**Курсовая работа**

**по дисциплине: «»**

**«»**

2019 г

# Задание

Вариант 2

Написать стековый калькулятор, который принимает в качестве аргумента командной строки имя файла, содержащего команды. Если аргумента нет, то использовать стандартный поток ввода для чтения команд. Использовать вещественные числа.

Реализовать следующий набор команд:

# – строка с комментарием.

POP – снять число со стека.

PUSH – положить число на стек.

+, -, \*, /, SQRT – арифметические операции. Используют один или два верхних элемента стека, измают их из стека, помещая результат назад.

PRINT – печать верхнего элемента стека (без удаления).

DEFINE – задать значение параметра. В дальнейшем везде использовать вместо параметра это значение.

Пример (должно вывести 2):

DEFINE a 4

PUSH a

SQRT

PRINT

Оглавление

[Задание 2](#_Toc10900450)

[Введение 4](#_Toc10900451)

[1 Язык программирования C# 6](#_Toc10900452)

[2 Среда разработки Visual Studio 7](#_Toc10900453)

[3 Исходные данные и способ их ввода 8](#_Toc10900454)

[3.1 Ввод данных с клавиатуры. 8](#_Toc10900455)

[3.2 Ввод данных из файла. 8](#_Toc10900456)

[4 Описание реализованных классов 9](#_Toc10900457)

[5 Тестирование приложения 11](#_Toc10900458)

[Заключение 20](#_Toc10900459)

[Список литературы 21](#_Toc10900460)

[ПРИЛОЖЕНИЕ А. Исходный код программы 22](#_Toc10900461)

[ПРИЛОЖЕНИЕ B. Содержимое тестовых командных файлов 35](#_Toc10900462)

# Введение

В настоящее время, с развитием научно-технического прогресса и информационных технологий, сложность автоматизируемых предметных областей постоянно и неуклонно возрастает. Программист проводит всё своё рабочее время, по сути, в борьбе со сложностью программ, отлаживая свой код в поисках допущенных ошибок. Ради борьбы со сложностью эволюционируют языки программирования. После машинных кодов и перфокарт появились текстовые редакторы и язык ассемблера, что позволило писать чуть более сложные и объёмные программы при меньших трудозатратах. Далее появились процедурные языки высокого уровня – C, Паскаль и многие другие, что перевело технологии программирования на новый уровень.

Следующим крупным скачком на пути борьбы со сложностью стало появление объектно-ориентированного подхода и разработка соответствующих языков и компиляторов. Так C эволюционировал в C++. Но прогресс и сопутствующий рост сложности информационных систем не останавливаются на месте. C++ тянет за собой тяжёлым багажом обратную совместимость со своим предшественником и, как бы не усовершенствовался язык с выходом новых стандартов, все равно у программиста остаётся в распоряжении весь спектр способов «выстрелить себе в ногу», перешедший их языка C. По прежнему необходимо в большинстве случаем контролировать распределение памяти в программе, следить за валидностью указателей и т.д.

На стыке тысячелетия компания Microsoft выпустила на рынок новую платформу для разработки приложений, .NET, которая стала базой для нескольких языков программирования, поддерживаемых этой компанией в рамках среды разработки Visual Studio. Одновременно с этим в Visual Studio появился и новый язык программирования, C#, который также базировался на новой технологии .NET.

C# был призван на замену языку C++, что подразумевается самим его названием – по сути, имеется в виду, что это язык (C++)++, четыре «плюса» которого как бы и составляют значок «#». C# избавляет программиста от многих неудобств, которые годами досаждают программистам на С++, но не могут быть устранены из-за наследия его предшественника. Например, очень удобно наличие автоматического сборщика мусора, позволяющего больше не заботиться о «ручном» распределении памяти, а сосредоточиться на логике функционирования программы.

Время, прошедшее с начала тысячелетия, показало, что язык C++, при всех его кажущихся недостатках, пока рано списывать со счетов – всё-таки за удобства нового языка приходится расплачиваться некоторой потерей производительности, что неприемлемо для ряда задач. Поэтому C# стал не заменой языку C++, а скорее равноправным конкурентом на рынке большого спектра промышленных приложений.

То же самое можно сказать и о другом близком «родственнике» и по совместительству, конкуренте – популярном языке программирования Java. Как и Java, C# при своём зарождении рассматривался как инструмент разработки веб-разработки, и примерно три четверти его синтаксических конструкций позаимствованы у ближайшего конкурента.

Данная работа предназначена для демонстрации выразительной мощи и лаконичности языка C#. Целью работы является получение практических навыков по проектированию и реализации программной системы на примере простейшего стекового калькулятора. Пользовательский интерфейс выполнен в виде текстовой консоли.

# Язык программирования C#

Стандартное расширение для файлов с исходным кодом на языке C# – cs (например, main.cs). Программа на языке C# может состоять из одного или нескольких файлов с исходным кодом. Файлы с исходным кодом превращаются в программы с помощью компилятора.

Компилировать файлы можно из командной строки и из среды разработки Visual Studio. Из командной строки нужно явным образом запустить компилятор (исполняемый файл csc.exe) на выполнение, передав ему имя файла с тестом (исходным кодом) программы: csc main.cs. В результате компиляции исходного файла main.cs по умолчанию – если не указывать иного – получается исполняемый файл main.exe, который уже можно запускать на выполнение, как и любую программу Windows.

Язык C# является объектно-ориентированным и обеспечивает типовую безопасность данных. Как уже упоминалось ранее, выполнение программ, написанных на языке C#, базируется на универсальной платформе .NET от компании Microsoft. Единая среда выполнения программ основана на использовании промежуточного языка IL (Intermediate Language – промежуточный язык), исполняющего почти ту же роль, что и байт-код виртуальной машины языка Java. Используемые в рамках технологии .NET компиляторы с различных языков транслируют программы в IL-код. Так же как и байт-код Java, IL-код представляет собой команды гипотетической стековой вычислительной машины. Но есть и разница в устройстве и использовании IL.

