CSCI 401 Operating Systems
Instructor: Gedare Bloom
Project 1, Part 1 (50 points)

You must do this assignment by yourself. You may discuss the problem or solutions with your classmates or instructors, but **you may not share code with anyone**. You may not use code downloaded from the Internet.

Read all instructions carefully.

Due Sunday, September 9

Part 0 (0 points)

You will eventually need a Linux development environment for this course. The easiest approach is to install Virtualbox and download the virtual machine available from the text book's companion site http://os-book.com/ -- You have to select the "Operating Systems Concepts" (10th edition) book and find the link to the Linux Virtual Machine. A link is also available on Piazza. While you're on the site I highly recommend you browse around. The VM is a big file (almost 3 GiB) so you will want to start the download immediately.

If you need some help with the C language, you should definitely check out the http://cslibrary.stanford.edu/ resources 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".

Part 1 (50 points)

Create a C-string linked list data structure for storing character strings using a header file (list.h) and separate source file (list.c) for all functions. You need to implement the following functions, as well as define the struct(s) necessary for your list to work correctly. Do not use arrays. Do not declare arrays. There is no need to use [] anywhere. Do not use functions from the C++ Standard Template Library. Do not use cin or cout.

```
/* Allocates and initializes a new list */
list* create_list( );

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

/* Removes the string from the front of the list and
 * returns a pointer to it. The caller is expected to free
 * the string returned when finished with it. */
```

```c
char* remove_from_list(list* ll);

/* Prints every string in the list, 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 items in the list should be freed. */
void flush_list(list* ll);

/* De-allocates all data for the list. Ensure all memory
 * allocated for this list is freed, including any
 * allocated strings and the list itself. */
void free_list(list *ll);
```

Requirements

Put any struct definitions, typedefs 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 EOL characters \n not DOS style \n\r. You can write code on Windows, but you must 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. You should check return values on any function that can possibly fail.

Write a Makefile that compiles list.c and list_test.c to create a binary executable named `list_test`. If your project does not make with your Makefile - we won’t fix it or grade it. Your program should compile without any warnings or errors when using all standard GCC warnings in -Wall.

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.**

COMMENT YOUR CODE. If something doesn’t work, you may get partial credit if your comments show that you were on the right track. Remember to put a comment block at the top of every file. 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 of course for correctness and efficiency.

Hints

1. Some helpful C functions are `strlen, strncmp, printf, strncpy, strdup, malloc, free` (`man` these if necessary. Pay attention to return values!)
2. Always watch Piazza for updates on the assignments. Feel free to ask/answer questions there.

Submission Instructions (read carefully)

Make a tar and gzip file with your HowardU username (email address without the @.edu part) as the name.
– tar -zcvf HowardU-project1.tgz HowardU-project1/
Where all of your files are in the HowardU-project1 directory. For example I would replace HowardU above with gedare.bloom. Your submission must consist of your source code, README.txt, and Makefile; do not include compiled output! Upload the tgz file to Project 1 on Blackboard.