Калькулятор выражений

Весна 2020,
с изменениями от 13.05.2019

В качестве курсового проекта требуется написать программу — калькулятор, умеющий вычислять значение арифметических выражений, которые задаются пользователем в виде строки текста: "3 * 2.5 / (2 + 2 * 7)".

Желтым маркером помечаются изменения в тексте условия по сравнению с первоначальным вариантом. Иногда приходится уточнять задание в течение семестра после получения обратной связи. Добавлены и изменены пункты: **2.5.**

Программы получаются по почте и проверяются полуавтоматически при помощи глупого робота, поэтому существует внушительный набор **обязательных** формальных требований, которых необходимо придерживаться. В противном случае ваша программа может быть просто не замечена или забракована. Заодно проверяется ваша способность работать на "заказчика", выполняя его предписания.

Робот последовательно проводит серию тестов (разбитых на стадии), отмечая результат прохождения в таблице и намекает на ошибки:

- **S0** (stage00): проверка формата письма, архива с исходными файлами, самих файлов на предмет запрещенных конструкций, компилирование;
- **S1** (stage01): короткая проверка реакции приложения на входные данные, без сверки ответов (15+ выражений);
- **S2** (stage02): проверка реализации базовых арифметических операций, со сверкой ответов (150+ выражений);
- **S3** (stage03): более серьезный автоматический тест на различных выражениях, проверка реакции приложения на входные данные, со сверкой ответов (ещё 200+ выражений).
- **S4** (stage04): дополнительные проверки на поведение в нестандартных ситуациях (нехватка памяти и т.п.).
- **S5** (stage05): проверка выражений с заданиями на отлично.

Далее идёт общение с преподавателем, исправление кода по замечаниям, проверка задания на отлично и выставление оценки.

Все работы по курсовику разбиты на части (этапы), которые нужно проходить один за другим, точно в срок. На сегодняшний момент выделены следующие промежуточные этапы до сдачи финальной версии проекта:

- **M0** (до даты второй лекции, 00:00): на адрес **робота** написано приветственное письмо, автоматически проходится S0, т.е. это письмо получено роботом, его формат понят, присланная в архиве минимальная программа на Си скомпилирована без ошибок, вами получен ответ.
- **M1** (до 1 марта, 00:00): автоматически проходится S1, т.е. сделана интерфейсная часть без вычислительного модуля (пп.1, 2.1 задания).
- **M2** (до 1 апреля, 00:00): автоматически проходится S2, т.е. сделан базовый вычислительный модуль (пп.3.2.1, 3.3.2, 3.3.3, 3.5, 3.6).
- **M3** (до 1 мая, 00:00): автоматически проходится S3, т.е. все контрольные выражения считаются корректно, корректно определяются ошибочные

выражения.
- **RC** (до 0:00 пятницы зачётной недели): автоматически проходится S4, возможно S5, уже пройден этап “ручной” проверки, защиты курсовой преподавателю, сдачи отчёта. Ведомость на курсовую работу заполняется в пятницу, положительная оценка выставляется при наличии всех необходимых пунктов.

Программу не имеет смысл показывать роботу и преподавателю, если не выполнен хотя бы п.1 полностью и обязательно п.2.1 полностью, с подпунктами (вас всё равно никто не поймёт). Остальные пункты можно выполнять постепенно, разумеется, **не злоупотребляя количеством итераций сдачи**.

Не следует использовать робота в качестве бесплатного низкосортного тестировщика. Робот является своего рода ЭКЗАМЕНАТОРОМ, к которому следует приходить хорошо подготовившись, минимизируя количество встреч.

Злоупотребление роботом приведёт к ужесточению условий, уменьшению времени, когда он доступен. Каждое следующее письмо должно **существенно отличаться** от предыдущего (см. п.1.4.5).

При выявлении случая плагиата на любой стадии сдачи сразу же выставляется оценка “неудовлетворительно” без возможности пересдачи данного проекта.