Во-первых, в отличие от JVM (Java Virtual Mashine – виртуальная маштна Java), IL не привязан к одному языку программирования. В составе, предварительных версий Microsoft.NET имеются компиляторы с языков C++, C#, Visual Basic. Независимые разработчики могут добавлять другие языки, создавая компиляторы с этих языков в IL-код.

Во-вторых, IL предназначен не для программной интерпретации, а для последующей компиляции в машинный код. Это позволяет достичь существенно большего быстродействия программ. Содержащие IL-код файлы несут достаточно информации для работы оптимизирующего компилятора.

# Среда разработки Visual Studio

Microsoft Visual Studio Community Edition – [свободно](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) распространяемая [интегрированная среда разработки (англ. IDE, Integrated Development Environment) приложений](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) для семейства языков программирования, в том числе [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))/[C++](https://ru.wikipedia.org/wiki/C%2B%2B). Пакет установки доступен по адресу: https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=16.

Среда разработки включает редактор кода с поддержкой технологии IntelliSense – механизма автоматического дополнения кода, в зависимости от структуры программы и текущего контекста. Также имеются встроенные редактор форм и веб-редактор, помогающие разработчику быстро создавать графический пользовательский интерфейс.

Microsoft Visual Studio предоставляет множество других удобных функций для редактирования исходного кода программы – подсветку ключевых слов, идентификаторов и комментариев, а также навигацию по структуре программы и поиск по файлам проекта. Кроме того, имеется очень тесная связь с поддерживаемыми компиляторами – в нашем случае, Microsoft C++ Compiler – можно очень детально настраивать опции компиляции, используя при этом удобный диалоговый интерфейс. Кроме C++, поддерживаются также языки Visual Basic, Visual C# и Visual F#. Комплект поставки, как правило, включает также в том или ином объёме СУБД MS SQL Server.

Следует также упомянуть об удобных средствах интерактивной отладки, встроенной в редактор кода и включающей множество полезных функций – например, наглядную навигацию по ходу выполнения программы, просмотр значений переменных или содержимого стека выполнения. Встроенный в интегрированную среду разработки отладчик может работать на уровне как исходного, так и машинного кода.

Кроме встроенных инструментов, поставляемых с установочным комплектом, Visual Studio позволяет создавать свои или подключать сторонние расширения, дополняющие общую функциональность интегрированной среды – например, модули взаимодействия с различными системами контроля версий.

# Исходные данные и способ их ввода

Исходными данными в программе является последовательность команд, которые можно поделить на три категории – манипуляция со стеком (добавить/удалить операнд), информационная команда вывода на печать вершины стека и команды арифметических операций.

## Ввод данных с клавиатуры.

По умолчанию, если запустить программу без параметров, она будет принимать список команд в качестве пользовательского ввода с клавиатуры. Каждая введённая строка интерпретируется как очередная команда. Для того, чтобы ввести команду в программу, необходимо набрать её на клавиатуре и нажать Enter. После этого она будет обработана, а результат, при наличии такового, будет выведен на экран.

Завершить сеанс работы можно, либо введя команду END, либо нажав стандартное сочетание клавиш Ctrl + C.

## Ввод данных из файла.

Исходными данными в этом режиме является текстовый файл, содержащий команды и, возможно, комментарии к ним. Для активации этого режима, программу необходимо запустить из командной строки с единственным аргументом – именем файла. После запуска и инициализации переменных, программа считывает содержимое файла построчно, интерпретируя каждую непустую строку, не являющуюся комментарием, как отдельную команду – так же, как если бы она была введена с клавиатуры.

# Описание реализованных классов

Исходный код приложения состоит из двух классов – стандартного класса **Program** и класса **Calc**, представляющего собой абстракцию стекового калькулятора.

Класс **Program** – главный класс программы, «скелет» которого генерируется при создании пустого проекта. Этот класс содержит точку входа приложения – статическую функцию **Main**, с которой всегда начинается выполнение программы. Исходный код этого класс содержится одноимённом файле Program.cs.

Функция **Main** занимается анализом аргументов командной строки и, в зависимости от его результатов, организует получение исходных данных либо из файла – если первым аргументом передано его имя, либо непосредственно от пользователя, с клавиатуры – если программа запущена без аргументов.

Полученные так или иначе исходные данные интерпретируются в качестве последовательности строк, каждая из которых представляет собой команду стекового калькулятора, и передаются для обработки методам класса Calc. Этим занимается функция **processCommand**, которая предварительно анализирует тип команды, классифицирует её и подготавливает аргументы, если таковые имеются.

Класс **Calc** управляет двумя внутренними коллекциями – стеком, хранящим полученные от пользователя операнды, и словарём, в котором содержатся определённые пользователем именованные параметры, значения которых используются в арифметических операциях вместо имени. Кроме того, этот класс отвечает за непосредственное выполнение поступивших от пользователя арифметических операций.

Класс содержит функции, названия которых соответствуют командам калькулятора:

**Define** – команда «DEFINE», определить именованного параметра;

**Push** – команда «PUSH», положить значение на вершину стека;

**Pop** – команда «POP», снять значение с вершины стека;

**Print** – команда «PRINT», напечатать значение на вершине стека;

**Add** – команда «+», сложение двух операндов с вершины стека;

**Sub** – команда «-», вычитание двух операндов с вершины стека;

**Mul**, – команда «\*», умножение двух операндов с вершины стека;

**Div** – команда «/», деление двух операндов с вершины стека;

**Sqrt** – команда «SQRT», извлечение квадратного корня из вершины стека.

Кроме того, имеется вспомогательная функция **GetValue**, которая на основе переданного ей строкового литерала определяет, является ли он числом или именем параметра. Так или иначе, она возвращает соответствующее этому литералу действительное числовое значение – либо строка конвертируется в число, либо число извлекается из словаря по имени параметра, содержащемуся в этой строке.

Содержимое файлов с исходным кодом программы приведено в «Приложении А».

# Тестирование приложения

В ходе тестирования приложения рассмотрено большое количество различных наборов входных данных, включая содержащие преднамеренную ошибку.

