

VG101 FA2021 Lab 3

Author: Haorong Lu

If you have any question, please contact: ancientmodern@sjtu.edu.cn, or via Feishu/WeChat

Note: To complete Lab 3, you may only read "Tutorial" and "Manual" sections, the other is just some nonsense

On Tuesday Evening ...

As you are comfortably sitting in the cafeteria and enjoying the food, recalling the delicate MATLAB programs you just wrote in VG101 Mid1, your mobile phone loudly rings and interrupts your rare free time in JI. You receive an email from the VG101 TA, just one hour after mid1... Is that good news?

Date: **Tue. 19 Oct. 2021 18:40:27 +0800**

From: ancientmodern@sjtu.edu.cn

To: vg101-excellent-students@sjtu.edu.cn

Subject: **The mathsh Project**

Attachment: **Tutorial: Data Type & Expression in C, lab3_Tutorial.c**

Dear students,

Congratulations! Everyone passed the Mid 1 and got the "Master of MATLAB" certificate. Because of your outstanding performance, Mr. Jigang decided to assign you a new project. However, because only GCC is installed on our server, you have to use C to implement the project. Don't worry about the difference between C and MATLAB, we have prepared a tutorial (in the attachment) to give you a better understanding of the data types in C before you start project work.

Please contact us as soon as possible after completing the tutorial and we will send you another email with detailed information about the mathsh project.

Thanks a lot for your devotion and cooperation. For the VG101!

Best Wishes,

Haorong Lu

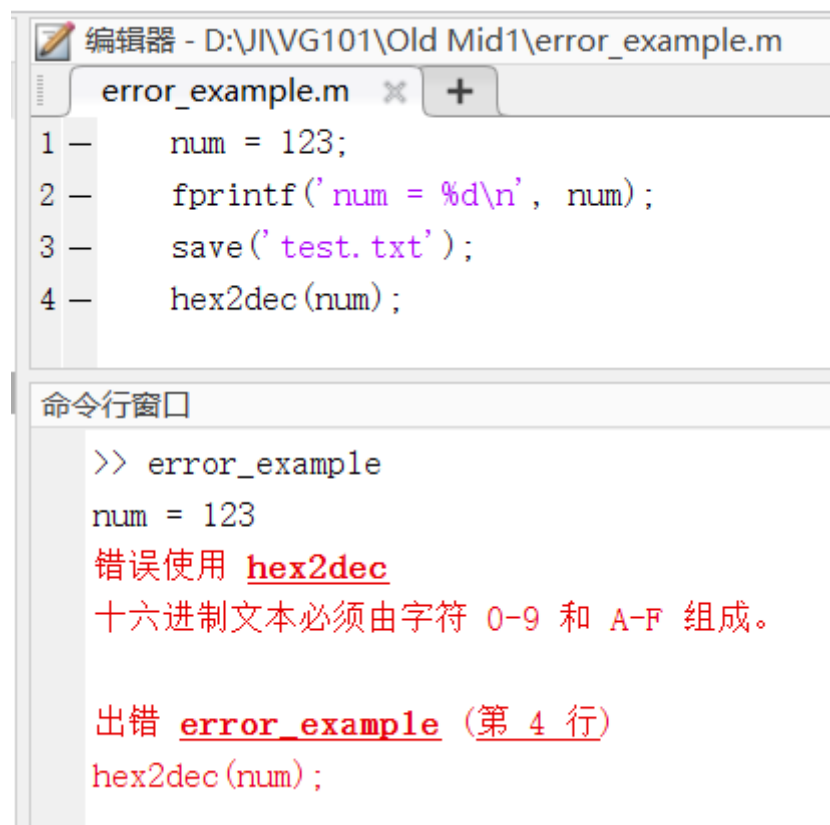
As an experienced JI student, you are already accustomed to taking new assignments as a "reward" for completing the exam. Rapidly eat up the remaining food, you open the tutorial attached in the email.

1. Tutorial: Data Type & Expression in C

Introduction

Different from MATLAB, C and C++ are *statically typed languages*, and we need to explicitly define the types of variables when you declare them.

