lafayette, IN  47906 765-495-9258
```

(c) (10 points) Define a function, `write_book()`, that accepts two arguments—a pointer to `section_t` (the `head` of a singly-linked list), and a pointer to `char` that points to the `filename` to which all appropriate data should be written. It should return an `int` indicating the number of entries written or -1 in the event of an error. Traverse the linked list and respective binary trees "least to greatest" and write each entry to the file in the following format:

```
<Section Name>
<Name> <Address> <Phone Number>
<Name> <Address> <Phone Number>
...
<Section Name>
<Name> <Address> <Phone Number>
<Name> <Address> <Phone Number>
...
```

Use `write_entries()` as described on the previous page.