Для удобства тестирования создан набор командных файлов, содержащих списки команд и поясняющие комментарии. Большинство тестовых случаев покрыты именно при помощи файлов и только один, последний, тест убеждается, что программа одинаково работает с исходными данными с клавиатуры точно так же, как и при получении их из файла. Содержимое тестовых командных файлов приведено в Приложении Б.

Входные данные для каждого теста использовались разнородные – положительные, отрицательные, целые и действительные, потому что при работе с арифметическими операциями нередки ошибки потери точности или случайного округления значений при использовании неправильного типа данных. Кроме того, на вход калькулятора подавались как непосредственно сами числовые значения, так и предварительно определённые именованные параметры.

Первый тест, команды которого хранятся в файле test1.cmd, проверяет работоспособность команд управления содержимым стека и печати. При этом предпринимаются попытки распечатать пустой стек и удалить из пустого стека значение. Результаты выполнения первого теста приведены на рисунке 1.

Для простоты проверки, комментарии в каждом командном файле содержат для каждой команды вывода (PRINT) ожидаемое значение. Кроме того, для удобства отслеживания содержимого стека в каждый конкретный момент, тщательно закомментированы все операции.

Далее идёт набор тестов, проверяющих работоспособность каждой арифметической операции на различных наборах входных данных.

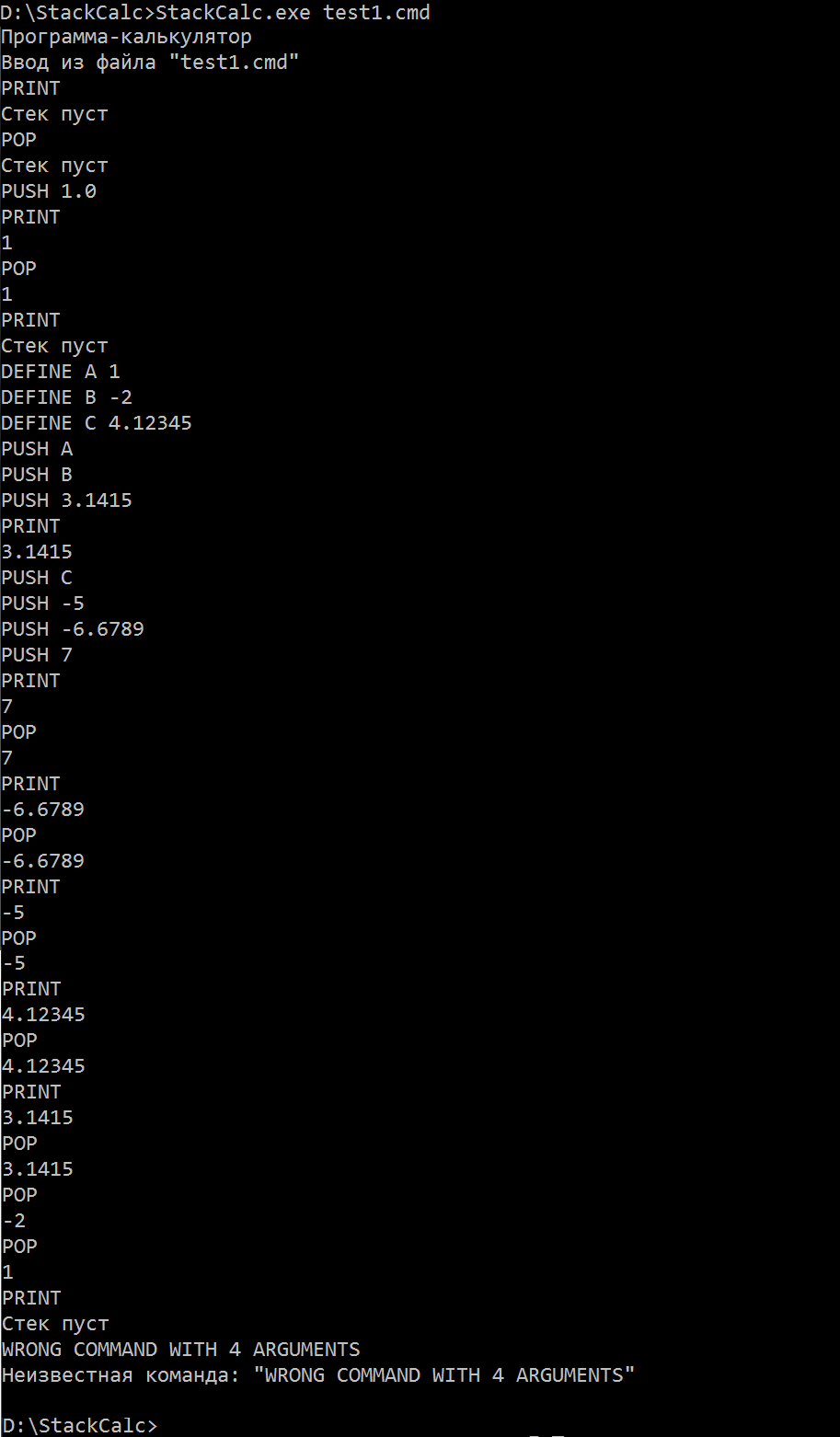


Рисунок 1. Тестирование инструментов работы со стеком

Второй тест, команды которого хранятся в файле test2.cmd, проверяет работоспособность операции сложения, результаты приведены на рисунке 2.