1. Требования к форме сдачи

- **1.1.** Язык программирования: ANSI C, стандарт 1989 г., плюс дополнения Microsoft (напр., отладки crtdbg.h).
- **1.2.** Среда разработки: Microsoft® Visual C++® 2008/2010/2012/2013 Express или 2015 Community (робот использует **2015 Community**).
- **1.3.** Алгоритм: произвольный, т.е. а) итеративный поиск операндов и операций, вычисление и замена в строке, б) преобразование в обратную польскую запись с её вычислением, в) построение дерева выражений, г) собственное изобретение… Каждый из пунктов на самом деле является классом алгоритмов, так что количество реализаций огромно.
- **1.4.** Сдача: в несколько подходов в течение семестра по электронной почте с последующей очной защитой. К формату сдачи предъявляются определенные требования:
  - **1.4.1.** Тема письма должна начинаться с фразы “`calc:`”, за которой через пробел следует **фамилия** и **имя**, именно в этом порядке, записанные латиницей (единообразно от письма к письму). Например, “calc: Smirnov Pavel”. Больше ничего в теме быть не должно.
  - **1.4.2.** К письму должен быть приложен один архив формата ZIP, содержащий исходные файлы программы.
  - **1.4.3.** Архив должен содержать только исходные файлы программы (названные латиницей) без каталогов, подкаталогов и вспомогательных файлов. Допустимые расширения файлов: *.c, *.h. Примечание: файл проекта при компиляции всё равно не используется.

    - **1.4.4.** Программа должна компилироваться без ошибок и предупреждений вплоть до 4-ого уровня, что надо указать в свойствах проекта (Configuration Properties → C/C++ → General → Warning Level = 4 и Treat Warnings As Errors = True). Отключать предупреждения компилятора любыми способами запрещается, их нужно принимать во внимание и исправлять соответствующий код. Примечание: #define для библиотеки crtdbg необходимо указывать в настройках проекта, а не в самих исходных файлах.
    - **1.4.5.** Тело письма должно содержать непустой список существенных отличий этой версии от предыдущей (присланной предыдущим письмом). Никаких изменений наугад в коде быть не должно, только обнаруженные и исправленные ошибки, дополненная функциональность. Следует стараться минимизировать количество писем роботу, в идеале требуется пройти весь курсовик за 5-10 попыток.
- **1.5.** Исходные коды: только собственноручно написанные. Разрешается общаться, советоваться, обсуждать алгоритмы, но не обмениваться кусками кода или программами. *При выявлении случая плагиата на любой стадии сдачи сразу же выставляется оценка **"неудовлетворительно"** без возможности пересдачи данного проекта.* Соответственно, экзаменационная оценка будет выставляться исходя из максимальной "тройки", а для курсового проекта выдаётся другое задание.
- **1.6.** Отчёт: при наличии отдельной оценки в учебных планах оформляется в виде электронного документа (по ходу действия) и распечатки (в конце семестра при защите).
    - **1.6.1.** Формат электронного документа PDF (напр. LaTeX или экспорт из Microsoft Word).
    - **1.6.2.** Распечатка на листах A4 с одной стороны, скрепленная степлером.
    - **1.6.3.** Оформление в соответствии с действующим законодательством и нормативными документами
        - Положение по содержанию, оформлению, организации выполнения и защиты курсовых проектов и курсовых работ. — СПб.: СПбГПУ. — 2013: http://dmo.spbstu.ru/images/stories/Polozenie_o_KR_i_KP.pdf
        - Маслов, В. И. Правила оформления студенческих текстовых документов: дипломных (курсовых) проектов (работ), отчётов и рефератов [Электронный ресурс]: методические рекомендации / В.И. Маслов, Л.Н. Шуткевич; — СПб.: СПбГПУ. — 2013: http://elib.spbstu.ru/dl/2/2976.pdf
    - **1.6.4.** Содержание должно включать в себя: постановку задачи, описание алгоритма в целом и решение некоторых частных задач, см. шаблон на сайте курса.

2. Требования к тестовой программе

- **2.1.** Программа должна быть консольным приложением (Win32 Console Application), запускаемым из командной строки.
    - **2.1.1.** Если программа запущена без параметров, то входные данные должны читаться из стандартного потока, построчно, до конца потока. Примечание: в среде Windows в консольном приложении в интерактивном режиме сигнал конца потока с клавиатуры можно

передать в программу, если **в начале** строки ввода нажать комбинацию клавиш Ctrl+Z, ENTER; в режиме перенаправления ввода-вывода конец потока приходит автоматически.

