CS205 C/ C++ Programming - Project 2 A Better Calculator

Name: Lv Yue **SID:** 11710420

***This project is hosted at https://github.com/JustLittleFive/simpleCalculator ***

Part1: Analysis

Project goal: Implement a much better calculator than that in Project 1.

It is envisaged that the calculator will support high precision addition + subtraction - multiplication * division / square root sqrt()->\$ remainder % exponentiation ^ factorial! operations.

In addition, the calculator supports user-defined variables.

Part2: Core code

The **customFunction.hpp** declare the *calculator's input processing functions, core component functions, calculator mode overload functions* and *operation execution functions*:

```
23
     // calculator's input processing functions
24
25
     bool validChecker(vector<string> &seglist);
     vector<string> initializer(string input);
26
     list<string> suffixed(vector<string> seglist);
27
     tuple<string, int, int, bool> preprocess(string str);
28
29
30
     //core component functions
     int addZeros(string &num1, string &num2);
31
     int addZerosR(string &num1, string &num2);
32
     string strAdd(string s1, string s2);
33
     string strMinus(string s1, string s2);
34
35
     // calculator mode overload functions
36
     string calculator(list<string> suffixList);
37
     string calculate(string num, string unaOperator);
38
     string calculate(string num1, string num2, string biOp
39
     // operation execution functions
41
     string funcAdd(string num1, string num2);
42
     string funcSub(string num1, string num2);
43
     string karatsuba(string str1, string str2);
44
45
     string funcDivide(string num1, string num2);
     string funcExp(string num1, string num2);
46
     string funcMod(string num1, string num2);
47
48
```

The **globalVariables.hpp**, where a *map* is declared as *custom variable storage space* and multiple *operator string constants*:

```
#pragma once
18
19
     #include "StandardLibraryHeaderFile.hpp"
20
21
     using namespace std;
22
23
     const string symbol = "+-*/^$!%()";
24
     const string biOperator = "+-*/^%";
25
     const string unaOperator = "$!";
26
27
     extern map<string, string> customVarible;
28
29
```

The **StandardLibraryHeaderFile.hpp**, contains all required standard library header files:

```
#include <algorithm>
19
     #include <cmath>
20
21 #include <cstring>
   #include <iostream>
22
     #include <list>
23
     #include <map>
24
25 #include <regex>
26 #include <stack>
27 #include <tuple>
28
     #include <math.h>
29
30
```

Finally the **headerCollections.hpp**, collect all the headers, to make .cpp files #include once and for all:

```
#include "StandardLibraryHeaderFile.hpp"

#include "globalVariables.hpp"

#include "customFunctions.hpp"

#include "customFunctions.hpp"
```

The project file has a **brand new copyright header** (sorry for the copyright header error caused by my previous oversight!):

(My GitHub home page: https://github.com/JustLittleFive)

The function implementation is divided into five project files:

- 1. **simpleCalculator.cpp**: Main function, as the program entry, handles the assignment statement of the custom variable.
- 2. **preprocess.cpp**: Preprocessing function implementation. Not only processing user input, but also preprocessing functions for scientific notation and decimal point are here.
- 3. **calculator.cpp**: Manages the order in which expressions are evaluated (suffix expression), and the overloaded calculate function handles unary and binary operators separately.
- 4. **calFunc.cpp**: The corresponding operation functions are implemented for all supported operators. Some temporarily unsupported inputs are also processed here.
- 5. **calComponents.cpp**: The common core component that implements string-type mathematical operations, and is used in almost every operator's implementation function.

... and the *Karatsuba multiplier* in calFunc.cpp:

```
/// @brief Multiply function by Karatsuba algorithm, inspired by the
article
/// https://www.geeksforgeeks.org/karatsuba-algorithm-for-fast-
multiplication-using-divide-and-conquer-algorithm/
/// @param str1
/// @param str2
/// @return string
string karatsuba(string num1, string num2) {
  int subLen = len / 2;
  string a = str1.substr(0, subLen);
  string b = str1.substr(subLen, len - subLen);
  string c = str2.substr(0, subLen);
  string d = str2.substr(subLen, len - subLen);
  string ac = karatsuba(a, c);
  string bd = karatsuba(b, d);
  // string acPbd = strAdd(ac, bd);
  // string adPbc = strAdd(karatsuba(a, d), karatsuba(b, c));
  string aPbcPd = karatsuba(strAdd(a, b), strAdd(c, d));
  string adPbc = strMinus(aPbcPd, strAdd(ac, bd));
  int bitShift = len - subLen;
  for (int i = 0; i < bitShift * 2; i++) {
    ac = ac + '0';
  }
  for (int i = 0; i < bitShift; i++) {
    adPbc = adPbc + '0';
  }
  result = strAdd(strAdd(ac, adPbc), bd);
}
```

There are many more annotations in the code too!

