

Лабораторная работа 4

Длинная арифметика

Цель работы

Изучить особенности работы классами и операторами в C++.

Стандарт языка

C++17 и новее.

Описание

В программе должен быть реализован класс `LN`, позволяющий выполнять арифметические операции над целыми числами произвольной точности в десятичной системе. С его использованием должна проводиться работа с данными.

Аргументы программе передаются через командную строку:

`src2 <имя_входного_файла> <имя_выходного_файла>`

Входной файл содержит выражение в форме обратной польской записи. Каждое число и знак операции ('+', '-', '*', '/', '%' – остаток от деления, '~' – квадратный корень, '_' – унарный минус, '<', '<=', '>', '>=', '==', '!=') располагается на отдельной строке. Каждая строка оканчивается символом новой строки.

Выходной файл должен содержать состояние стека на момент завершения работы программы. Каждое значение находится на новой строке, начиная с вершины, строка оканчивается символом новой строки.

Результат операций сравнения: 0 (для false) или 1 (для true). Число -0 должно быть полностью эквивалентно 0 (включая вывод).

Реализации операций умножения (оператор '*'), деления (оператор '/'), остаток от деления (оператор '%') и квадратный корень ('~') должны работать с адекватной скоростью (то есть требуется алгоритм уровня “в столбик”, а не “умножение на n путём сложения n раз”).

Квадратный корень и деление округляют к 0. Результат при взятии корня из отрицательного числа или делении на ноль: NaN. Результат арифметических действий и сравнения с NaN: в соответствии со стандартом IEEE-754.

Корректность входных данных гарантируется.

Пример входных и выходных данных:

input	output
2	1
3	2
2	
+	
6	
-	
-1	
==	

Класс должен иметь конструкторы:

- из long long со значением по умолчанию: 0;
- из строки символов C (const char *);
- из строки std::string_view;
- конструктор копирования;
- конструктор перемещения.

Для класса должны быть реализованы операторы:

- оператор копирующего присваивания;
- оператор перемещающего присваивания;
- арифметические: +, -, *, /, %, ~, - (унарный);
- комбинация арифметических операций и присваивания (например, +=);
- сравнения: <, <=, >, >=, ==, !=;
- преобразования типа в: long long (с генерацией исключения в случае, когда значение не помещается), bool (неравенство нулю);
- создания из литерала произвольной длины с суффиксом _ln (например, должно работать выражение: LN x; x = 123_ln;).

Необходимо реализовать вспомогательные функции/методы (сложение, вычитание, универсальное сравнение (возвращает -1, 0, 1)), которые будут использоваться в функциях/методах: оператор '+', оператор '<' и пр. Например, при реализации и оператора '+', и оператора '-' необходимо выполнять и сложение, и вычитание, в зависимости от знаков входных чисел.

Реализация по файлам

Объявление класса необходимо поместить в заголовочный файл LN.h, реализация крупных методов (больше 1 строки) класса должна быть в LN.cpp. Код функции main располагается в main.cpp.

Заголовочные файлы должны быть с защитой от повторного включения.

Создавать свои файлы со вспомогательным кодом не запрещено.

Работа с памятью

Нехватка памяти при операциях с классом должна обрабатываться через исключения C++.

Кол-во перевыделений памяти в конструкторах и операциях не должно быть порядка длины числа (или больше...).

Использовать **STL** внутри класса **LN** не запрещено, НО оценивается в -5 баллов. Соответственно, можно получить отрицательное кол-во баллов за эту работу в целом.

Рекомендация по решению

В классе отдельно хранить знак числа и отдельно разряды числа, при этом разряды рекомендуется хранить массивом, начиная с младшего (`digits[0]` соответствует самой младшей цифре числа).

Требования к программе

Программа должна:

1. быть написана на C++ по заданному стандарту;
2. выполнять поставленную в ТЗ задачу;
3. не использовать внешние библиотеки;
4. всегда корректно освобождать память;
5. всегда корректно закрывать файлы;
6. обрабатывать ошибки:
 - a. файл не открылся;
 - b. не удалось выделить память;
 - c. на вход передано неверное число аргументов командной строки.

