University of Colorado at Colorado Springs
Operating Systems Project 1
C Programming and `Makefile`

Instructor: Serena Sullivan
Total Points: 100
Out:
Due: 11:59 pm

Introduction

The purpose of this project is to refresh your C programming skills, using command line, and writing a `Makefile`.

You must do this assignment by yourself and members of your team. You may discuss the problem or solutions with your classmates or instructor, but **you may not share code with anyone**. You may not use code downloaded from the Internet. Please read all instructions carefully.

Project submission

For each project, please create **a zip file containing the following items**, and submit it to Canvas.

1. A report called `README.txt` that includes (1) the (printed) full names of the project members, and the statement: We have neither given nor received unauthorized assistance on this work; (2) the name of your virtual machine (VM), and the password for the account (regular user or root account) that can successfully run your code; and (3) a brief description about how you solved the problems and what you learned (more details on page 4). The report can be in txt, doc, or pdf format. Please use the template provided in Canvas Modules → Template → writeup-template.docx.

2. Your source code and `Makefile`; **please do not include compiled output**. Even though you have your code in your VM, submitting code in Canvas will provide a backup if we have issues accessing your VM.

Project

Part 0: Preparation

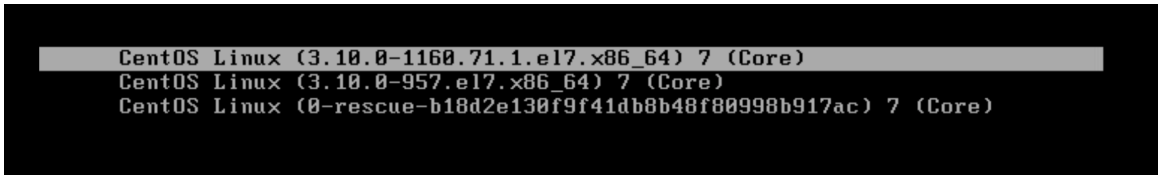
First, Update and Reboot

After you successfully installed CentOS 7, and logged into your new VM, first run the following command as **root** (note that the `#` prompt indicates that the command requires root privilege. `$` indicates regular user privilege):

```
# yum update -y
```

If you don't include the "-y" option, then make sure to read the messages on the screen. If it asks something like "Total size 747M, is this ok [y/d/N]?" Make sure to type "y" and hit enter. If you just hit enter, it won't update at all ¯_(ツ)_/¯ When it's updating, it will take a while so feel free to take a coffee break

After update is complete, please reboot the VM by typing `reboot` as root (or click the UI to reboot), and log into the newer version. You will see a new option at boot up (you will only see it after `yum update` and `reboot`), and go ahead to select the new one (should be chosen by default):



Once logged in, you can verify the kernel version using the following command (either as root or normal user):

```
$ uname -r
```

You should see the exact string `3.10.0-1160.71.1.el7.x86_64` as the output.

Second, Install Tools

Install some development tools like the compiler `gcc` by running this command as **root**:

```
# yum install -y gcc ncurses-devel make wget perl
```

Now your environment should be ready to go. Before going forward, please find the following resources that you may find useful:

1. If you need some help with C, you should definitely check out `http://cslibrary.stanford.edu/` especially "Essential C", "Pointers and Memory", "Linked List Basics", and "Linked List Problems". If you need help with using *nix for software development then you should go through the "Unix Programming Tools".

2. Some helpful C functions are `strlen`, `strncmp`, `printf`, `strncpy`, `strdup`, `malloc`, `free` (`man` these functions on command line or Google if necessary. Pay attention to return values.)

3. `Makefile` tutorial: `https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/`

Part 1: Writing C Code (60 points)

Create a linked list data structure in C, where each node in the list stores a character string called `item`, and a pointer called `next` pointing to the next node in the list (line 2 – 7 in the code below). Use a header file (`list.h`) and separate source file (`list.c`) for all functions, and make a test file `list_test.c` that tests your list implementation.

You need to implement the functions in the snippet below, as well as define the struct(s) necessary for your list to work correctly.