Recall that we do not need to define what type a variable is in MATLAB when declaring the variables since MATLAB is a *dynamically typed language*, in which the type is determined automatically at runtime. Seems easier for the programmer, but significantly more error-prone, and the errors will not be discovered until runtime. For example,



```
编辑器 - D:\JI\VG101\Old Mid1\error_example.m
error_example.m x +
1 - num = 123;
2 - fprintf('num = %d\n', num);
3 - save('test.txt');
4 - hex2dec(num);

命令行窗口
>> error_example
num = 123
错误使用 hex2dec
十六进制文本必须由字符 0-9 和 A-F 组成。

出错 error_example (第 4 行)
hex2dec(num);
```

In this script, we forget to convert `num` to `string` before passing it into `hex2dec()`, but because the type is determined at runtime, the error is not detected until `num` is already printed and something is saved into `test.txt`. This may cause some unexpected dangerous problems, especially for a low-level language that has the access to hardware and operating system resources.

Moreover, this may lead to lower performance, for example, when MATLAB encounters a new variable, it automatically allocates the right amount of storage; when a variable changes, it re-allocates memory if necessary. This re-allocation process will definitely reduce the performance.

In *statically typed languages*, on the other hand, variable types are explicitly declared and thus are determined at compile time. This lets the compiler decide whether a given variable can perform the actions requested from it or not, and thus many errors can be pre-detected by the compiler.

The concept of type is very important in C and C++, it will be helpful to make every thing clear at the beginning of your journey.

Tasks

In this section, you need to finish all **"TODO"** requirements in the attaching file `lab3_Tutorial.c`. There are some tasks about the data type and expression in C language.

Your final goal is to make this file compliable under following compiling command:

```
gcc ./lab3_Tutorial.c -o lab3_Tutorial -Wall -Wextra -Wconversion -Werror -std=c99
```

And the final executable file should be able to run normally. There should be no "Assertion failed!" in the output and the exit code for your program should be 0.

For the detailed information about what you should do, please refer to `lab3_Tutorial.c`.

Requirements

1. You can not modify any lines other than specified by **"TODO"** comments.
2. Only modify the part that **"TODO"** comments asks you to modify, i.e. only modify type specifier, value after certain symbol.
3. Do not delete any comments or support functions

Hints

1. It will be helpful to first learn about the implicit type conversion in C. One possible resource is [this article](#) on cppreference. You can also refer to other resources if you like.
2. `assert` function is also useful in your own coding. You can use this function to check some invariants in your program.
3. Operation precedence is also important in C/C++ (also in this lab), you may learn about this on [this page](#) from cppreference, as well. It defines in what sequence should the program process an arithmetic expression.

Half an hour later ...

Complete the tutorial quickly and accurately, you contact TA ASAP and receive another email,

Date: Tue. 19 Oct. 2021 19:10:54 +0800
From: ancientmodern@sjtu.edu.cn
To: vg101-excellent-students@sjtu.edu.cn
Subject: [Auto Reply] The mathsh Project
Attachment: Manual: The mathsh Command-Line Interface

