**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](#_Toc10814120)

[Введение 4](#_Toc10814121)

[1 Исходные данные и способ их ввода 5](#_Toc10814122)

[2 Глобальные данные 6](#_Toc10814123)

[2.1 Основные модули программы 6](#_Toc10814124)

[2.2 Описание глобальных данных модуля main.c 6](#_Toc10814125)

[2.3 Описание глобальных данных модуля dict.c 7](#_Toc10814126)

[3 Описание пользовательских макроопределений 8](#_Toc10814127)

[3.1 Макроопределения модуля main 8](#_Toc10814128)

[3.2 Макроопределения модуля dict 8](#_Toc10814129)

[3.3 Макроопределения модуля menu 8](#_Toc10814130)

[3.4 Макроопределения модуля input 9](#_Toc10814131)

[3.5 Глобальные макроопределения 10](#_Toc10814132)

[4 Описание пользовательских функции 11](#_Toc10814133)

[4.1 Функции модуля main 11](#_Toc10814134)

[4.2 Функции модуля dict 13](#_Toc10814135)

[4.3 Функции модуля menu 13](#_Toc10814136)

[4.4 Функции модуля input 14](#_Toc10814137)

[4.5 Функции модуля list 15](#_Toc10814138)

[5 Нестандартные функции 17](#_Toc10814139)

[6 Внешние файлы 18](#_Toc10814140)

[7 Описание порядка работы с программой 19](#_Toc10814141)

[Заключение 27](#_Toc10814142)

[Список литературы 28](#_Toc10814143)

[ПРИЛОЖЕНИЕ А. Исходные текст программы 29](#_Toc10814144)

[ПРИЛОЖЕНИЕ B. Команды тестов 42](#_Toc10814145)

# 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 как бы подразумевается самим его названием. По сути, это языка (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 конструкции 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.

# 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 пользовательских функции

## Функции модуля main

**int Run()** – основная рабочая функция программы

**int main(int argc, char\* argv[])** – точка входа программы.

**Функции обратного вызова.**

Обработчики пунктов верхнего меню:

* **int Add(MENU\*, ITEM\*)** – пункт «Добавить»
* **int Edit(MENU\*, ITEM\*)** – пункт «Изменить»
* **int Delete(MENU\*)** – пункт «Удалить»
* **int Search(MENU\*, ITEM\*)** – пункт «Поиск»
* **int Sort(MENU\*, ITEM\*)** – пункт «Сортировка»
* **int Save(MENU\*, ITEM\*)** – пункт «Сохранить»
* **int Help(MENU\*, ITEM\*)** – пункт «Справка»
* **int Exit(MENU\*, ITEM\*)** – пункт «Выйти»

Обработчики пунктов меню подтверждения выхода:

* int **ExitYes(MENU\*, ITEM\*)** – выход с сохранением данных.
* **int ExitNo(MENU\*, ITEM\*)** – выход без сохранения данных.
* **int ExitCancel(MENU\*, ITEM\*)** – отмена выхода.

Обработчики горячих клавиш:

* **int DefaultESC(MENU\*)** – обработчик клавиши Esc по умолчанию, устанавливается для большинства меню и производит закрытие текущего меню без дополнительных действий.
* **int ESC(MENU\*)** – обработчик клавиши Esc для главного и верхнего меню, проверяет необходимость сохранения данных перед выходом.
* **int F1(MENU\*)** – обработчик горячей клавиши F1 (Справка).
* **int F2(MENU\*)** – обработчик горячей клавиши F2 (Сохранение).
* **int F3(MENU\*)** – обработчик горячей клавиши F3 (Поиск).
* **int F4(MENU\*)** – обработчик горячей клавиши F4 (Изменить).
* **int F9(MENU\*)** – обработчик горячей клавиши F9 (Переход к верхнему меню).

Функции обратного вызова для односвязного списка:

* **void curr\_menu\_changed(MENU\* menu, int direction, int wrap)** – обработчик прокрутки меню на один элемент вперёд или назад.
* **int dict\_entry\_display(void\* data, int index, void\* param)** – перенос очередного элемента словаря в элемент описания пункта меню.
* **int dict\_entry\_find(void\* data, void\* param)** – проверка элемента словаря на соответствие критерию поиска.
* **int dict\_entry\_compare(void\* data1, void\* data2, void\* arg)** – сравнение двух элементов словаря.

**Вспомогательные функции.**

**int HelpFromFile(HANDLE hStdOut, const char\* file\_name, const char\* title, SMALL\_RECT rect)** – отображение текста справки из файла на всплывающем окне.

**int ShowMenu(HANDLE hStdOut, ITEM\_DEF\* menu\_items, int item\_count, const char\* title, SMALL\_RECT rect, int flags, int user\_tag, ExecuteHotketCB f1CB)** – отображение упрощённого вертикального меню в один столбец.

**ITEM\_DEF\* MenuItemsFromFile(const char\* file\_name, int max\_count, int max\_len, int\* pcount)** – загрузка пунктов меню из файла.

**void LoadInitialData()** – загрузка тестовых данных в словарь (отладочная функция).

## Функции модуля dict

**DICT\_ENTRY\* dict\_entry\_new(const char\* word\_eng, const char\* word\_part, const char\* word\_rus)** – создание нового элемента словаря.

