

CS205 Project 2 - A Much Better Calculator

Name: 匡亮(KuangLiang)

SID: 12111012

Part 1 - Brief Introduction

项目位于：<https://github.com/sustechkl/Simple-Calculator>，将在 2022 年 10 月 17 日零点设为公开（如果我忘记了，麻烦邮件提醒）。

代码用 CMake 管理，在 CMake 中开启了 C++11 标准。

交互方法：

编译后，在终端输入 `./simbc` 回车开始使用。

交互方法基本模仿 Linux 下的 `bc` 计算器。输入会忽略所有空格，以回车作为一次指令的结束。

1. 以 # 号开头的指令：

1. 输入 `"#h"` 可以查看所有用法；
2. 输入 `"#a"` 可以查看所有预设的数学函数和说明；
3. 输入 `"#r"` 可以读取文件；
4. 输入 `"#q"` 可以结束进程；

带井号的指令只读取井号后的一个字符。

2. **支持自定义变量**。输入的内容带有 `"=`"，则将左侧作为变量名、右侧表达式作为变量值（不能出现当前变量，即，不支持解方程）。合法变量名只能含有字母、下划线，且不应该重复。
3. **支持数学函数**。用 `sqrt(x)`、`log(x)`、`sin(x)`、`cos(x)` 等来调用一个函数。`x` 可以是一个表达式。
4. **支持自定义函数**。输入的内容带有 `"(x)="`，则将左侧作为函数名，右侧可以写一个含有 `x` 的表达式。`x` 可以是任意合法变量名。之后可以进行调用。自定义函数的变量 `x` **不会被**设置为新的自定义变量，如果它与已有自定义变量重复，计算时会忽略已有自定义变量的值。

可能的报错：

1. `syntax error`: 输入不能构成一个表达式。
2. `invalid variable name`: 错误的变量名，可能是不合法或重复。
3. `invalid function name`: 错误的函数名，可能是不合法或重复（包括和预设函数重复）。
4. `runtime error`: 运算中发生错误，可能是对 0 进行除法或取模等。
5. `unknown command`: 未知指令，如果第一个字符为 # 而之后输入的内容不是现有的指令。
6. `unknown variable`: 算式中出现了未知的变量名。
7. `unknown function`: 算式中出现了未知的函数名。
8. `recursion`: 自定义函数出现了递归行为。

报错后，会忽略这次计算或定义，并给出可能的出错位置。

颜色说明：

为了方便使用，终端的所有输入输出均用颜色进行了区分（仅 Linux，尚未测试在 Windows 系统下表现如何）。

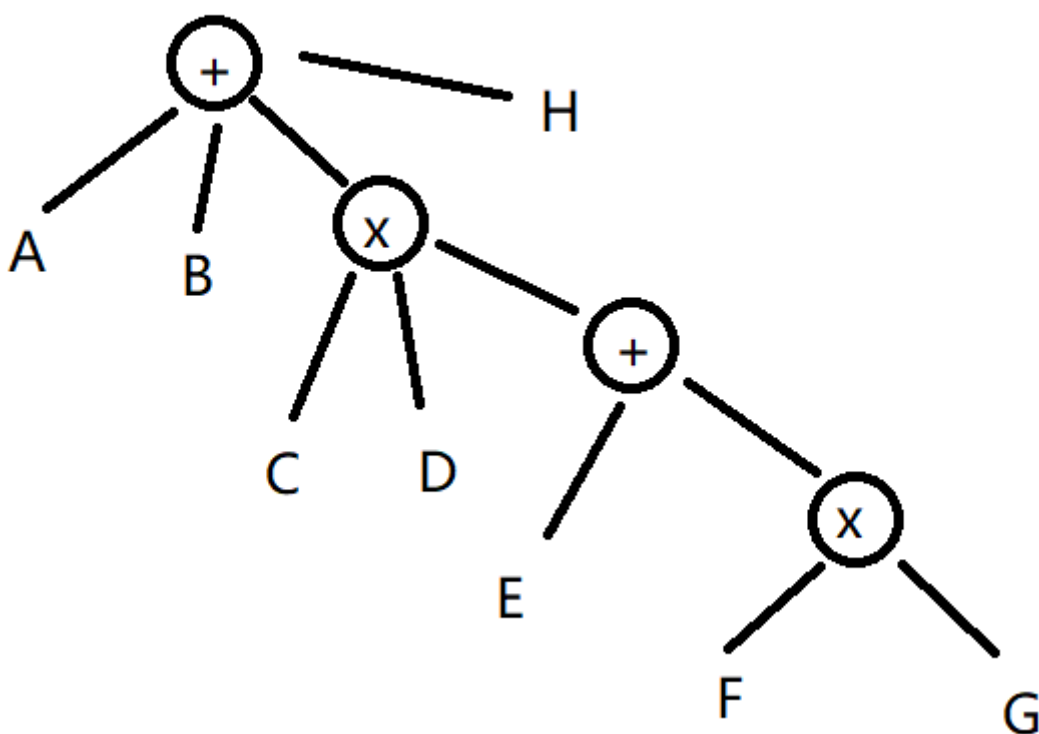
1. 白色：用户的所有输入，包括终端和文件。
2. 绿色：运行中没有发生错误时，输出的答案。
3. 红色：运行中发生了错误时，输出的提示。
4. 蓝色：运行错误的提示中，展示用户输入。
5. 黄色：系统的一般提示。

