Homework: String Calculator

# Overview

Bjarne is jealous of Python’s support for arithmetic with unbounded integers. Help Bjarne to feel better about C++ by building a calculator to do basic arithmetic operations with arbitrarily large numbers represented internally as strings, i.e., sequences of characters.

## An example execution of the calculator

String Calculator

"q" or "quit" or ctrl+d to exit

>> 1 + 1

ans =

2

>> 1371195958099968000 \* 191898783962510625

ans =

263130836933693530167218012160000000

>> 8222838654177922817725562880000000 - 263130836933693530167218012160000000

ans =

-254907998279515607349492449280000000

>> quit

farvel!

# 

# Objectives

1. Practice using C++ string objects
2. Practice problem-solving
   1. Implement standard arithmetic algorithms
   2. Create or discover novel algorithms
3. Read the header file

# Starter Code

The starter code on Mimir contains three files:

1. main.cpp := contains the user interface that drives the program
2. string\_calculator.h := contains required function declarations, read it.
3. string\_calculator.cpp := contains function definitions

**Read the Header File**

# Requirements

1. The source files to submit are named
   1. main.cpp
   2. string\_calculator.h (you should read this file)
   3. string\_calculator.cpp
2. You are required to implement the following functions, which are declared and described in the string calculate header file, which you should read.
   1. **unsigned int digit\_to\_decimal(char digit)**

digit\_to\_decimal(‘7’)should return 7

digit\_to\_decimal('/') should throw std::invalid\_argument

* 1. **char decimal\_to\_digit(unsigned int decimal)**

decimal\_to\_digit(7) should return ‘7’

decimal\_to\_digit(36) should throw std::invalid\_argument

* 1. **std::string trim\_leading\_zeros(std::string num)**

trim\_leading\_zeros(“00123”) should return 123

trim\_leading\_zeros(“-00123”) should return -123

* 1. **std::string add(std::string lhs, std::string rhs)**

add(“1”, “2”) should return “3”

add(“1”, “-2”) should return “-1”

add(“-1”, “2”) should return “1”

add(“-1”, “-2”) should return “-3”

* 1. **std::string subtract(std::string lhs, std::string rhs)**

subtract(“1”, “2”) should return “-1”

subtract(“1”, “-2”) should return “3”

subtract(“-1”, “2”) should return “-3”

subtract(“-1”, “-2”) should return “1”

* 1. **std::string multiply(std::string lhs, std::string rhs)**

multiply(“2”, “3”) should return “6”

multiply(“2”, “-3”) should return “-6”

multiply(“-2”, “3”) should return “-6”

multiply(“-2”, “-3”) should return “6”

1. You may use the following includes:
   1. iostream
   2. limits
   3. sstream
   4. string
   5. string\_calculator.h
2. The program must compile without warnings or errors.

g++ -std=c++17 -Wall -Wextra -pedantic -Weffc++ string\_calculator.cpp main.cpp

1. The program must run without errors or unhandled exceptions.

(user input in **bold red**; everything else is output)

⎵ is a space character (‘ ’)

⮒ is a newline character (‘\n’, also displayed as ↵)

$⎵./a.out

String⎵Calculator⮒

"q"⎵or⎵"quit"⎵or⎵ctrl+d⎵to⎵exit⮒

>>⎵**1⎵+⎵1⮒**

⮒

ans⎵=⮒

⮒

⎵⎵⎵⎵2⮒

⮒

>>⎵**quit⮒**

⮒

farvel!⮒

⮒

$⎵

# Recommendations

1. Read the header file.
2. Use descriptive (long) naming conventions for variables and functions.
3. Add comments to the code to describe anything which is not obvious from the code.
4. Use whitespace (indentation, newlines) to visually organize code.
5. Use functions to reduce code duplication and increase abstraction.

# Getting Started

1. Start early.
2. Download the starter code.
3. Read the header file.
4. Compile and run the program.
   1. It won’t do anything, but it also won’t crash
5. Submit it to Mimir.
6. Pick a test and a function to implement first and implement it just enough to pass the test
   1. Recommendations: decimal\_to\_digit, digit\_to\_decimal, trim\_leading\_zeros (small, positive)
   2. Plan your program on paper (digital or analog) before mashing the keyboard
7. Recompile and rerun.
   1. Check for errors.
   2. If no errors, move on
   3. Else, start debugging
8. Resubmit to Mimir.
   1. If the target test is passing, move on
   2. Else, start debugging
9. Continue by pickling a new test and writing just enough code to pass it (step 4)

# 

# Submission

The source files to submit are named

1. main.cpp
2. string\_calculator.h
3. string\_calculator.cpp

# Have fun!

# 

# Extra Challenges

Allow single value inputs and empty lines:

>> 123

ans =

123

>>

>>

>>

Remember the answer:

>> 2 \* 3

ans =

6

>> ans

ans =

6

>> ans \* 4

ans =

24

>> 5 \* ans

ans =

120

>>

Perform input validation:

>> x + y

Invalid LHS

>> 6 + x

Invalid RHS

>>

Implement quit on EOF (ctrl+D):

>> <ctrl+D>

farvel!

$

Support operations in any base in [2, 36]:

>> dead\_16 \* c0de\_16

ans =

a7c2da06\_16

>>