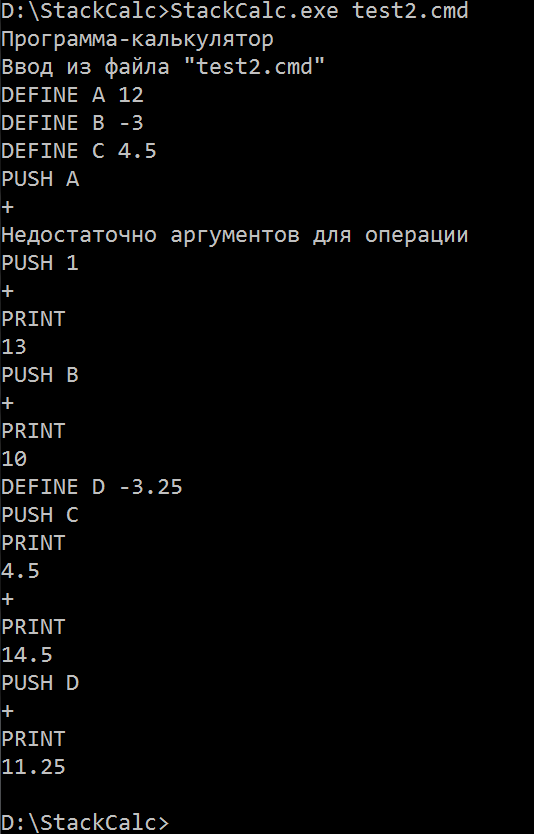


Рисунок 2. Тестирование операции сложения

Третий тест, команды которого хранятся в файле test3.cmd, проверяет работоспособность операции вычитания. Результаты выполнения этого теста приведены на рисунке 3.

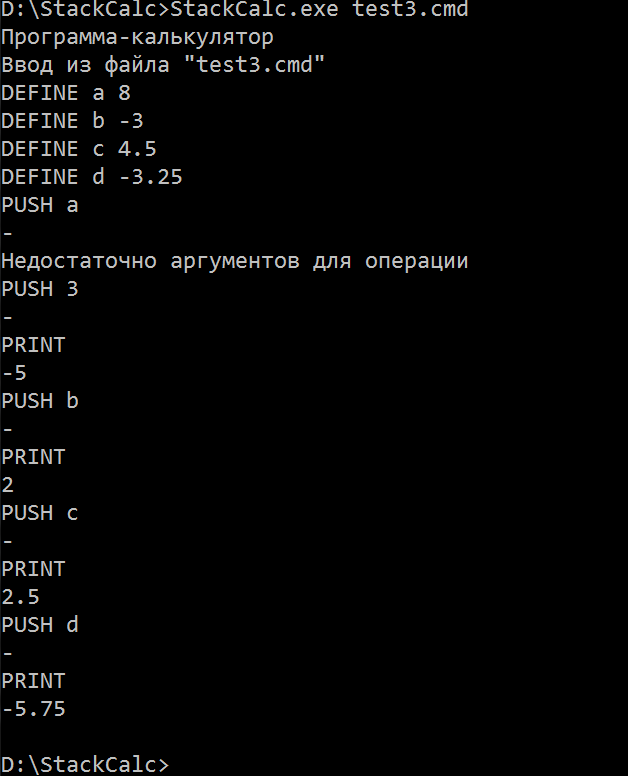


Рисунок 3. Тестирование операции вычитания

Четвёртый тест, команды которого хранятся в файле test4.cmd, проверяет работоспособность операции умножения. Результаты приведены на рисунке 4.

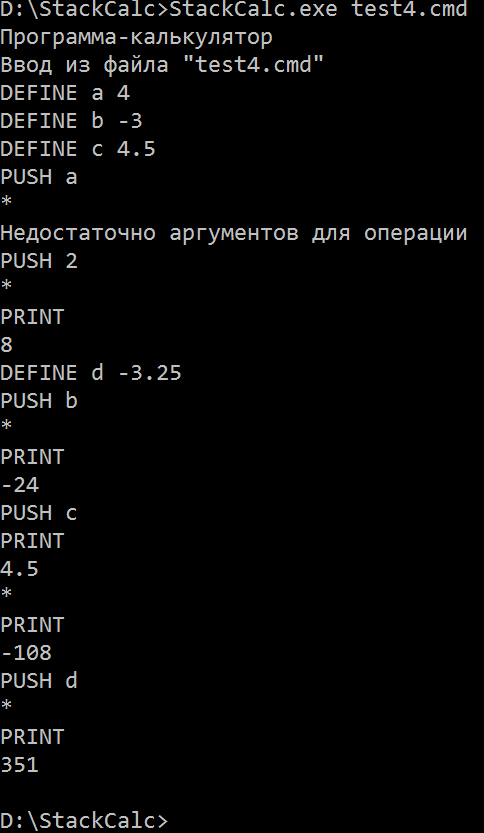


Рисунок 4. Тестирование операции умножения

Пятый тест, команды которого хранятся в файле test5.cmd, проверяет работоспособность операции деления. Результаты выполнения пятого теста приведены на рисунке 5.

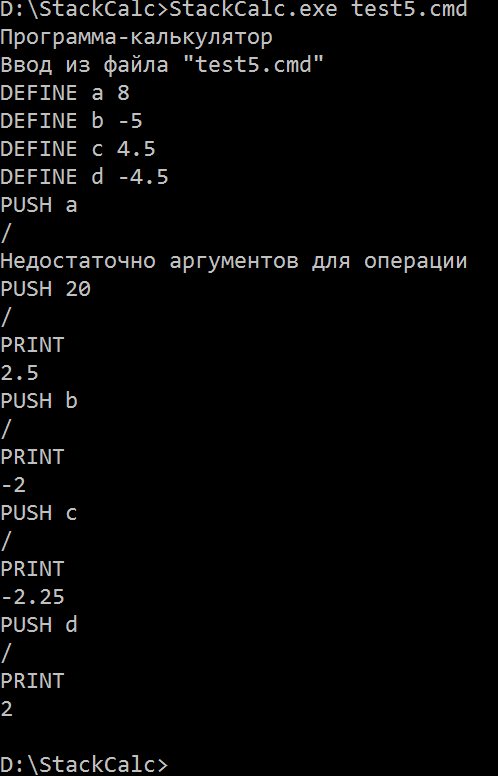


Рисунок 5. Тестирование операции деления

Шестой тест, команды которого хранятся в файле test6.cmd, проверяет работоспособность операции извлечения квадратного корня. Результаты выполнения этого теста приведены на рисунке 6.