- **2.1.2.** Если программа запущена с одним параметром (именем файла), то входные данные должны читаться из этого текстового файла, построчно, до конца (потока). Примечание: никакой программной разницы при чтении из стандартного потока и из файла нет, и то, и другое — потоки. Никакого дублирования кода не требуется, и оно не должно присутствовать в тексте программы.
- **2.1.3.** Если программа запущена без параметров, но с перенаправлением ввода и/или вывода, она по-прежнему должна работать и завершаться корректно, аналогично предыдущим двум способам запуска.
- **2.1.4.** Если параметры командной строки вас не устраивают по количеству или содержимому, программа должна вывести одну строчку, содержащую сообщение об ошибке. Она должна начинаться с фразы "ERROR:", за которой через пробел следует пояснение проблемы.
- **2.1.5.** Вывод должен осуществляться в стандартный поток вывода. Одной прочитанной строке должна соответствовать одна напечатанная, завершённая переводом строки. Никаких дополнительных подсказок, переводов строк для красоты и «пауз» быть не должно. Сколько строк было во входном потоке, столько должно быть и в выходном (легко проверяется с помощью перенаправления ввода и вывода).

  Примечание 1: строки входного потока должны разделяться, а не обязаны завершаться переводами строк; последнего перевода может и не быть на входе, но он должен быть на выходе.

  Примечание 2: в соответствии с общепринятой практикой финальный перевод строки в потоке не считается образующим новую пустую строку! Таким образом, “`2+2\n`” считается одной строкой на входе.
- **2.1.6.** Прочитанные пустые строки и строки комментариев должны выводиться без изменений. Пустой строкой считается строка, содержащая ноль или более пробельных символов (не только пробелов! ориентируйтесь на `isspace()`), расположенная не в конце файла. Строкой комментария считается строка, начинающаяся с двух косых черт “`//`” и содержащая произвольный текст. Строка может содержать пробельные символы до начала комментария, но не другой текст.
- **2.1.7.** Остальные строки должны восприниматься как выражения и сразу же вычисляться. Выводиться должна исходная формула без изменений и, в той же строке, численное значение результата (или краткое описание ошибки). Исходная формула от результата (или ошибки) должна отделяться двойным знаком равенства (==), обрамлённым пробелами.
- **2.1.8.** Результат должен выводиться с необходимой точностью. Требуемый формат вывода соответствует спецификатору "%g" (конечные нули не выводятся, при необходимости используется научная форма записи).
- **2.1.9.** При возникновении ошибки во время обработки выражения сообщение о ней должно выводиться вместо результата (после знака равенства). Оно должно начинаться с фразы "ERROR:", за которой через пробел следует пояснение проблемы. Работа программы при этом не

должна прерываться.
    - **2.1.10.** Ничего другого вводиться и выводиться не должно, чтобы программа могла работать в неинтерактивном (автоматическом) режиме.
    - **2.1.11.** Никакого явного или неявного ограничения на длину входной (и выходной) строки быть не должно. Программа должна корректно читать и обрабатывать строки любой длины.
    - **2.1.12.** Корректно работающая программа должна завершаться с кодом возврата 0 ("ноль", отсутствие ошибки). В случае возникновения фатальных ошибок, мешающих продолжению программы (п.2.1.9 к ним не относится), возможны ненулевые коды возврата. Пример: в пункте 2.1.4 разумно использовать ненулевой код возврата.
- **2.2.** Текст программы должен подчиняться выбранному преподавателем стилю кодирования (см. отдельную инструкцию), определяющему:
    - **2.2.1.** Форматирование исходного текста
    - **2.2.2.** Наименование переменных, констант, типов, функций
    - **2.2.3.** Комментарии к коду
- **2.3.** Должен соблюдаться принцип модульности: алгоритм должен быть логически и физически отделен от интерфейса с пользователем.
    - **2.3.1.** Вычислительный алгоритм должен быть оформлен в виде отдельной функции, принимающей на вход строку, в которой записано выражение (напр., “double calculate(char const * expression);”)
    - **2.3.2.** Эта основная функция должна заниматься одной задачей — вычислением выражения, и не должна ничего читать с клавиатуры или выводить на экран. Она может (и должна) быть разбита на вспомогательные подфункции, выполняющие отдельные подзадачи.
    - **2.3.3.** Эта основная и все вспомогательные функции должны быть помещены в отдельный модуль (модули) со своим правильно оформленным заголовочным файлом.
- **2.4.** Ошибки времени выполнения должны корректно обрабатываться.
    - **2.4.1.** Программа не должна "падать" (генерировать исключения) и "зависать" (попадать в бесконечный цикл).
    - **2.4.2.** Программа должна быть максимально защищена от неправильных входных данных ("защита от дурака"), ошибок ввода-вывода, а также от ошибок, возникающих в ходе вычислений (таких как деление на ноль).
    - **2.4.3.** Алгоритмический модуль должен корректно сообщать интерфейсному модулю о возникновении ошибки, а не приводить к аварийному завершению программы (напр., "`error_t Calculate(char const* expression, double* result);`" или "`double Calculate(char const* expression, error_t* error);`").
    - **2.4.4.** По тексту программы в правильных местах должны быть расставлены проверки на нарушение внутренней логики (assertions) для своевременного обнаружения ошибок в ваших алгоритмах (пред-/постусловия для функций и блоков команд).
    - **2.4.5.** Программа должна быть вами всесторонне протестирована на различных входных данных. Упростите жизнь, создайте себе файл с выражениями для проверки.
    - **2.4.6.** Программа должна использовать встроенные средства для отладки памяти (crtdbg.h), чтобы отследить возможные ошибки: утечки памяти (неосвобождённые блоки), пропись (переполнение буфера) и