В этих случаях необходимо выдавать сообщение об ошибке и корректно завершаться с ненулевым кодом возврата (см. `"return_codes.h"`);

7. никогда ничего не писать **в поток вывода**;
8. выводить сообщения об ошибке в **поток вывода ошибок**.

Ограничения

1. Запрещено использование `exit(...)` в коде.
2. Запрещено использовать VLA (как массивы, так и указатели), т.к. его нет в стандарте C++.
3. Ограничивается использование глобальных переменных (кроме констант) - необходимость их использования вы должны обосновать на защите. Ваш код должен быть максимально приспособлен к переносимости в другие проекты и/или использованию другими разработчиками.
4. Запрещается подключать системные библиотеки через `#include "..."`.
5. Запрещается использовать `setlocale(...)`. Учимся писать небольшие комментарии пользователю по-английски.
6. Запрещается использовать `system("pause")`.
7. Если вы создаёте свои макросы, то их название не должно быть `"DEBUG"`, `"_DEBUG"`, `"NDEBUG"` и прочие, определяемые компиляторами имена. Допустимы любые другие названия из заглавных букв и символов подчёркивания.

Полезные check-листы и ссылки:

- [Проверки](#)
- [Отправка на GitHub](#)
- [Про критерии оценивания](#)
- [Суперпопытка](#)

Особенности сдачи работы

[Фича 1] Проверка отправок до дедлайна

Если вы уверены, что не хотите больше вносить изменений в свой код и подготовили его раньше, чем за 3 дня до дедлайна, то можете рассчитывать на досрочную проверку. Для этого нужно написать проверяющему в лс с ссылкой на репозиторий. Важный момент: если у вас не проходят тесты или не собирается решение, то досрочно работа проверяться не будет.

[Фича 2] Примеры тестов

Примеры входных файлов также можно посмотреть в репо `.github/workflow/in*.txt`

Любое изменение файлов в каталоге `.github/workflow` строго запрещено. Если будет замечено, что файлы из этого каталога модифицировались, то отправка проверяться не будет и дедлайн засчитан также не будет.

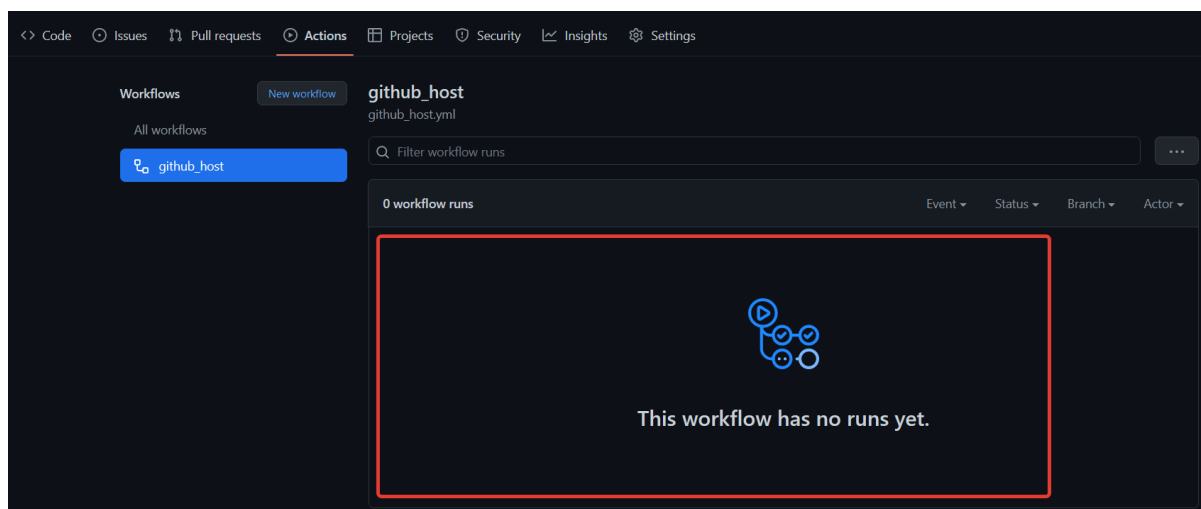
[Фича 3] Частичная автопроверка

При создании PR с dev на main будет запускаться скрипт автопроверки, который позволит посмотреть, собирается ли ваш код в принципе, соответствует ли оформлению clang-format и проходит ли простые тесты.