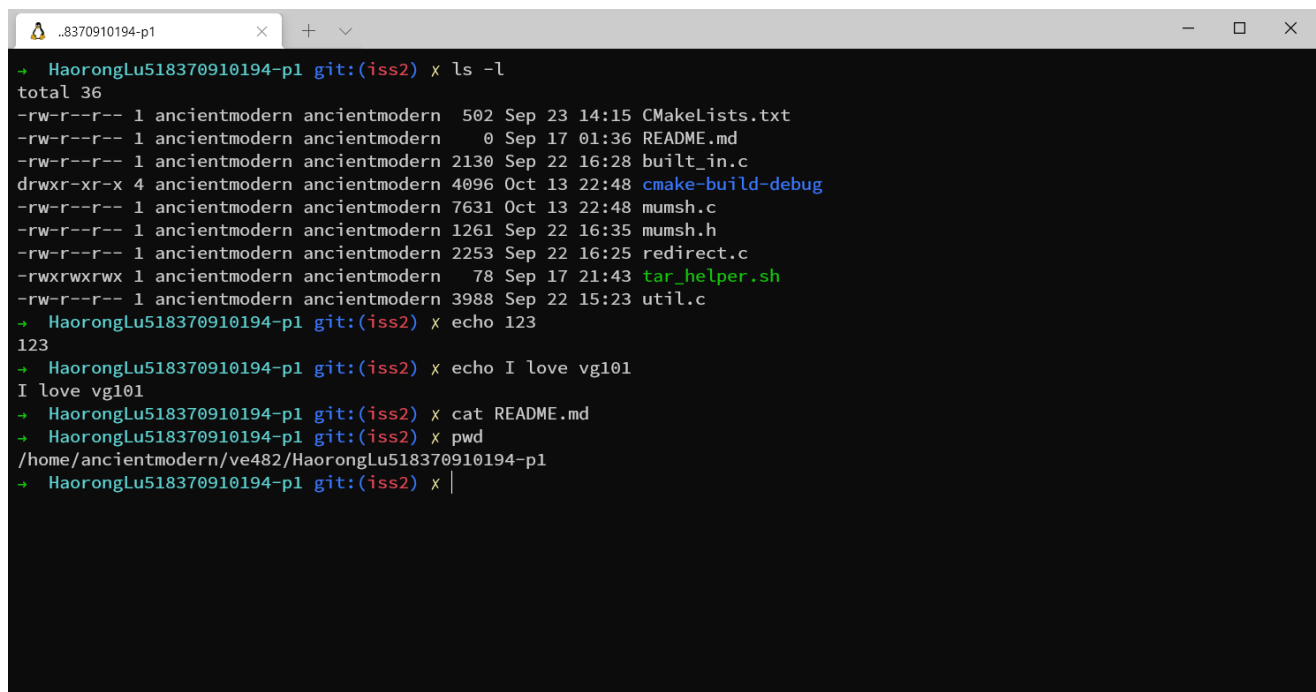
Dear students,
Well done! I have never seen a student who completes the tutorial so quickly as you. It's time to take some real challenges! Mr. Jigang wants you to implement a math-oriented command-line interface called "**mathsh**". Don't worry if you do not know what is command-line interface or shell, you can check the manual for a brief explanation. Let's start coding!

Thanks a lot for your devotion and cooperation. Long life to Mathsh!

Best Wishes,
Haorong Lu

2. Manual: The mathsh Command-Line Interface

Introduction



```
HaorongLu518370910194-p1 git:(iss2) x ls -l
total 36
-rw-r--r-- 1 ancientmodern ancientmodern 502 Sep 23 14:15 CMakeLists.txt
-rw-r--r-- 1 ancientmodern ancientmodern 0 Sep 17 01:36 README.md
-rw-r--r-- 1 ancientmodern ancientmodern 2130 Sep 22 16:28 built_in.c
drwxr-xr-x 4 ancientmodern ancientmodern 4096 Oct 13 22:48 cmake-build-debug
-rw-r--r-- 1 ancientmodern ancientmodern 7631 Oct 13 22:48 mumsh.c
-rw-r--r-- 1 ancientmodern ancientmodern 1261 Sep 22 16:35 mumsh.h
-rw-r--r-- 1 ancientmodern ancientmodern 2253 Sep 22 16:25 redirect.c
-rwxrwxrwx 1 ancientmodern ancientmodern 78 Sep 17 21:43 tar_helper.sh
-rw-r--r-- 1 ancientmodern ancientmodern 3988 Sep 22 15:23 util.c
HaorongLu518370910194-p1 git:(iss2) x echo 123
123
HaorongLu518370910194-p1 git:(iss2) x echo I love vg101
I love vg101
HaorongLu518370910194-p1 git:(iss2) x cat README.md
HaorongLu518370910194-p1 git:(iss2) x pwd
/home/ancientmodern/ve482/HaorongLu518370910194-p1
HaorongLu518370910194-p1 git:(iss2) x |
```

A **command-line interface (CLI)** processes commands to a computer program in the form of lines of text. The program handling the interface is called a **command-line interpreter**. The most widely-used command-line interface is the shell, which is the outermost layer around the operating system. It exposes an operating system's services to a human user or other program. Here are some common shells on different operating systems,

- Windows: cmd, powershell
- Mac: terminal (zsh by default)
- Linux: sh, bash, zsh, ...