т.п.

- **2.4.7. РАЗЪЯСНЕНИЕ:** Случаи нехватки памяти должны корректно обрабатываться как и другие ошибки времени выполнения. При нехватке памяти во время вычислений должно выдаваться сообщение об ошибке вместо результата (см. п.2.1.9). При нехватке памяти во время чтения входной строки должно выдаваться сообщение об ошибке ("ERROR: ...") вместо всего выражения, а входные данные корректно пропускаться до начала следующей строки (содержащей следующее выражение). Программа не должна аварийно завершаться, нехватка памяти для одного выражения не означает, что ее не хватит для остальных.

- **2.5. На отлично:** Программа должна проходить проверку специальным инструментом-линтером (PVS-Studio или аналогичным, по согласованию с преподавателем), который помогает обнаружить большое количество логических ошибок и неточностей. Большинство выявленных ошибок и нестабильностей (потенциальных ошибок) должно быть устранено (см. лекции), в первую очередь уровня High и Medium.

  Примечание: для бесплатной проверки при помощи PVS-Studio (на соответствующей стадии) ваши си-файлы должны начинаться с комментария определенного вида. Будет неплохо, если вы позаботитесь об этом заранее, даже если не собираетесь выполнять задания "на отлично".

  Примечание 2: робот запускает PVS-Studio из командной строки, и линтер ему выдаёт сообщения **обо всех** найденных ошибках, но не все доходят до вас — робот не считает критическими ошибки уровня > 2, а также игнорирует коды V1xx, V2xx, V3xx, V802. Обратите внимание, что расширение для Visual Studio может иметь **другие настройки по умолчанию** и фильтровать сообщения по-другому (скрывая от вам больше, в частности правила из набора MISRA).
- **2.6. По желанию:** Второй интерфейсный модуль, то есть второй вариант программы, работающий в виде оконного Windows-приложения с тем же алгоритмическим блоком.

3. Требования к функциональности расчётного модуля

- **3.1.** Выражения представляют собой набор операндов, разделенных знаками операций с соответствующими приоритетами и ассоциативностью, известными из математики и/или языка Си и других языков. Примечание: есть много онлайн-калькуляторов, начиная с google.com, с которыми можно и нужно сравниваться. Спорные моменты решаются преподавателем.
  - **3.1.1.** Никакого явного или неявного ограничения на сложность выражения быть не должно за исключением нехватки оперативной памяти (тем не менее, см. п.2.4). Алгоритм должен корректно и быстро читать и обрабатывать выражения любой сложности (как и строки любой длины).
- **3.2.** Операндами могут являться:
  - **3.2.1.** Целые числа (литеральные константы).
  - **3.2.2.** Вещественные числа (литеральные константы), в том числе и в научной (экспоненциальной) форме записи. В формате записи констант следует ориентироваться на стандарт языка Си, см. например, http://msdn.microsoft.com/en-us/library/w9bk1wcy.aspx