Это не отменяет проверку проверяющим. Время окончания проверки никак не влияет на то, успели ли вы в дедлайн или нет.

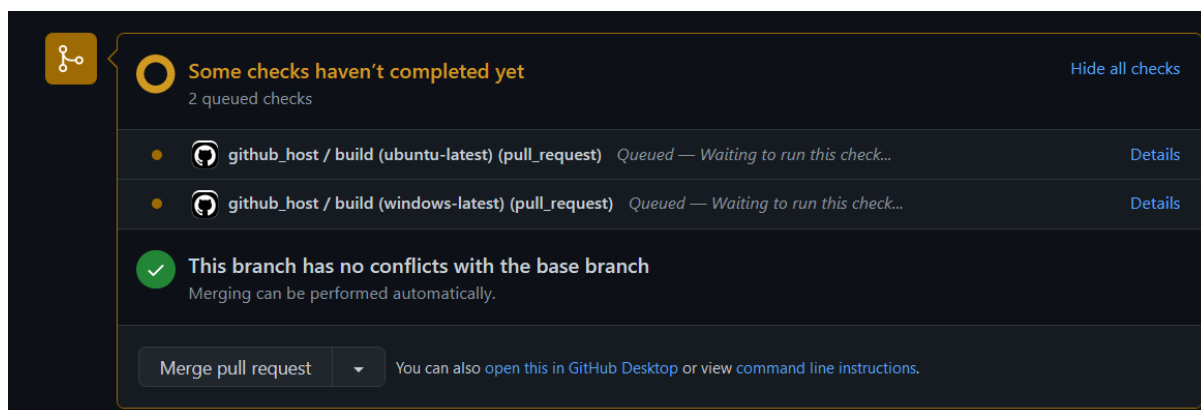
Важно: даже если автопроверка завершилась с кодом 1 и вы видите красный крестик, это не значит, что ваша работа не будет проверена проверяющим, если она отправлена в срок и отправка соответствует требованиям (сдача в срок, PR называется верно, стоит reviewer).

Во вкладке Actions на Github можно посмотреть свои запуски. Они будут отображаться списком в месте, выделенным красной рамкой. Также каждый запуск можно посмотреть над кнопкой Merge request определенного Pull request.

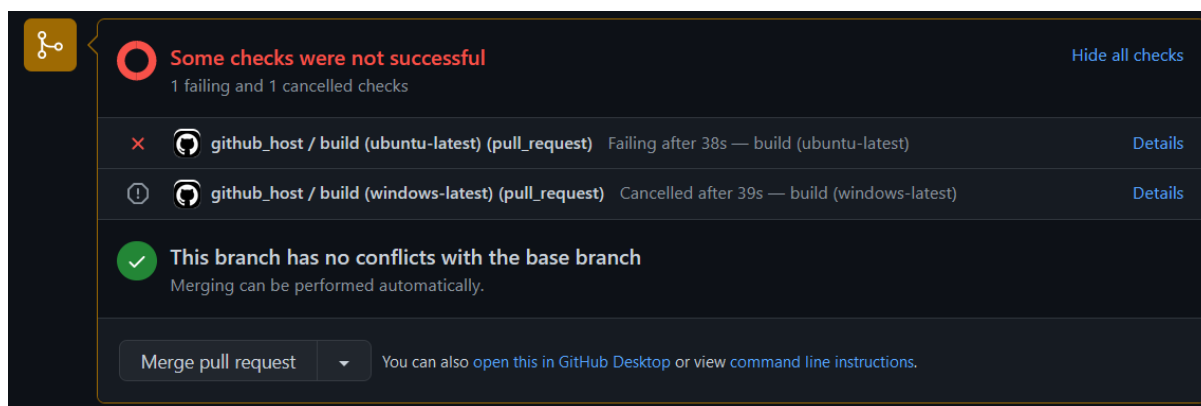


Запуск автопроверки происходит при каждом создании (open) или переоткрытии (reopen) PR с dev на main.

Каждый раз будет запускаться проверка под двумя системами - Ubuntu 20.04 и Windows Server 2022.