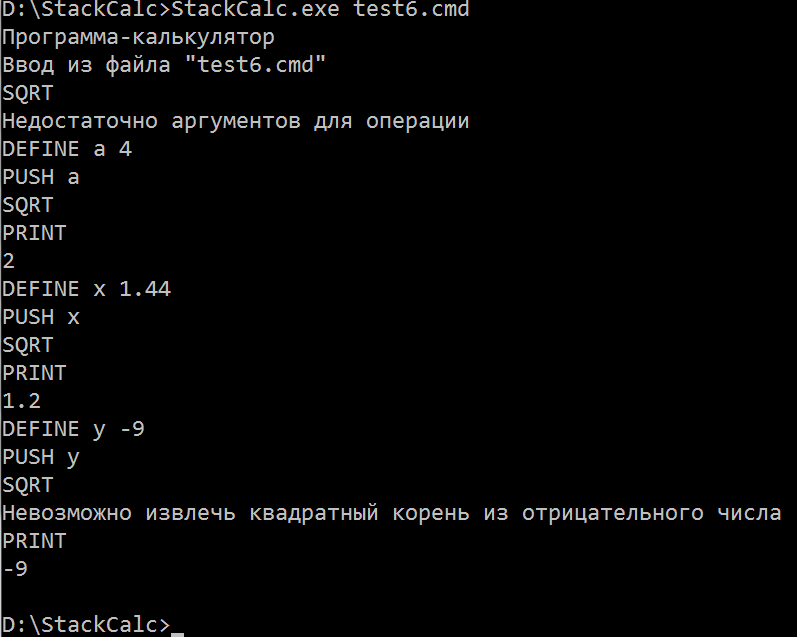


Рисунок 6. Тестирование операции извлечения корня

Как видно из рисунка, при попытке выполнения невозможной операции выводится сообщение об ошибке, а операнд остаётся на вершине стека.

Седьмой тест, команды которого хранятся в файле test7.cmd, проверяет работу калькулятора в комплексе – задаются, как обычно, различные параметры, вводятся разнородные значения и выполняются на этот раз все доступные операции по очереди. Результаты выполнения седьмого теста приведены на рисунке 7.

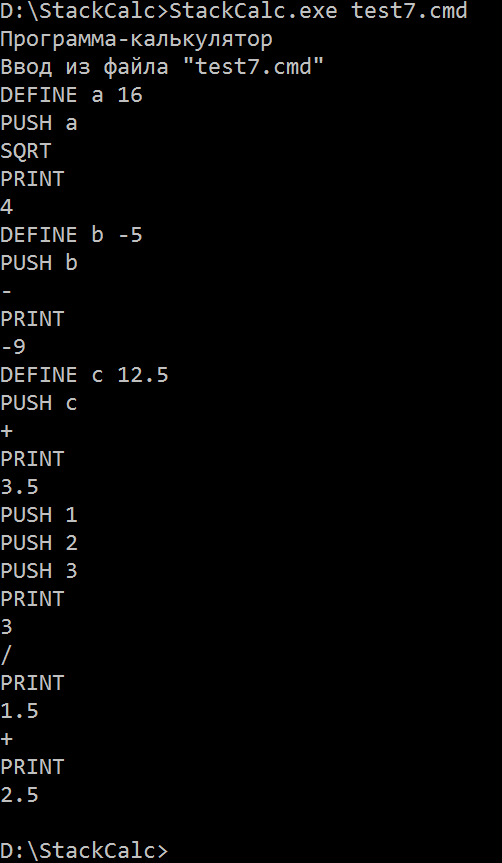


Рисунок 7. Комплексное тестирование арифметических операций

Восьмой, последний, тест, команды которого хранятся в файле test8.cmd, проверяет систему ввода данных приложения. Если до сих пор команды вводились исключительно из файлов, то теперь они вводятся в командной строке с клавиатуры. Для того, чтобы сравнить идентичность работы калькулятора в этих двух режимах, вводятся команды из предыдущего теста. Результаты выполнения теста приведены на рисунке 8.

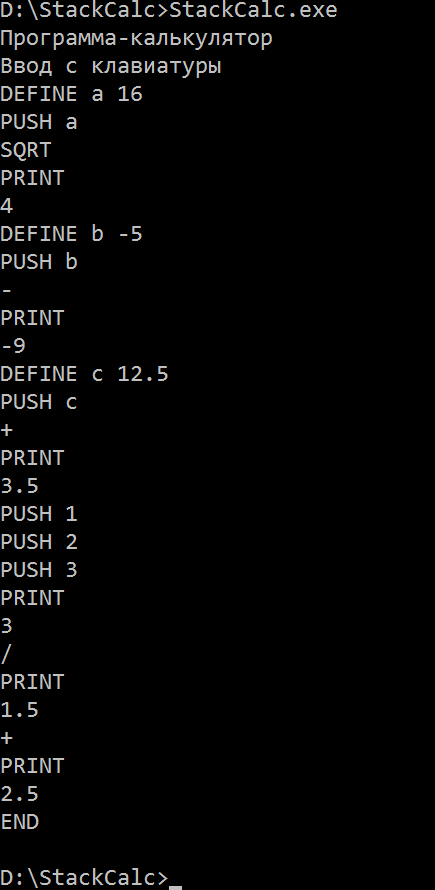


Рисунок 8. Повторение предыдущего теста с клавиатуры

# Заключение

В результате работы была спроектирована и разработана программа, позволяющая обрабатывать входные массивы строк, интерпретируя их в качестве команд стекового калькулятора и выполняя заданные ими вычисления с выводом результат на экран.

В ходе работы над проектом получены практические навыки разработки программ на языке высокого уровня C#. Изучены особенности создания приложений с помощью этого языка программирования и среды разработки Microsoft Visual Studio. Отработаны методики работы со строками и коллекциями – стеком Stack, который использовался для работы с основным операционным стеком калькулятора и словарём (ассоциативной коллекцией) Dictionary, который использовался для хранения пользовательских именованных параметров.

В целом, на базе практического применения, закреплены базовые знания, полученные ранее, при изучении теории.

# Список литературы

1. Бородакий, Ю. В. Эволюция информационных систем / Ю.В. Бородакий, Ю.Г. Лободинский. – Москва:СИНТЕГ, 2011. – 368 c.

