Software Development Fundamentals (SDF) Project - (CS1023)

Jan-May 2024

Problem Statement

Implementation of an entire infinite-precision arithmetic library in **C++** (version 17.0.0) using OOP concepts. Given two strings as input, you need to add support for addition, subtraction, multiplication and division, for both integer and floating point data types. The answer should be printed on the terminal screen.

Library specification

Create a namespace in C++ called **InfiniteArithmetic** which consists of two classes **Integer** and **Float.**

Integer class:

Your class should support the following constructors/destructors:

Default constructor Integer() that initializes the instance with value 0.
Constructor Integer(string s) that initializes the instance by the number whose string representation is given by 's'. Eg: Integer("-34534536454");
Copy constructor that creates an instance of Integer
Destructor that explicitly frees the memory.

Your class should support the following functions:

parse(const string& s) - a static function that returns an instance of **Integer** class.
Overload the following integer arithmetic operators: (+, -, *, /)

Float class:

Your class should support the following constructors/destructors:

Default constructor Float() that initializes the instance with value 0.f.
Constructor Float(string s) that initializes the instance by the number whose string representation is given by 's'. Eg: Float("-345.32393298");
Copy constructor that creates an instance of Float
Destructor that explicitly frees the memory.

Your class should support the following functions:

parse (const string& s) - a static function that returns an instance of **Float** class.
Overload the following integer arithmetic operators: (+, -, *, /)

Note:

Using these classes, one should be able to compute any expression involving arbitrarily large *integers* and arbitrarily large *float* numbers with arbitrary precision. All the operations that you implement should preserve the complete information of the number and should not introduce any kind of round-off errors.

Expected Binaries

libmy_inf_arith.a
my_inf_arith

Makefile instructions

It should have at least these targets.

make all: To build my_inf_arith executable.
make libmy_inf_arith: To build the static library (libmy_inf_arith.a).
make clean: To clean the temporaries/executables.

Two Modes

As an Executable:

The my_inf_arith executable should accept 4 command line arguments in the given order.

<int/float>
<add/sub/mul/div>
First operand

Second operand

As a Library:

The libmy_inf_arith.a library which can be linked to any executable/library.

Evaluation Criteria (Total 100 marks)

Stage	Domain	Operations	Points
1	Integers	+, -	45
2	Integers	*, /	20
3	Floating point	+, -	20
4	Floating point	*, /	15

Submission guidelines

Here are some general guidelines that you should follow: (All of these factors will have weights in the grading of this project)

Git:

Use git commands to upload the assignment in GitHub classroom
Assign git tags for each stage of the project.

Code:

The code should be properly organized: what goes into each classes, functions, blocks etc
The code should have appropriate separation between the source files and headers, etc.
All the code should be properly documented. This includes
proper variable names, proper indentation, …
documentation (at local level, function level, class level).

Report: Submit a report.pdf written in LaTeX describing your approach. It should clearly write the names and IDs of the team. It should outline at least the following:

The design section, which describes the design decisions that you made. It should also include the class diagrams in UML format.
A README section that explains to the users how to use your library.
A snapshot of the git commits.

Any limitations of your library.
Verification approach
Any key learnings from the overall project.
…

Public Test Cases :

Input: ./my_inf_arith int add 236500782249129494973109332402504293978326246711379838638440149 8
Output: 665898614873800632956973176417 48
Input: ./my_inf_arith int sub 3116511674006599806495512758577577452423003463811444464538840 08
Output: -546287306263397813371950941125431
Input: ./my_inf_arith int mul 1434416316044592994268069731232223017167694823904478474013730519
Output: 330162008905899217578310782382075660760972861550182008086155118
Input: ./my_inf_arith int div 879272636528306057983395052167721149383525361708964745499835 8
Output:
17804979
Input: ./my_inf_arith float div 879272636528306057983395052167721149383525361708964745499835 8.0
Output:
17804979.091469989302961159520087878533680521806036711679246839841308153986092362737855068303554024486248084164734360677209716426077098087105970442196627540469031041948493652479229112208111436434654956
Input: ./my_inf_arith float add 84486723.420039 70974199.843732
Output: 155460923.263771
Input: ./my_exe float sub 840196454.51725 712586963.70283
Output: 127609490.81442
Input: ./my_exe float mul 6400251.9377695 2326541.6827934
Output: 14890452913599.9717457253213
Input: ./my_inf_arith float div 244727.15202 75964.3891
Output:
3.22160363453775211100855150561593866618747020240303624056920112847981818364942264769690618100422530 74 (1000-digit precision)
Input : ./my_inf_arith int div 25 123
Output : 0
Input : ./my_inf_arith float div 3227 555
Output : 5.8144144… (repeating till 1000 digits)
Input : ./my_inf_arith float div 5.5 2
Output : 2.75
Input : ./my_inf_arith int div 2 0

Output : Division by zero error