Lab 1: C

Introduction

Real operating systems are virtually always written in C. We will be using C throughout this course. This lab lets you revisit some language features specific to C:

- Pointers;
- Dynamic memory management;
- Variadic functions.

Overview

In this lab, you will write an *argument manipulator* program called `nyuc`, which stands for **Now You Understand C**. Your program will print each command-line argument in its original form, uppercase form, and lowercase form.

For example, if you run:

```
$ ./nyuc Hello, world
```

It should print:

```
[./nyuc] -> [./NYUC] [./nyuc]
[Hello,] -> [HELLO,] [hello,]
[world] -> [WORLD] [world]
```

Download

Please download nyuc.tar.xz. You can use the following command to unarchive it:

```
$ tar xf nyuc.tar.xz
```

Your task

The `main()` function is already given to you in `nyuc.c`:

```c
#include <ctype.h>
#include <stdio.h>

#include "argmanip.h"

int main(int argc, const char *const *argv) {
  char **upper_args = manipulate_args(argc, argv, toupper);
  char **lower_args = manipulate_args(argc, argv, tolower);

  for (char *const *p = upper_args, *const *q = lower_args; *p && *q; ++argv, ++p, ++q) {
    printf("[%s] -> [%s] [%s]\n", *argv, *p, *q);
  }

  free_copied_args(upper_args, lower_args, NULL);
}
```

The `main()` function invokes two functions, defined in `argmanip.h`:

```c
#ifndef _ARGMANIP_H_
#define _ARGMANIP_H_

char **manipulate_args(int argc, const char *const *argv, int (*const manip)(int));
void free_copied_args(char **args, ...);

#endif
```

Your task is to implement these two functions.

`manipulate_args()`

The `manipulate_args()` function takes three arguments:

- `argc` and `argv` are the same as the command-line arguments.
- `manip` is a pointer to a function that manipulates a character, such as `toupper()` and `tolower()`.

In this function, you will make a "copy" of the argument list. First, you will call `malloc()` once for the overall array in which each element is a string of type `char *`. Then, you will call `malloc()` for each element of that array, each of which will hold the manipulated version of each argument. Of course, you will have to copy each string character-by-character, passing through the `manip` function as you go.

`free_copied_args()`

The `free_copied_args()` function takes a variable number of arguments, each of which is a "copied" argument list returned by `manipulate_args()`. It is terminated by a `null` argument.

For each argument list, you must `free()` everything that you have `malloc()`ed. First, you need to `free()` all individual strings. Then, you need to `free()` the overall array.

Compiling

We will grade your submission on linserv1.cims.nyu.edu, which runs CentOS Linux release 7.9.2009. We will compile your program using `gcc` 9.2.0 with the C17 standard and GNU extensions. You need to run the following command to load it:

```
$ module load gcc-9.2
```

We have provided a `Makefile`, which turns on a few compilation flags to let `gcc` catch more potential issues:

```
CC=gcc
CFLAGS=-g -pedantic -std=gnu17 -Wall -Werror -Wextra

.PHONY: all
all: nyuc

nyuc: nyuc.o argmanip.o

nyuc.o: nyuc.c argmanip.h

argmanip.o: argmanip.c argmanip.h

.PHONY: clean
clean:
    rm -f *.o nyuc
```

To compile your program, type:

```
$ make
```

Testing

We will run your program under Valgrind Memcheck, a tool to detect many memory-related errors that are common in C programs, such as:

- Allocating the wrong size of memory.
- Using an uninitialized pointer.
- Accessing memory after it was freed.
- Overrunning a buffer.
- Leaking memory.
- …

These errors can lead to crashes and unpredictable behavior.

Valgrind is already installed on the Linux server. To run your program under Memcheck, type:

```
$ valgrind --leak-check=full ./nyuc Hello, world
```

It will print a lot of information. Here is a sample output:

```
==12345== Memcheck, a memory error detector
==12345== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==12345== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==12345== Command: ./nyuc Hello, world
==12345==
[./nyuc] -> [./NYUC] [./nyuc]
[Hello,] -> [HELLO,] [hello,]
[world] -> [WORLD] [world]
==12345==
==12345== HEAP SUMMARY:
==12345==     in use at exit: 0 bytes in 0 blocks
==12345==   total heap usage: 8 allocs, 8 frees, 108 bytes allocated
==12345==
==12345== All heap blocks were freed -- no leaks are possible
==12345==
==12345== For lists of detected and suppressed errors, rerun with: -s
==12345== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

In addition to printing the correct output, **your program must have "0 errors from 0 contexts,"** which means no illegal memory access, no memory leak, etc. Otherwise, Valgrind will give you detailed information about the memory errors and potential ways to diagnose the issue. You must fix all of them. You may find `gdb` useful for debugging.

Submission

You will submit a single file, `argmanip.c`.

You **must not** modify any provided files (`nyuc.c` , `argmanip.h` , and `Makefile`). We will compile your code together with our provided files and use automated scripts to grade your submission.

Rubric

The total of this lab is 100 points, mapped to 5% of your final grade of this course.

You will get 40 points as long as you submit something meaningful and your program compiles successfully.

You will get another 30 points if your program prints the correct output.

You will get the final 30 points if your program prints the correct output and Valgrind reports 0 errors. Otherwise, for each error context, there will be a 5-point deduction, up to 30 points.

Tips

If you are experienced in C programming, this lab shouldn't be too difficult. However, if you are not familiar with C (even if you know Java), now is an excellent time to catch up before Lab 1 posts. *The C programming language (2nd ed.)* by Brian W. Kernighan and Dennis M. Ritchie is a timeless classic (NYU students can read it online for free), and this C tutorial is a good alternative.

References: Lee et al., Follow the River and You Will Find the C, SIGCSE'11.