Part 3: Result & Verification

```
Welcome to simple calculator! Type exit to end calculate.
Please input your expression:
2432902008176640000
576777900467899500530800912469032805151182781285793507160527215050410603227371800685809141890937903834621421591438290396071318654590539851
837217246626474290254270413408646888479750197575461622436996166279491944773217205744466644054147786693046813986294713241065886012790343473
797749866791838315808215170441880497514730492142630950420494345309354394137827773251413112785726474035396496687709469407882843319421938325
764994683204692541775145210837688051597154558390369325409141751395233603593400451440326354285307992850967007096040561029247890807042876978
321332314390571938739377583715555027219292790913364006984932780044392718807661928373994664046511357826260245159824267582288844648538868700
380997936052309553208606689870039550601745769222758600008197102012027207655734518352085253132752668915997062667387629756577250041886461676
338971774165868315623888859490833720585506621362200036932039662187704426725647142881475232468208558283225230644588200037834429430957281027
921512511394354234528205656037325442251307421415646649151660357887478897024077398174449630579427182340795580179311336618274292371656616828
457415037228990363717976318568444785911215152715252872799961464244441790377411209903421866702510364513942638254602490318791735649176818423
723886576102237295495695280684758600914112888579325638496495712907610787337079218339410957546835100297120075275128841068178230171243011659
743505172919819256548447112024062302603532103521610731401260550972005788480165536531948243753112896058116318461642180712809831469618018375
221193986775006774367795056573431539238048347291369277624985942735636838884505137003966951358764415489231404926048120664231671694613518407
213466458839424301407033431076263893118990317751843070025603918368820913221591306219001412058442082206862238637959429998196692851740415106
392346212387740422178146576184532778999657942283496984547000743550388485526433560459164121878477882783203664281524791130536628957325502112
508097961253402109767552831579448636826153551879606697066225302526584245484967133673712107164565652435455707070723548816975812023603280593
695417747498584258414900836289203973654674047148441399202195726648278185917199662250982368932209049264321701406476412307367589545181318532
443682135568335615947224675306910639057320471295983465975835421301835359617678216500674034525181673667953214078788643037600640456080666402
633509866004281630053951685628421762358974229517105003931915533168745003592261196490338840477286992940609460955483907310896860418670339752
579389901917987412989357787619729307738013069060234986739271938565613760607406846275822007501465507896262354165433514828232155494328482394
701785244952134268589452832639571237871741353214462783141845053278607049077633437672372103761500958358922778565002539431295513609220056587
997852983419153349749143988211300428989697686221197170468638769628107111308593897399418192022441521803197571595666766327741699442049523180
```

986154358959745315297629008214965401858748635037736743909504192808475323842293844297984564763062044857329043067060575906713730924<mark>2007</mark>96094 sart(123,456) 11.111076 exit Calculator exit... "/usr/bin/gdb" --interpreter=mi --tty=\${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-xz5auxup.30w" 1>"/tmp/Micr [1] + Done osoft-MIEngine-Out-4uynlwhp.xxu" root@DESKTOP-VH54EV2:~/workSpace/proj2#

Some functions still lack support for precision, scientific notation or decimals.

Part 4 - Difficulties & Solutions

1. In order to ensure precision and no data loss, using string as an operator operation object requires rewriting the basic operation logic.

- 2. Code analysis somtimes shows that there are still riskes to be resolved: bunch of "Potential leak of memory", but I didn't use any new in my functions.
- 3. List in C++ is implemented in the form of linked list, which is different from other programming languages.
- 4. C++'s requirement that functions have only one return value forced me to use pointers and references.
- 5. Implementing support for scientific notation and decimals for various operators is more complicated than I thought.