也就是说，有且仅有用户输入是白色的。这样之后的测试截图就会显得更清晰。

Part 2 - Code Analysis

Section 1 - 表达式解析

$A+B+CxDx(E+FxG)+H$



在每层运算时，只考虑优先级**最低**的运算，对其他运算部分进行分治。这样每层只用处理一种优先级的运算（从低到高：加减、乘除模、乘方、函数和括号）。

Section 2 - 函数运算

主要想法：如果是预设的函数，用 `cmath` 库对应的函数进行计算；如果是自定义函数，算出自变量的值后，为函数输入时使用的自变量强制设置一个临时值，然后重新调用计算表达式的主方法对存储的函数表达式进行计算。

实现时，通过使用 `sscanf` 函数和正则表达式，明显简化了过程。至于识别一个函数，最开始的想法是用 `map` 将 `string` 转成 `int`，但在实现的过程中，由于数据都是通过两个指针 `begin` 和 `end` 来指示的，转成 `string` 还要额外的一步，于是最后考虑用了手写的 `Trie` 树来识别。

Section 3 - 普通运算

提取出最低级的运算符，然后依次计算，在除法和取模时判断是否发生错误。

Section 4 - 预设函数

使用了 C++11 标准后，函数可以作为一个实例存储在 `std::function` 类中，大大简化了引入预设函数的代码。想要为计算器添加新的预设函数时，只需在此声明即可（可选：添加解释，将在 #a 中展示），其他库中的函数或是自己定义的函数（如 `sec` 等）都可以。