2. Вагнер, Билл С# Эффективное программирование / Билл Вагнер. – М.: ЛОРИ, 2013. - 320 c.

3. Троелсен, Эндрю Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. - М.: Вильямс, 2015. – 486 c.

4. Абрамян, Михаил Visual C# на примерах / Михаил Абрамян. – М.: БХВ-Петербург, 2015. – 572 c.

5. Культин, Н. С# в задачах и примерах / Н. Культин. – М.: БХВ-Петербург, 2016. – 952 c.

6. Ишкова, Э. А. Самоучитель С#. Начала программирования / Э.А. Ишкова. – М.: Наука и техника, 2017. – 496 c

7. ГОСТ 2.105-95 Общие требования к текстовым документам.

# ПРИЛОЖЕНИЕ А. Исходный код программы

**Program.cs**

using System;

using System.IO;

namespace StackCalc

{

// Основной класс приложения

class Program

{

// Ссылка на класс калькулятора

static Calc calc = new Calc();

// Основная функция (точка входа) приложения

static void Main(string[] args)

{

Console.WriteLine("Программа-калькулятор");

try

{

// Проверяем наличие аргументов командной строки

if (args.Length > 0)

{

// Первый аргумент считаем именем файла,

// считываем команды из него

string fileName = args[0];

Console.WriteLine("Ввод из файла \"" + fileName + "\"");

// Открываем файл на чтение

StreamReader reader = new StreamReader(fileName);

// Читаем строки до конца файла

while (true)

{

string cmd = reader.ReadLine();

if (cmd != null)

{

// Каждую прочитанную строку печатаем

// и отправляем в калькулятор

if (processCommand(cmd, true))

{

break;

}

}

else

{

break;

}

}

// Закрываем файл

reader.Close();

}

else

{

// Программа запущена без параментров,

// считываем команды из консоли

Console.WriteLine("Ввод с клавиатуры");

while (true)

{

// Каждую прочитанную строку передаём

// калькулятору без вывода на экран

if (processCommand(Console.ReadLine(), false))

{

break;

}

}

}

}

catch (System.Exception exception)

{

Console.WriteLine("Ошибка: "+ exception.Message);

}

}

// Функция обработки отдельной команды

// Возвращаемое значение:

// false - продолжаем ввод команд

// true - завершаем работу

static bool processCommand(string cmd, bool trace)

{

// Комментарии и пустые строки пропускаем

if (cmd.Length == 0 || cmd[0] == '#')

{

return false;

}

// Выводим исходную команду на консоль

// (при вводе с клавиатуры не имеет смысла,

// т.к. пользователь уже её сам набрал в консоли)

if (trace)

{

Console.WriteLine(cmd);

}

// Переводим все символы в верхний регистр

// (убираем зависимость от регистра)

cmd = cmd.ToUpper();

// Разбиваем команду на имя команды

// и аргументы (если есть)

string[] tokens = cmd.Split(new char[] { ' ' });

// Первый элемент обязан содержать имя команды

if (tokens.Length == 0)

{

return false;

}

if (tokens[0] == "DEFINE")

{

// Задать значение параметра

// (ожидаем: DEFINE имя значение)

if (tokens.Length == 3)

{

try

{

calc.Define(tokens[1], tokens[2]);

}

catch (System.FormatException exeption)

{

Console.WriteLine("Formatxception: " +

exeption.Message);

}

}

}

else if (tokens[0] == "PUSH")

{

// Положить число на стек

// (ожидаем: PUSH значение)

if (tokens.Length == 2)

{

calc.Push(tokens[1]);

}

}

else if (cmd == "POP")

{

// Снять число со стека

calc.Pop();

}

else if (cmd == "PRINT")

{

// Печатаем верхушку стека

calc.Print();

}

else if (cmd == "+")

{

// Складываем два верхних операнда

calc.Add();

}

else if (cmd == "-")

{

// Вычитаем два верхних операнда

calc.Sub();

}

else if (cmd == "\*")

{

// Перемножаем два верхних операнда

calc.Mul();

}

else if (cmd == "/")

{

// Делим два верхних операнда

calc.Div();

}

else if (cmd == "SQRT")

{

// Извлекаем корень из операнда на верхушине стека

calc.Sqrt();

}

else if (cmd == "END")

{

// Возвращаем признак завершения работы

return true;

}

else

{

// Сообщаем об ошибке, но не прерываем работу

Console.WriteLine("Неизвестная команда: \"" + cmd + "\"");

}

// Возвращаем признак продолжения работы

return false;

}

}

}

**Calc.cs**

using System;

using System.Collections.Generic;

using System.Globalization;

namespace StackCalc

{

/\*

\* Класс, отвечающий за основные функции программы -

\* управление стеком, хранение параментров и выполнение операций

\*/

class Calc

{

Stack<string> stack = new Stack<string>();

Dictionary<string, double> defs = new Dictionary<string, double>();

// Задать значение сохраняемого параметра

// параметры функции:

// name - имя сохраняемого параметра

// str - строковое значение сохраняемого параметра

public void Define(string name, string str)

{

// Преобразовать строковое значение

// в число с плавающей точкой

double value = double.Parse(str.Trim(),

CultureInfo.InvariantCulture.NumberFormat);

if (defs.ContainsKey(name))

{

// Заменить существующее значение

defs[name] = value;

}

else

{

// Добавить новое определение

defs.Add(name, value);

}

}

// Поместить операнд на вершину стека

// параметры функции:

// arg - строковое представление операнда

public void Push(string arg)

{

stack.Push(arg);

}

// Снять операнд с вершины стека и напечатать

public void Pop()

{

if (stack.Count > 0)

{

Console.WriteLine(GetValue(stack.Pop()).ToString());

}

else

{

Console.WriteLine("Стек пуст");

}

}

// Напечатать значение операнда на вершине стека

// если на вершине стека находится имя сохраённого

// параметра, то его значение извлекается из хранилища

public void Print()

{

if (stack.Count > 0)

{

Console.WriteLine(GetValue(stack.Peek()).ToString());

}

else

{

Console.WriteLine("Стек пуст");