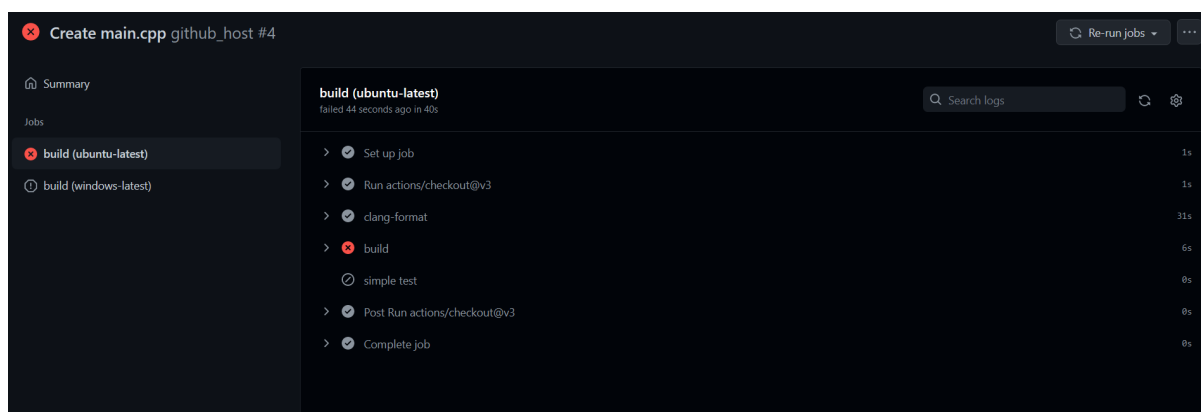


В случае, если одна из проверок завершилась с ненулевым кодом, то вторая автоматически прерывается.



При изменении кода в исходниках на dev статус проверки во вкладке Pull request пропадет, но его всё ещё можно посмотреть в Actions.

При нажатии на кнопку Details вы попадаете в лог проверки (также можно туда попасть через раздел Actions).



Этап “clang-format” всегда будет отмечен галочкой, однако в логах в случае несоответствия форматирования будет сообщение об этом (см. ниже). Оценка за clang-format будет выставляться по данному сообщению.


```
clang-format 38s
1 ▶ Run if ('ubuntu-latest' -eq 'windows-latest')
18 --2022-05-26 00:18:18-- https://apt.llvm.org/llvm-snapshot.ppg.key
19 Resolving apt.llvm.org (apt.llvm.org)... 151.101.50.49, 2a04:4e42:c::561
20 Connecting to apt.llvm.org (apt.llvm.org)[151.101.50.49]:443... connected.
21 HTTP request sent, awaiting response... 200 OK
22 Length: 3145 (3.1K) [application/octet-stream]
23 Saving to: 'STDOUT'
24
25      0K ...                               100% 76.3M=0s
26
27 2022-05-26 00:18:18 (76.3 MB/s) - written to stdout [3145/3145]
28
29 Formatting [1/1] /home/runner/work/cpp2_template/cpp2_template/main.cpp
30 /home/runner/work/cpp2_template/cpp2_template/main.cpp:4:2: warning: code should be clang-formatted [-Wclang-format-violations]
31 {
32 ^
33 /home/runner/work/cpp2_template/cpp2_template/main.cpp:5:10: warning: code should be clang-formatted [-Wclang-format-violations]
34     x = &i;
35         ^
```