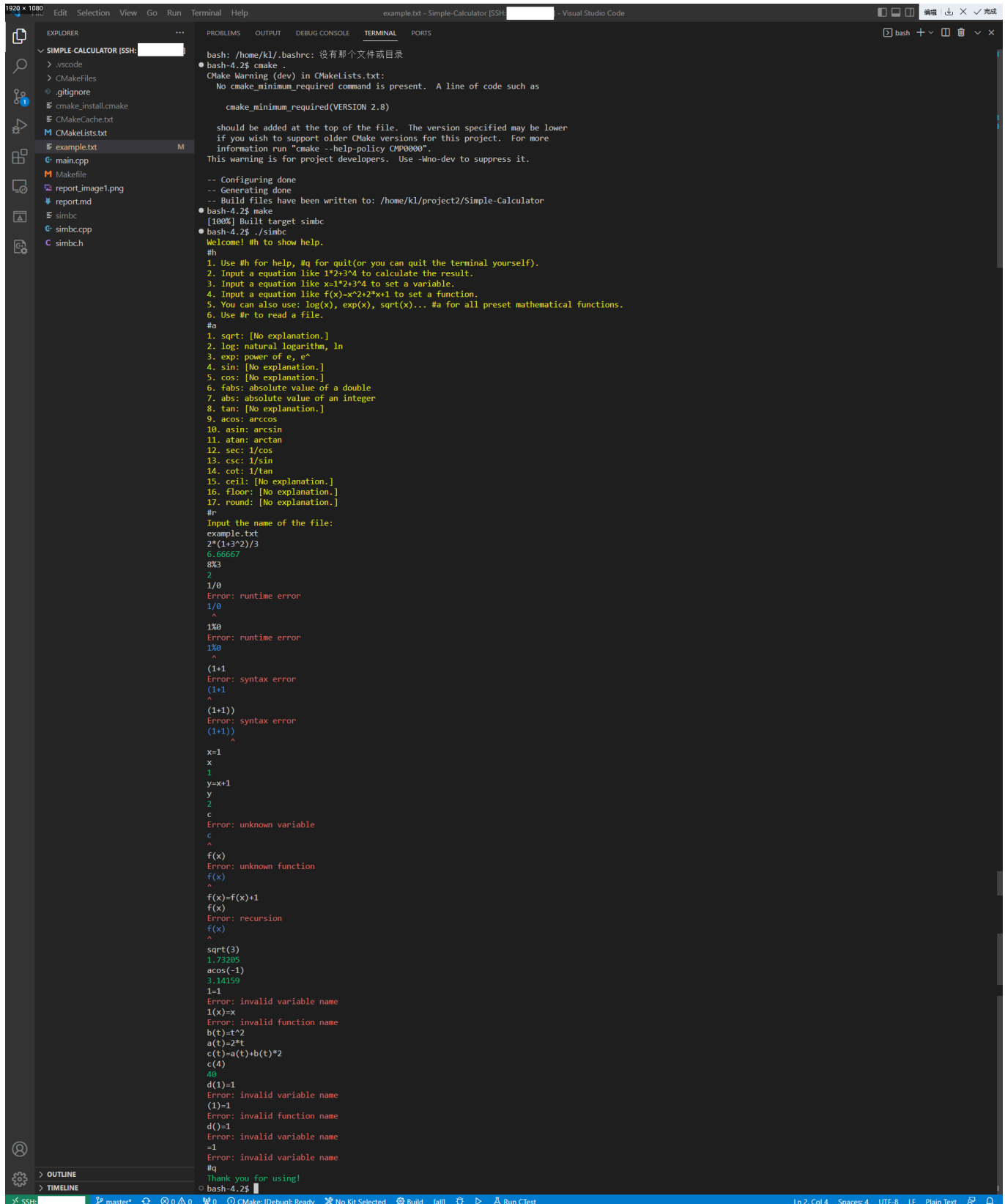
```
inline void addFunction(char* name, std::function<double(double)> F, std::string
explanation = "") {
    ++presetFunctionId;
    presetFunctions[presetFunctionId] = F;
    presetFunctionName[presetFunctionId] = std::string(name);
    if(explanation.length()) presetFunctionExplanations[presetFunctionId] =
explanation;
    functionTrie.Insert(name, 0, -presetFunctionId);
    return;
}
double sec(double x) { return 1 / cos(x); }
double csc(double x) { return 1 / sin(x); }
double cot(double x) { return 1 / tan(x); }
void init() {
    // You can add more function here easily
    addFunction("sqrt", sqrt);
    addFunction("log", log, "natural logarithm, ln");
    addFunction("exp", exp, "power of e, e^");
    addFunction("sin", sin);
    addFunction("cos", cos);
    addFunction("fabs", fabs, "absolute value of a double");
    addFunction("abs", abs, "absolute value of an integer");
    addFunction("tan", tan);
    addFunction("acos", acos, "arccos");
    addFunction("asin", asin, "arcsin");
    addFunction("atan", atan, "arctan");
    addFunction("sec", sec, "1/cos");
    addFunction("csc", csc, "1/sin");
    addFunction("cot", cot, "1/tan");
    addFunction("ceil", ceil);
    addFunction("floor", floor);
    addFunction("round", round);
    return;
}
```