    - **3.2.3.** Именованные константы **`pi`** и **`e`**, заданные с достаточной (максимально возможной) точностью.
- **3.3.** Должны поддерживаться основные арифметические операции:
    - **3.3.1.** Унарный минус (–a). Не требует круглых скобок для корректного вычисления нескольких минусов подряд, т.к. порядок вычислений определяется однозначно: `1+---2` = –1.
    - **3.3.2.** Бинарное сложение (a+b), вычитание (a–b), умножение (a*b), деление (a/b) и возведение в степень (a^b), которое имеет ~~наивысший приоритет и~~ ассоциативность справа. С точки зрения приоритета наблюдается не сразу очевидное поведение степени по отношению к унарному минусу: `-2^2` = –4.
    - **3.3.3.** Круглые скобки для группировки (переопределения приоритета). Скобки не являются обязательными в том случае, когда порядок применения операций очевиден и однозначно определён.
- **3.4.** Кроме операций формулы могут содержать вызовы функций вида "sin(5)" (своего рода унарная операция). Необходимость скобок вокруг аргумента в задании не специфицируется, решение принимается исполнителем. Должны поддерживаться следующие функции:
    - **3.4.1.** корень (sqrt);
    - **3.4.2.** тригонометрические (sin, cos, tg, ctg, arcsin, arccos, arctg);
    - **3.4.3.** натуральный логарифм (ln);
    - **3.4.4.** округление вниз (floor) и вверх (ceil).
- **3.5.** Вычисления должны проводиться с точностью типа double.
- **3.6.** Формулы могут содержать любые пробельные символы (не только пробелы!), не мешающие вычислениям (свободная форма записи выражения), то есть, не разбивающие лексемы на части.
- **3.7. На отлично:** функция логарифма по произвольному основанию “log(a,x)” от двух аргументов.
- **3.8. На отлично:** сохранение промежуточных результатов во временных переменных и использование их в качестве операндов. Это подразумевает последовательное вычисление нескольких выражений, перечисленных через точку с запятой, с возвращением значения последнего из них. Пример такого выражения: "`x = 2 * 2; y = 1 + 2; x^y + x`", его значение равно 68. Значения временных переменных не сохраняются и не могут быть использованы в следующих выражениях.

  Примечание: для успешной сдачи достаточно реализации однобуквенных переменных, если сложно обеспечить полноценные идентификаторы.
- **3.9. По желанию:** Сохранение результатов в глобальных переменных для использования в последующих выражениях. Для того, чтобы отличать от временного присвоения, следует использовать знак ":=". Пример такого выражения: "`x := 2 + 2`".
- **3.10. По желанию:** Возможность определения собственных функций от одного или нескольких параметров, для использования в последующих выражениях. Пример такого определения: "`sqr(x) := x * x`".
- **3.11. По желанию:** Поддержка шестнадцатеричных и двоичных ("**`0x`**`A0C6`", "**`0b`**`101001`") констант в качестве операндов.

См. примеры диалога на следующей странице.

Пример

Вводимое пользователем выделено **цветом и жирным шрифтом**.

Перенаправление ввода-вывода

```
C:\> calc.exe < input.txt > output.txt
```

Чтение из файла

```
C:\> calc.exe input.txt > output.txt
```

input.txt	output.txt
2+2 **2*(3-1)** **// hello, world** **2+(1+** **2+2+2+2+2+2+2+2+2+2** **2+2** **1+1**	2+2 == 4 2*(3-1) == 4 // hello, world 2+(1+ == ERROR: brackets ERROR: not enough memory 2+2 == ERROR: not enough memory 1+1 == 2

Интерактивный режим

```
C:\> calc.exe
```

stdin+stdout
2+2 2+2 == 4 **2*(3-1)** 2*(3-1) == 4   **// hello, world** // hello, world **2+(1+** 2+(1+ == ERROR: brackets **2+2+2+2+2+2+2+2+2+2** ERROR: not enough memory **2+2** 2+2 == ERROR: not enough memory **1+1** 1+1 == 2 **^Z** ← *примечание: это не степень; так отображается нажатие Ctrl+Z*