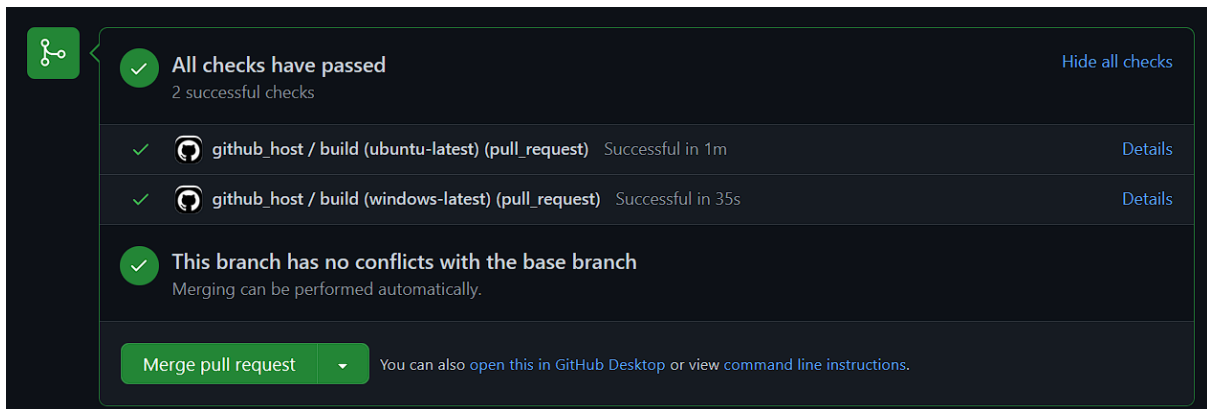
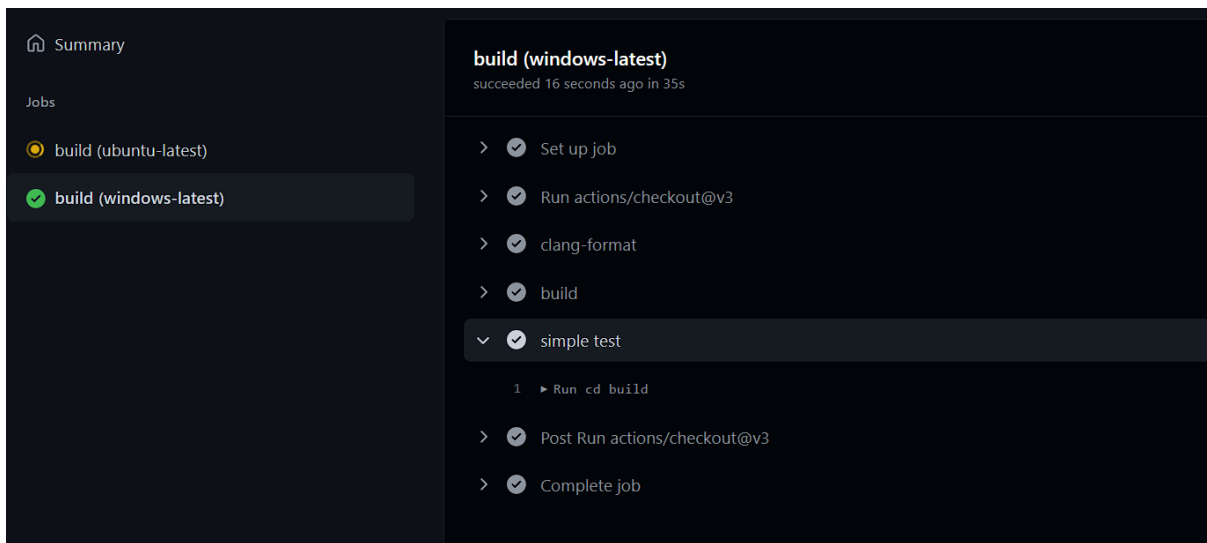
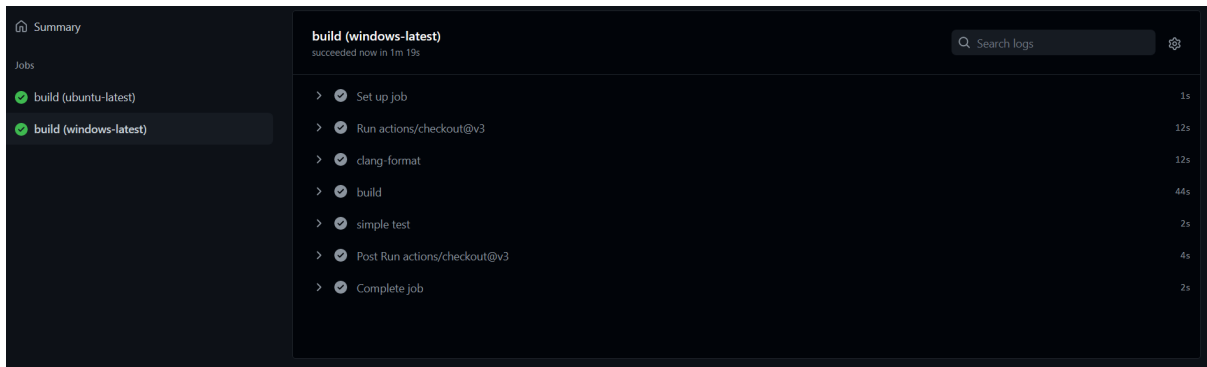
На этапе “build” происходит попытка сборки вашего решения из исходников, найденных в репозитории. Лог сборки можно найти в раскрывающейся вкладке.

```
build 6s
15 ▶ Run Remove-Item 'build' -Recurse -ErrorAction SilentlyContinue -Force
14 /home/runner/work/cpp2_template/cpp2_template/main.cpp:5:3: error: use of undeclared identifier 'x'
15     x = &i;
16     ^
17 /home/runner/work/cpp2_template/cpp2_template/main.cpp:5:7: error: cannot take the address of an rvalue of type 'int'
18     x = &i;
19     ^~
20 2 errors generated.
21 Error: Process completed with exit code 1.
```