Section 5 - 补充功能

1. 颜色：通过 `std::cout << "\033[31m";` 等实现。
2. 文件：通过 `std::ifstream` 类实现。

Part 3 - Result & Verification

这里用读入 `example.txt` 的方式来完成测试。



```
bash: /home/kl/.bashrc: 没有那个文件或目录
bash-4.2$ cmake .
CMake Warning (dev) in CMakeLists.txt:
  No cmake_minimum_required command is present.  A line of code such as

    cmake_minimum_required(VERSION 2.8)

  should be added at the top of the file.  The version specified may be lower
  if you wish to support older CMake versions for this project.  For more
  information run "cmake --help-policy CMP0000".
  This warning is for project developers.  Use -Wno-dev to suppress it.

-- Configuring done
-- Generating done
-- Build files have been written to: /home/kl/project2/Simple-Calculator
bash-4.2$ make
[100%] Built target simbc
bash-4.2$ ./simbc
Welcome! #h to show help.
#h
1. Use #h for help. #q for quit(or you can quit the terminal yourself).
2. Input a equation like 1*2+3*4 to calculate the result.
3. Input a equation like x=1*2+3*4 to set a variable.
4. Input a equation like f(x)=x*2+2*x+1 to set a function.
5. You can also use: log(x), exp(x), sqrt(x)... #a for all preset mathematical functions.
6. Use #r to read a file.
#a
1. sqrt: [No explanation.]
2. log: natural logarithm, ln
3. exp: power of e, e^x
4. sin: [No explanation.]
5. cos: [No explanation.]
6. fabs: absolute value of a double
7. abs: absolute value of an integer
8. tan: [No explanation.]
9. acos: arccos
10. asin: arcsin
11. atan: arctan
12. sec: 1/cos
13. csc: 1/sin
14. cot: 1/tan
15. ceil: [No explanation.]
16. floor: [No explanation.]
17. round: [No explanation.]
#r
Input the name of the file:
example.txt
2*(1+3*2)/3
6.66667
883
2
1/0
Error: runtime error
1/0
^
180
Error: runtime error
180
^
(1+1
Error: syntax error
(1+1
^
(1+1))
Error: syntax error
(1+1))
^
x=1
x
1
y=x+1
y
2
c
Error: unknown variable
c
^
f(x)
Error: unknown function
f(x)
^
f(x)=f(x)+1
f(x)
Error: recursion
f(x)
^
sqrt(3)
1.73205
acos(-1)
3.14159
1-1
Error: invalid variable name
1(x)=x
Error: invalid function name
b(t)=t*2
a(t)=2*t
c(t)=a(t)+b(t)*2
c(4)
40
d(1)=1
Error: invalid variable name
(1)=1
Error: invalid function name
d()=1
Error: invalid variable name
=1
Error: invalid variable name
#q
Thank you for using!
bash-4.2$
```

Part 4 - Difficulties & Solutions

项目最大的难点自然是表达式解析。开始写任何代码之前，我在草稿纸上设计了非常多种解析表达式的方法，最后最让我满意的也就是我最后采用的利用分治来解析。相比其他几种解析的方法，分治式的解析在真正进行运算的时候不需要考虑优先级（优先级通过分治树的结构来体现），因此代码要简单非常多。

刚刚完成最初的代码的时候，它是没有自定义变量和函数的功能的。但由于代码的框架很清晰，加上第一次编写时留下了很多注释，所以之后加入新的功能的时候也可以很方便地分别进行模块化。最终，自定义变量和自定义函数的功能没有调试多久就成功上线了。

本来项目做到这里就结束了，但写报告的时候突然发现展示测试是一个很麻烦的事情。写好“剧本”后，输入时得小心翼翼，但即使这样做出来的效果也不好：难以区分输入和输出。于是我想到了加入颜色和文件的功能，这样把测试样例存在 `example.txt` 中，就不用测试时一次次输入剧本了，而且输入输出也很容易区分。