}

}

// Складывает два верхних операнда,

// результат помещает обратно в стек

public void Add()

{

if (stack.Count > 1)

{

// Забрать из стека первый операнд

double a = GetValue(stack.Pop());

// Забрать из стека второй операнд

double b = GetValue(stack.Pop());

// Выполнить операцию (сложение)

double c = a + b;

// Поместить результат обратно в стек

stack.Push(c.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Вычитает два верхних операнда,

// результат помещает обратно в стек

public void Sub()

{

if (stack.Count > 1)

{

// Забрать из стека первый операнд

double a = GetValue(stack.Pop());

// Забрать из стека второй операнд

double b = GetValue(stack.Pop());

// Выполнить операцию (вычитание)

double c = a - b;

// Поместить результат обратно в стек

stack.Push(c.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Перемножает два верхних операнда,

// результат помещает обратно в стек

public void Mul()

{

if (stack.Count > 1)

{

// Забрать из стека первый операнд

double a = GetValue(stack.Pop());

// Забрать из стека второй операнд

double b = GetValue(stack.Pop());

// Выполнить операцию (умножение)

double c = a \* b;

// Поместить результат обратно в стек

stack.Push(c.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Делит два верхних операнда,

// результат помещает обратно в стек

public void Div()

{

if (stack.Count > 1)

{

// Забрать из стека первый операнд

double a = GetValue(stack.Pop());

// Забрать из стека второй операнд

double b = GetValue(stack.Pop());

// Выполнить операцию (деление)

double c = a / b;

// Поместить результат обратно в стек

stack.Push(c.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Извлекает корень из операнда на верхушине стека,

// результат помещает обратно в стек

public void Sqrt()

{

if (stack.Count > 0)

{

// Забрать из стека аргумент

double x = GetValue(stack.Pop());

if(x >= 0)

{

// Выполнить операцию (взятие квадратного корня)

x = Math.Sqrt(x);

}

else

{

Console.WriteLine("Невозможно извлечь квадратный корень из отрицательного числа");

}

// Поместить результат обратно в стек

stack.Push(x.ToString());

}

else

{

Console.WriteLine("Недостаточно аргументов для операции");

}

}

// Возвращает действительное число, соответствующее

// переданному строковому аргументу.

// Если это число, то функция преобразует строковое

// представление в число с плавающей точкой, иначе

// извлекает число из хранилища именованных параметров,

// параметры функции:

// arg - строковое представление операнда

private double GetValue(string arg)

{

double value = 0.0;

try

{

// Пробуем преобразовать строковый аргумент в число с плавающей точкой

value = double.Parse(arg.Trim(),

CultureInfo.InvariantCulture.NumberFormat);

}

catch (System.FormatException)

{

// Ошибка - возможно, это имя параметра,

// пробуем найти его в коллекции определений

if (defs.ContainsKey(arg))

{

// Заменить существующее значение

value = defs[arg];

}

else

{

// Ничего не получилось - выдать ошибку

throw new System.Exception("Некорректный аргумент: " + arg);

}

}

return value;