Though today many users rely upon graphical user interfaces and menu-driven interactions, the command line interface is often faster than graphical user interfaces as there is no need to move the mouse around to click in many “random places” on the desktop; commands are usually simple and well documented.

The reason we introduce **CLI** is that different from running MATLAB script by directly clicking the icon, for C and C++, you have to type some commands in some kind of **CLI** to compile and run the program, just like the compiling command for the tutorial. You may feel very awkward at first, but after getting familiar with it, you will be addicted to its high efficiency and high degree of freedom.

Despite the fact that Mathsh contains the word "sh", it's actually an application command-line interface instead of an operating system command-line interface. It's similar to the "command window" of MATLAB, but it only has some built-in commands, which means you do not need to write other programs to support the commands.

I believe that you may be already tired of long boring introductions. Let's start coding!

Tasks

In this part, you need to implement the `mathsh`, which simply consists of an endless loop (unless exit) that parses the user input, and execute the calculation according to the user input. When waiting the `mathsh` displays a prompt, here we want it to display `mathsh $`.

As a math shell, your shell is expected to support following commands,

- **max: Find the largest value in a series of numbers**
 - Format: `max {number of input integers} {n integers}`
 - Input: `max 5 2 1 3 5 4`
 - Output: `max: 5`
- **min: Find the smallest value in a series of numbers**
 - Format: `min {number of input integers} {n integers}`
 - Input: `min 4 1 2 3 4`
 - Output: `min: 1`
- **avg: Calculate the average of a series of numbers**
 - Format: `avg {number of input integers} {n integers}`
 - Input: `avg 4 2 2 1 1`
 - Output: `avg: 1.500`
- **fib: Calculate the Fibonacci number F_n , where $F_0 = 0, F_1 = 1, F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$**
 - Format: `fib n (n is integer)`

- Input: fib 30
- Output: fib: 832040
- exit: **Exit the** mathsh
 - Format/Input: exit
 - Output: Exit mathsh! and exit the program.

A simple example:

```
mathsh $ max 3 1 2 3
max: 3
mathsh $ min 3 1 2 3
min: 1
mathsh $ avg 3 1 2 3
avg: 2.000000
mathsh $ exit
Exit the program!
```

Hints

1. There will be **no invalid inputs** in our testcases, and the number of input integer is always **greater than 0**.
2. As you have not learned about array and string in C, we provide a template `lab3_Mathsh.c` to show how to get and compare the command name. You may write your `mathsh` based on the template.
3. The input numerical arguments are always **integers within** $[-100000, 100000]$, which data type should be used to receive the input arguments?
4. You may want to use an array to store all numerical arguments and then start manipulation, but actually it can be done without an array by combining loops, `scanf()`, and simple calculation.
5. It's obvious that the results of `avg` may be a non-integer, how to get the precise result? Is a explicit type conversion needed?

Submission

Your solution should be written into two `.c` files called `lab3_Tutorial.c` and `lab3_Mathsh.c`. When you come to the lab, show these files to TA and run it to show the output according to TA's instructions. We will grade according to both your output, oral presentation and your code.

After the lab session is over, remember to compress your 2 files into a single zip file in the following name format: `lab3-[Your Last Name]-[Your First Name]-[Your SJTU ID].zip`, for example, `lab3-Lu-Haorong-518370910194.zip` and submit it to corresponding homework page on canvas.

Reference

1. <https://introcs.cs.princeton.edu/java/11matlab/>
2. https://en.wikipedia.org/wiki/Command-line_interface
3. [https://en.wikipedia.org/wiki/Shell_\(computing\)](https://en.wikipedia.org/wiki/Shell_(computing))
4. Du Yuexi, lab04: Introduction to C: Data Type & Expression & The Hangman, VG101 FA2020
5. Manuel, VE482 FA2021 P1 & P2 (Inspiration of stories and introductions about CLI)