CISC105 Spring 2006 Lab12

- Solve each of the following problems. Be sure to save every separate program. All programs must be properly commented and indented (see Assignment Standards on the class website).
- Functions must be commented, but the names of functions and the names of parameters should clearly indicate their purpose. **Comments are explanatory, names are informational.**
- Every malloc() or calloc() must be followed by a call to free(). Calls to free() are cheap and good style.
- If you completed lab 11, then you know a lot about programming already. Pat yourself on the back. You are not an expert yet, but you have the foundation you need to go out and learn *any* programming language and become an expert.
- Name each program after rabbits from *Watership Down.*

Programs

1. Are you ready for the final? Try this without looking at your notes: Read in one string from the command line, determine its length, then allocate just enough space to hold that string and copy it into the new space. Print the string from the new space.

   If you need your notes, use them. Then try to do it later without.

2. The argument vector in main, `argv`, is declared in main's prototype as `char * argv[]`. This means `argv` is an array of type char* (read as "char pointer" or "pointer to char"). We cannot put character data inside a pointer; we have to allocate memory first, and then use the char* to put data into the memory. In the case of `argv`, the OS set up the necessary memory when the program was called[1].

   Declare your own array of five `char*`. Read in five words from the user (standard input, or stdin) and the five elements of your array point to new space that is the size required for the words you read in. Print.

3. Declare the following:

   ```
   struct dog * sdPtr;
   ```

   and set it equal to new space that is the size of a single struct dog. Now use the indirect selection operator to put data in your dog. Print your dog using sdPtr.

4. Write a function to print a struct dog when it is passed a struct dog *, and test with the program from number 3.

5. Declare the following:

   ```
   struct node {
     int data;
     struct node * next;
   };
   ```

[1] How do you know that the compiler could not have set aside the necessary memory?

Note that you have *not* declared a struct node inside a struct node. What have you done?

Declare three struct nodes, name them one, two, and seven[2], and put integers in their "data" components. Now that you have created them, make the first struct node point to the second as follows:

```
one.next = &two; //what exactly is this doing?
```

Then try to print the second struct *without using its name:*

```
printf("%d\n", one.next->data);
```

Link the second and third structs similarly. Can you use operators to print all three structs while only using the name of the first?

6. The only thing between you and one of the most powerful ideas in computer programming is a few calls to malloc. Change the previous program so that you malloc the three structs (now you will only use the indirect selection operator!). Once you get the program working again (this can be tricky, since the syntax is very new to you), jump up and cheer, and then send a note to your proud instructor. You have just made your first "linked list", something that holds an arbitrary number of data items. Variations on this idea underly most of the code in compilers and the Unix file system.

You should have a total of 5 programs named lab12.1.c to lab12.5.c. Make a single script file, but don't print it out unless you really hate trees.

Submit all code and script files to /dev/null (ask your TA for assistance). Note: The final is less than two weeks away. What would you like to know about C before then? Be sure to ask your instructor or TA.

[2]Do you think blindly following this instruction is critical to your program?