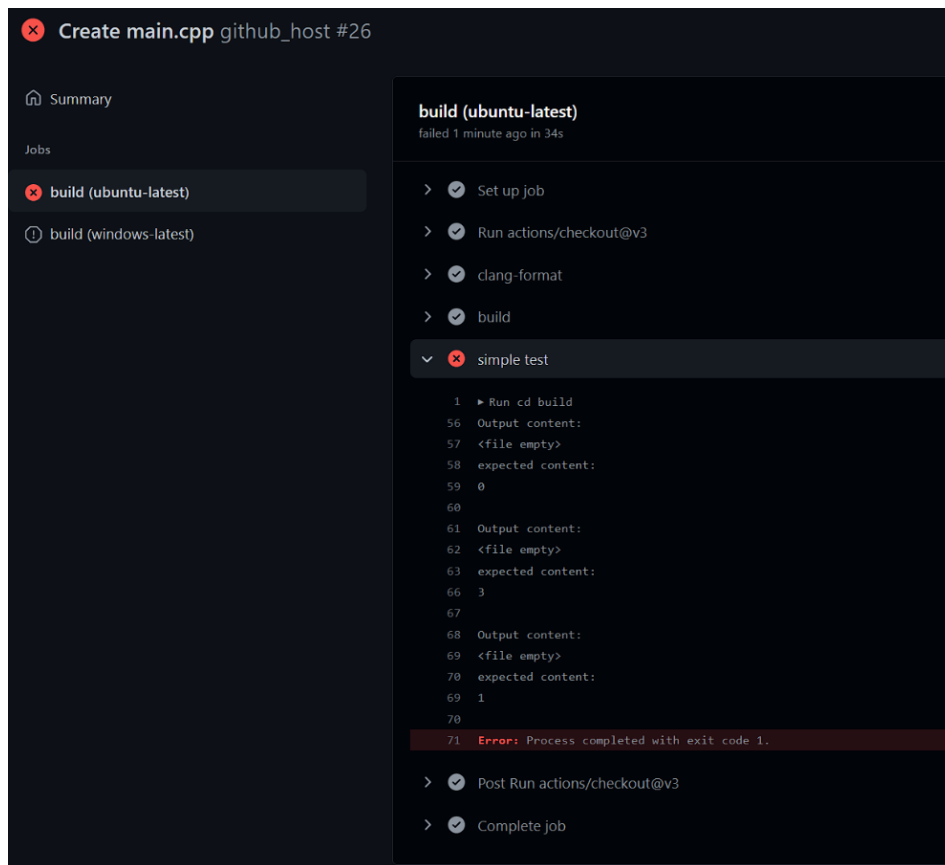
Если сборка не прошла, то вы увидите красный крестик и тесты запущены не будут.

Иначе всё отлично и собранное решение будет прогоняться на паре тестов на сложение, сравнение и унарный минус.

Если все тесты пройдены, то workflow завершиться с кодом 0 и статусом Success.



В противном случае в логах тестов будут комментарии, почему тест не прошел.



Злоупотреблять автопроверками крайне не советую, потому что в месяц на всех выделяется определенное кол-во минут, которое может кончиться.