Homework 4- MPCS 51040

(last modified: April 19, 2017)

Issued: April 17, 2017

1 General Instructions

1.1 Compiling

Your code must compile with `gcc -std=c11 -Wall -pedantic`. There should be no warnings or errors when compiling with gcc (as installed on `linux.cs.uchicago.edu`).

1.2 Handing in

To hand in your homework, you need to commit all requested files (with correct filenames!) to your personal git repository.

Make a subdirectory called 'homework/hw4' and place your files under that directory. Don't forget to commit and push your files! You can check on `http://mit.cs.uchicago.edu` to make sure all files were committed to the repository correctly.

The deadline for this homework is April 24, 2017. To grade the homework, the contents of your repository at exactly the deadline will be considered. Changes made after the deadline are not taken into account.

1.3 Code samples

This document, and any file you might need to complete the homework can be found in the git repository
`https://mit.cs.uchicago.edu/mpcs51040-spr-17/mpcs51040-spr-17/`.

1.4 Grading

Your code will be graded based on the following points (in order of descending importance):

- Correctness of the C code: there should be no compiler errors or warnings when compiling as described in 1.1. There should be no memory leaks or other problems (such as those detected by valgrind).
- Correctness of the solution. Your code should implement the required functionality, as specified in this document.
- Code documentation. Properly documented code will help understand and grade your work.

- Code quality: your code should be easy to read and follow accepted good practices (avoid code duplication, use functions to structure your program, . . .). This includes writing portable code (which will work on both 32 bit and 64 bit systems).
- Efficiency: your code should not use more resources (time or space) than needed.

2 Assignment

2.1 `varstring` library

For this homework, we'll be creating a datatype suitable for storing strings, together with a set of functions to manipulate instances of the type.

The `varstring` type aims to provide a more capable string type for C. It has the following improved features over traditional C strings;

- can store *any* character, including 0-bytes as part of the string.
- variable length: no need to decide on a fixed length.
- conversion to and from traditional C strings for easy integration with existing code.

Below is an example of the functionality (this small program **must** work with your homework):

```
#include "varstring.h"
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char ** args)
{
   varstring_handle s = varstring_from("my first varstring");
   varstring_append_str(s, " was very easy to create!");
   puts(varstring_access(s));
   varstring_free(s);
   return EXIT_SUCCESS;
}
```

For the full list of functions, as well as a description of each function, see `varstring.h` (provided in the repository).

2.2 Deliverables

For this homework, you need to deliver the following items (in directory homework/hw4):

- A makefile for which:
  - the default action (i.e. no target specified) is to build your test program (i.e. same as target '`varstring_test`').
  - has a target 'valgrind' which builds your test program and executes it under valgrind.
  - has a target '`varstring_test`' which builds your test program.
  - has a target 'clean' which removes any files built by the makefile.
- `varstring.h`: **DO NOT MODIFY THIS FILE**.

- `varstring.c`: complete this file.
- `varstring_test.c`: this file should contain a `main` function which uses your new `varstring` type:
  - **When executed without command line arguments**, your program should call every single function of `varstring.h`, trying to validate the function behaves correctly. For example, to test the `varstring_access` function, you could construct a `varstring` with a known content and then check if the access function returns a C string matching that content.
  - **When executed with command line arguments**, your program should construct an empty `varstring` and append every option specified on the command line to the `varstring`. The program should finalize by printing out the contents of the `varstring`.

2.3 Advice

- Keep in mind that your `varstring` needs to be able to store *any* character, including `0`! So the following should work:

  ```
  varstring_handle h = varstring_new();
  varstring_append_char(h, 0);
  varstring_append_char(h, '\0');
  // prints 'Length is now: 2'
  printf("Length is now: %u\n", varstring_length(h));
  varstring_free(h);
  ```

- You are not allowed to modify the headerfile `varstring.h`. This means that you will have to pick a definition of `varstring_t` which is compatible with the provided functions.
- Try to avoid code duplication; if you find yourself implementing similar code multiple times see if you cannot isolate the common code in a helper function instead.
- Don't wait to test with valgrind until the end. Valgrind will help you discover issues early on if you run it while working on your code.