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

Instructor: Serena Sullivan
Total Points: 100
Out: 8/26/2024
Due: 11:59 pm, 9/26/2024

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 if you have a team). You may discuss the problem or solutions with your classmates or instructor, but **you may not share code with anyone**. You may not directly use code from the Internet. Please read all instructions carefully, and **don't wait till last minute to work on projects**.

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 directory and name of your virtual machine (VM), the path in VM where your code is located, 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).
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 your VM and logged in, first run the following command with **sudo**:

```
$ sudo apt update && sudo apt upgrade
```

When the screen prompts "After this operation, XXX MB of additional disk space will be used. Do you want to continue? [Y/n]" please type "y" and hit enter. It will take a while to update so feel free to take a  break. After the update is complete, please reboot the VM by typing `reboot` or click the UI to reboot.

```
$ sudo reboot
```

If you are using `ssh`, at this time the connection will be closed. Please re-login once the reboot is complete.

Second, Install Tools

Install some development tools like the compiler `gcc` and `make`:

```
$ sudo apt-get install make gcc
```

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 all the functions listed in the snippet below, as well as define the struct(s) necessary for your list to work correctly. Note:

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`.**

3. **You will lose points if you change the struct or function definition shown below.**

```c
/* Declaring all the structs */
typedef struct Node node;

struct Node {
    char *item;
    node *next;
};

typedef struct List {
    node *head;
} list;

/* Allocate space for a new list and set its head to NULL.
 * Returns the created list if successful, NULL otherwise. */
list* create_list( );

/* Allocates a new node and copies the string from item to this node
 * (use malloc, strlen, and strncpy; or try strdup). Adds this new node
 * to end of the list ll. Returns 0 if successful, non-zero otherwise. */
int add_to_list(list* ll, char* item);

/* Removes the head of the list ll (and move the head of ll to the next node
 * in the list), extracts the string stored in the head, and returns a
 * pointer to this string. Also frees the removed head node. */
char* remove_from_list(list *ll);

/* Prints every string in each node of the list ll, with a new line
 * character at the end of each string */
void print_list(list *ll);

/* Flushes (clears) the entire list and re-initializes the list. The passed
 * pointer ll should still point to a valid, empty list when this function
 * returns. Any memory allocated to store nodes in the list should be freed.
 */
void flush_list(list *ll);

/* De-allocates all data for the list. Ensure all memory allocated for list
 * ll is freed, including any allocated strings and list ll itself. */
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. Note the input to `free_list` is `list **ll` instead of `list *ll`, otherwise setting `ll` to NULL in this function will only affect the local variable in `free_list`, not the one in `main`. You should check return values from 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: `https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/`

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 any notes about resources you used or discussions you had with others. **I suggest starting on your assignment with this file and maintaining it as you work.**

PLEASE 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.