1. **Please do not use arrays, and do not declare arrays.**
2. **Please do not use functions from the C++ Standard Template Library, i.e., you should implement the following functions by yourself. Do not use `cin` or `cout`.**

```
1  /* Declaring all the structs */
2  typedef struct Node node;
3
4  struct Node {
5      char *item;
6      node *next;
7  };
8
9  typedef struct List {
10     node *head;
11 } list;
12
13 /* Allocate space for a new list and set its head to NULL.
14  * Returns the created list if successful, NULL otherwise. */
15 list* create_list( );
16
17 /* Allocates a new node and copies the string from item to this node
18  * (use malloc, strlen, and strncpy; or try  strdup). Adds this new node
19  * to end of the list ll. Returns 0 if successful, non-zero otherwise. */
20 int add_to_list(list* ll, char* item);
21
22 /* Removes the head of the list ll, extracts the string stored in the head,
23  * and returns a pointer to this string. Also frees the removed head node. */
24 char* remove_from_list(list* ll);
25
26 /* Prints every string in each node of the list ll, with a new line
27  * character at the end of each string */
28 void print_list(list *ll);
29
30 /* Flushes (clears) the entire list and re-initializes the list. The passed
31  * pointer ll should still point to a valid, empty list when this function
32  * returns. Any memory allocated to store nodes in the list should be freed.
33  */
34 void flush_list(list* ll);
35
36 /* De-allocates all data for the list. Ensure all memory allocated for list
37  * ll is freed, including any allocated strings and list ll itself. */
38 void free_list(list *ll);
```

Requirements: Please place any `struct` definitions, `typedef` and the above function prototypes in a header file `list.h` and function bodies (implementations) in a source file

`list.c`. Your source code must use Unix style end-of-line (EOL) characters `\n`, not DOS style `\n\r`. You could write code on Windows (I don't recommend), but you will have to convert it to Unix text files before submitting. The `dos2unix` command may be useful for this.

Make a test file `list_test.c` that tests your list implementation. Your test file should try various combinations of your `create_list`, `add_to_list`, `remove_from_list`, `flush_list`, `print_list`, and `free_list` functions. Note the difference between `flush_list` and `free_list`: after calling `flush_list`, your list `ll` still points to a valid, empty list (like a brand new list whose head points to NULL); after calling `free_list`, `ll` itself becomes NULL. You should check return values on any function that can possibly fail.

Part 2: `Makefile` and `README` (40 points)

`Makefile` (30 points)

Write a `Makefile` that compiles `list.c` and `list_test.c` to create a binary executable named `list_test`. A simple `Makefile` tutorial can be found here (Reading through `Makefile` 4 is enough): `https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/` Make sure you read it otherwise you won't magically know how to write a `Makefile` ¯_(ツ)_/¯

If your project does not make with your `Makefile`, sorry we won't fix it for you or grade it. Your program should compile without any warnings or errors when using all standard `gcc` warnings, i.e., `CFLAGS` should include `-Wall` in your `Makefile`.

`README.txt` (10 points)

Write a plain text (i.e., not WYSIWYG/Word processed) `README.txt` file that explains how to build and run your program, gives a brief description of the pieces of your assignment, identifies any challenges you overcame, and contains any notes about resources you used or discussions you had with others. **I suggest starting work on your assignment with this file and maintaining it as you work.**

Word of Advice

COMMENT YOUR CODE. If something doesn't work, you will get partial credit if your comments show that you were on the right track. Document any places where you received help from others.

Important elements of the grading will be the quality and coverage (number of cases) that are tested and the cleanliness of your implementation and documentation, and (last but not the least) for correctness and efficiency.

Some VM Tips (Below Optional)

While you are logged in, from within the VM:

1. Commands to switch between user accounts: `su`, `whoami`, `exit`;
2. Command to find the IP address: `ifconfig` (look for IP that looks like 128.198.51.xx);
3. Command shortcuts: ctrl+R for reverse search for a command that you've ran under the same user account (ctrl+R, then type in a few characters that are unique to the command you want to search); ctrl+D for exiting the VM from ssh.

While you are on your own laptop or desktop machine: A few tricks to make `ssh` convenient.

1. First, the `ssh` config file: In Linux and Mac, it's located at `your_home_directory/.ssh/`, a hidden directory (if you don't have it, just '`mkdir .ssh`).

2. The file has to be named `config`, and formatted as follows (if you don't have it, just `vim config`). It contains multiple entries for all the machines that you would like to ssh into. Here is an entry for my VM as an example:

   ```
   Host centos
   HostName 128.198.51.75
   User ssulliv7
   ```

   You'll need to replace the values of Host, HostName, and User with the values that work for you. Then when you ssh, instead of `ssh yzhuang@128.198.51.75`, you just need to do `ssh centos`. You can add as many entries as you like if you have multiple remote hosts.

3. On Windows, you can install openSSH, Putty, or other types of ssh client, and set it up in a very similar way (`https://www.youtube.com/watch?v=GEA4t1Z1OP4`).

4. If you would like to setup **password-less** ssh, so you don't need to type password every time you log in, please read this: `http://www.linuxproblem.org/art_9.html`