}

}

}

# ПРИЛОЖЕНИЕ B. Содержимое тестовых командных файлов

**test1.cmd**

# Тест для проверки работоспособности базового инструментария:

# добавление/удаление параметров и операндов (целочисленных и дробных, положительных и отрицательных)

# печать вершины стека и проверка на отказустойчивость при работе с пустым стеком

# проверка того, что коллекция операндов работает именно как стек

# печать пустого стека ( сообщение об ошибке)

PRINT

# попытка удаления из пустого стека ( сообщение об ошибке)

POP

# добавление в стек положительного целого числа

PUSH 1.0

# печать вершины стека (ожидается: "1")

PRINT

# удаление из стека элемента "1"

POP

# печать вершины стека (ожидается сообщение об ошибке)

PRINT

# определение положительного целого параметра

DEFINE a 1

# определение отрицательного целого параметра

DEFINE b -2

# определение положительного дробного параметра

DEFINE c 4.12345

# добавление в стек положительного целого числа

PUSH a

# добавление в стек отрицательного целого числа

PUSH b

# добавление в стек положительного целого числа

PUSH 3.1415

# печать вершины стека (ожидается: "3.1415")

PRINT

# добавление в стек положительного дробного числа

PUSH c

# добавление в стек отрицательного целого числа

PUSH -5

# добавление в стек отрицательного дробного числа

PUSH -6.6789

# добавление в стек положительного целого числа

PUSH 7

# печать вершины стека (: "7")

PRINT

# удаление из стека элемента "7"

POP

# печать вершины стека (: "-6.6789")

PRINT

# удаление из стека элемента "-6.6789"

POP

# печать вершины стека (: "-5")

PRINT

# удаление из стека элемента "-5"

POP

# печать вершины стека (: "4.12345")

PRINT

# удаление из стека элемента "4.12345"

POP

# печать вершины стека (: "3")

PRINT

# удаление из стека элемента "3"

POP

# удаление из стека элемента "-2"

POP

# удаление из стека элемента "1"

POP

# печать вершины стека ( сообщение об ошибке)

PRINT

# неправильная команда ( сообщение об ошибке)

WRONG COMMAND WITH 4 ARGUMENTS

**test2.cmd**

# Тест для проверки операции сложения

# определение положительного целого параметра

DEFINE a 12

# определение отрицательного целого параметра

DEFINE b -3

# определение положительного дробного параметра

DEFINE c 4.5

# добавление в стек положительного целого числа

PUSH a

# попытка выполнения операции при недостаточном количестве операндов (ожидается сообщение об ошибке)

+

# добавление в стек положительного целого числа

PUSH 1

# выполнение операции: 12 + 1 = 13

+

# печать вершины стека (ожидается: "13")

PRINT

# добавление в стек отрицательного целого числа

PUSH b

# выполнение операции: 13 + (-3) = 10

+

# печать вершины стека (ожидается: "10")

PRINT

# определение отрицательного дробного параметра

DEFINE d -3.25

# добавление в стек положительного дробного числа

PUSH c

# печать вершины стека (ожидается: "4.5")

PRINT

# выполнение операции: 10 + 4.5 = 14.5

+

# печать вершины стека (ожидается: "14.5")

PRINT

# добавление в стек отрицательного дробного числа

PUSH d

# выполнение операции: 14.5 + (-3.25) = 11.25

+

# печать вершины стека (ожидается: "11.25")

PRINT

**test3.cmd**

# Тест для проверки операции вычитания

# определение положительного целого параметра

DEFINE a 8

# определение отрицательного целого параметра

DEFINE b -3

# определение положительного дробного параметра

DEFINE c 4.5

# определение отрицательного дробного параметра

DEFINE d -3.25

# добавление в стек положительного целого числа

PUSH a

# попытка выполнения операции при недостаточном количестве операндов (ожидается сообщение об ошибке)

-

# добавление в стек положительного целого числа

PUSH 3

# выполнение операции: 3 - 8 = -5

-

# печать вершины стека (ожидается: "-5")

PRINT

# добавление в стек отрицательного целого числа

PUSH b

# выполнение операции: -3 - (-5) = 2

-

# печать вершины стека (ожидается: "2")

PRINT

# добавление в стек положительного дробного числа

PUSH c

# выполнение операции: 4.5 - 2 = 2.5

-

# печать вершины стека (ожидается: "2.5")

PRINT

# добавление в стек отрицательного дробного числа

PUSH d

# выполнение операции: -3.25 - 2.5 = -5.75

-

# печать вершины стека (ожидается: "-5.75")

PRINT

**test4.cmd**

# Тест для проверки операции умножения

# определение положительного целого параметра

DEFINE a 4

# определение отрицательного целого параметра

DEFINE b -3

# определение положительного дробного параметра

DEFINE c 4.5

# добавление в стек положительного целого числа

PUSH a

# попытка выполнения операции при недостаточном количестве операндов (ожидается сообщение об ошибке)

\*

# добавление в стек положительного целого числа

PUSH 2

# выполнение операции: 2 \* 4 = 8

\*

# печать вершины стека (ожидается: "8")

PRINT

# определение отрицательного дробного параметра

DEFINE d -3.25

# добавление в стек отрицательного целого числа

PUSH b

# выполнение операции: -3 \* 8 = -24

\*

# печать вершины стека (ожидается: "-24")

PRINT

# добавление в стек положительного дробного числа

PUSH c

# печать вершины стека (ожидается: "4.5")

PRINT

# выполнение операции: 4.5 \* (-24) = -108

\*

# печать вершины стека (ожидается: "-108")

PRINT

# добавление в стек отрицательного дробного числа

PUSH d

# выполнение операции: -3.25 + (-108) = 351

\*

# печать вершины стека (ожидается: "351")

PRINT

**test5.cmd**

# Тест для проверки операции деления

# определение положительного целого параметра

DEFINE a 8

# определение отрицательного целого параметра

DEFINE b -5

# определение положительного дробного параметра

DEFINE c 4.5

# определение отрицательного дробного параметра

DEFINE d -4.5

# добавление в стек положительного целого числа

PUSH a

# попытка выполнения операции при недостаточном количестве операндов (ожидается сообщение об ошибке)

/

# добавление в стек положительного целого числа

PUSH 20

# выполнение операции: 20 / 8 = 2.5 (убедимся, что нигде не отбрасывается дробная часть)

/

# печать вершины стека (ожидается: "2.5")

PRINT

# добавление в стек отрицательного целого числа

PUSH b

# выполнение операции: -5 / 2.5 = -2

/

# печать вершины стека (ожидается: "-2")

PRINT

# добавление в стек положительного дробного числа

PUSH c

# выполнение операции: 4.5 / -2 = -2.25

/

# печать вершины стека (ожидается: "-2.25")

PRINT

# добавление в стек отрицательного дробного числа

PUSH d

# выполнение операции: -4.5 / -2.25 = 2

/

# печать вершины стека (ожидается: "2")

PRINT

**test6.cmd**

# Тест для проверки операции деления

# попытка выполнения операции при недостаточном количестве операндов (ожидается сообщение об ошибке)

SQRT

# определение положительного целого параметра

DEFINE a 4

# добавление в стек положительного целого числа

PUSH a

# выполнение операции: SQRT(4) = 2

SQRT

# печать вершины стека (ожидается: "2")

PRINT

# определение положительного дробного параметра

DEFINE x 1.44

# добавление в стек положительного дробного числа

PUSH x

# выполнение операции: SQRT(1.44) = 1.2

SQRT

# печать вершины стека (ожидается: "1.2")

PRINT

# определение отрицательного параметра

DEFINE y -9

# добавление в стек отрицательного целого числа

PUSH y

# попытка извлечения корня из отрицательного числа (ожидается сообщение об ошибке)

SQRT

# печать вершины стека (ожидается: "-9", поскольку последняя операция не выполнена)

PRINT

**test7.cmd**

# Тест для комплексно проверки работы программы - совместное выполнение операций

# определение положительного целого параметра

DEFINE a 16

# добавление в стек положительного целого числа

PUSH a

# выполнение операции: SQRT(16) = 4

SQRT

# печать вершины стека (ожидается: "4")

PRINT

# определение отрицательного целого параметра

DEFINE b -5

# добавление в стек отрицательного целого числа

PUSH b

# выполнение операции: -5 - 4 = -9

-

# печать вершины стека (ожидается: "-9")

PRINT

# определение положительного дробного параметра

DEFINE c 12.5

# добавление в стек положительного дробного числа

PUSH c

# выполнение операции: 12.5 + (-9) = 3.5

+

# печать вершины стека (ожидается: "3.5")

PRINT

# добавление в стек нескольких чисел подряд

PUSH 1

PUSH 2

PUSH 3

# печать вершины стека (ожидается: "3")

PRINT

# выполнение операции: 3 / 2 = 1.5

/

# печать вершины стека (ожидается: "1.5")

PRINT

# выполнение операции: 1.5 + 1 = 2.5 (2 и 3 должны быть уже изъяты из стека)

+

# печать вершины стека (ожидается: "2.5")

PRINT