VG101 — Intoduction to Computers & Programming

Lab 4

Instructor: Manuel Charlemagne

TA: Zhengyuan – UM-JI (Summer 2019)

Goals of the lab

- Design and build a project
- Build a static/shared library
- Documentation of C/C++

1 Introduction

Haruka, Kana, and Chiaki are busy with many many exams recently so that they have no time to introduce the background of their situation this time.

2 Naive Math Library

Moving from Matlab to C, Haruka found the way the arithmetical operations work in C annoying. For example, when typing `9/5`, she is expecting a `1.8`. In C, however, the expression would not be evaluated as `1.8`, neither the closest integer `2`, but `1`, which is `floor(1.8)`. Inspired by this, Haruka is considering to write a naive math library that is more intuitive to her.

2.1 Interface

Haruka plans to use a single function to deal with all the arithmetical operations that she wants to override. She describes the function as follows:

```c
// evaluate takes 2 integer and an operator as arguments and returns a double.
// The behavior of the function depend on the operator, as described below.
// Note that the result should never be rounded.
// operator == '+': return operand1 + operand2. (optional)
// operator == '-': return operand1 - operand2. (optional)
// operator == '*': return operand1 * operand2.
// operator == '/': return operand1 / operand2.
// operator == '^': return operand1 ^ operand2, where operand2 is the exponent.
// operator == 'l': return log(operand1, operand2), where operand1 is the base.
// operator == 'm': return r, which satisfies operand1 = q * operand2 + r,
// where q and r are integers, r * operand2 >= 0, and abs(r) < abs(operand2).
double evaluate(int operand1, char operator, int operand2);
```

2.2 Implementation

Create a c file and implement the function evaluate. Also, create a corresponding header file for the library.

Hints:

- It is a good practice to use switch-case statement in such scenarios. Note that the behavior of switch-case statements is different from that in Matlab in some ways.
- You cannot directly use the % operator in C for the modular operation described.
- You may find it non-trivial to implement exponent and logarithm from scratch. Although the standard library of C is not as rich as that of MatLab, you may still find some functions in standard library helpful for this task. When searching
  - Use `en.cppreference.com` or `cplusplus.com` for the documentations of the C standard library.
  - Use English instead of Chinese in search engines, use keywords instead of full sentences.
  - Generally avoid `csdn.net`, `cnblogs.com`, and any other Chinese websites. Some of the information on such sites may not be reliable. Use `stackoverflow.com` instead of the sites above. It includes peer reviews, so that the top answers are usually reliable.
- Use header guard for the header file.

2.3 Building

After implementing the library, Haruka is so satisfied with the library that she wants to share it with Kana. However, she does not want Kana to learn the tricks she used in the library. Therefore, she decides to give a pre-built object file as well as the header file to Kana.

To build the library as a static one, run the following command to get the `naive_math_lib.o`.

```
gcc -c naive_math_lib.c
```

For cmake users, add the following line to your `CMakeLists.txt`.

```
add_library (name_of_object_file STATIC naive_math_lib.c)
```

To start with cmake, you can checkout this Quick CMake Tutorial written by JetBrains for CLion users. Checkout the official documentation for further details.

Interlude: Kana was confused, since the header file Haruka wrote did not include any instructions on the usage. She did not know what to pass as the operator to the function. Neither can she look at the implementation to see the usage. Haruka was sorry for the mistake, and added comments in the header file to describe the usage of the library.

2.4 Readings from previous semesters

Library

In computer science, a library is a collection of non-volatile resources used by computer programs, often for software development. These may include configuration data, documentation, help data, message templates, pre-written code and subroutines, classes, values or type specifications.

For a large project, it may contain millions of lines of code. It will take several hours to build only part of the whole project. So it is very important to separate different parts of the project into libraries.

A static library or statically-linked library is a set of routines, external functions and variables which are resolved in a caller at compile-time and copied into a target application by a compiler, linker, or binder, producing an object file and a stand-alone executable.

A shared library or shared object is a file that is intended to be shared by executable files and further shared object files. Modules used by a program are loaded from individual shared objects into memory at load time or run time, rather than being copied by a linker when it creates a single monolithic executable file for the program.

3 CLI Calculator

Kana likes Haruka's library a lot. Indeed, she plans to write a simple command-line calculator using Haruka's library.

3.1 Interface

Kana describes the functionality she wants as follows.

```
$ ./calculator --help
usage: ./calculator [option | file | -]

Options:
-h       : print this help message and exit (also --help)
-c str   : evaluate str and print the result to stdout

Modes:
-        : read expressions from stdin and print results interactively (default)
file     : read expressions from file and print all results

Expression Format:
The format is up to you. Feel free to support advanced features.
```

3.2 Implementation

To accept command line arguments, use the `argc` and `argv` as arguments to the main function.

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    // argc (argument count): number of arguments
    // argv (argument vector): array of strings, each holding an argument
    // arguments are separated by spaces
    // first argument is the name of the executable
    // (prefixed by "./" if called in this style)

    // demo: print all the arguments
    for (int i = 0; i < argc; ++i) {
        printf("%s\n", argv[i]);
    }
    return 0;
}
```

Include Haruka's header file to have the declaration of the library function.

3.3 Building & Linking

To build the calculator executable, run

```
# build the calculator object file
gcc calculator.c -c
# link the object file with the library
gcc calculator.o naive_math_lib.o -o calculator
```

You can also combine the 2 steps as

```
# build and link with a single command
gcc calculator.c naive_math_lib.o -o calculator
```

For cmake users,

```
add_executable(calculator calculator.c)
target_link_libraries(calculator /path/to/naive_math_lib.o)
```

or combined

```
add_executable(calculator calculator.c naive_math_lib.o)
```

4 Library Revision

Kana found that the exponent operation in Haruka's library was sometimes slow and even inaccurate. She told Haruka the issue. Haruka searched for the issue and found that `pow()` works on floating points numbers, and is not optimized for integers. To help Haruka, optimize the exponent operation for integers.

Also, as Kana was trying to chain operations to evaluate complicate expressions, she found that if she uses the return value of `evaluate()` as an argument of `evaluate()`, the returned double would be casted into integer. She started to think that, it may be a better idea to stick to double for such math calculation, like MatLab and `pow()`. She began to doubt the point of the naive math library, and finally turned to the primitive operations in C and `math.h` for his calculator. Anyway, she did not regret this, as she learned the types in C as well as the usage of C library. Also, she learned to be skeptical to any non-standard libraries, like Haruka's naive math library.