**int dict\_load(const char\* file\_name)** – загрузка словаря из файла.

**int dict\_entry\_save(void\* data, int index, void\* param)** – сохранение одного элемента словаря в файл.

**int dict\_save(const char\* file\_name)** – сохранение словаря в файл.

**void dict\_entry\_clear(DICT\_ENTRY\* entry)** – удаление элемента словаря.

**void dict\_clear()** – удаление всего словаря из памяти.

## Функции модуля menu

int menu\_init(MENU\* menu, MENU\* parent, HANDLE hstdout,

ITEM\_DEF\* item\_defs, int item\_count, int cell\_count,

int orient, const SMALL\_RECT\* prect, int border,

char\* headers[]) –

инициализация экземпляра меню.

## Функции модуля input

int box\_init(InputBox\* box, HANDLE handle, SMALL\_RECT rect, char\*\*\* contents, int row\_count) – инициализация экземпляра окна ввода.

void box\_clear(InputBox\* box) – очистка полей экземпляра окна ввода.

int box\_save(InputBox\* box) – сохранить прямоугольник на экране.

int box\_draw(InputBox\* box) – нарисовать окно ввода и запустить цикл обработки нажатия клавиш.

int box\_restore(InputBox\* box) – восстановить прямоугольник на экране.

## Функции модуля list

# Тестирование приложения

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

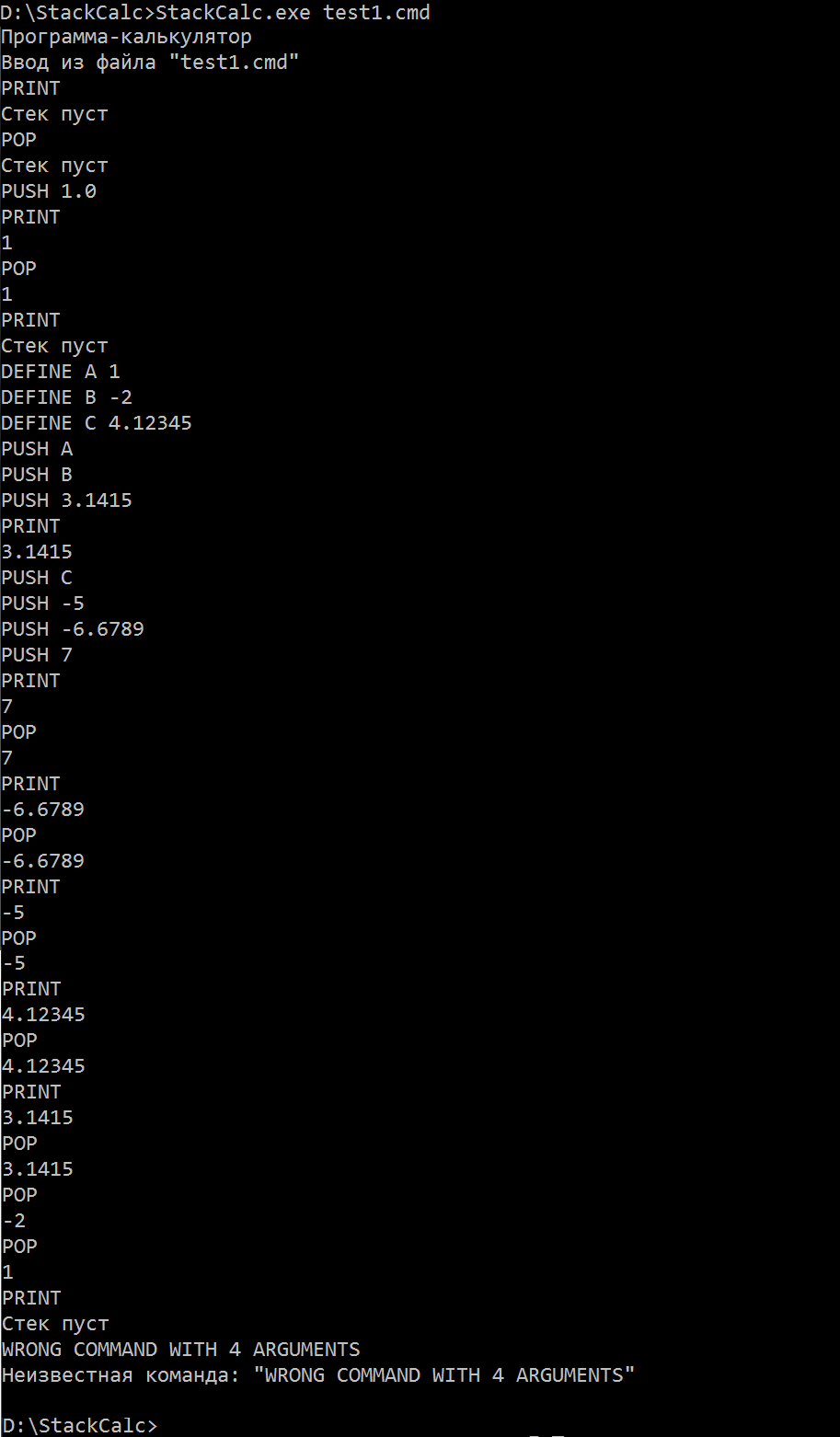


Рисунок 1. Тестирование инструментов работы со стеком

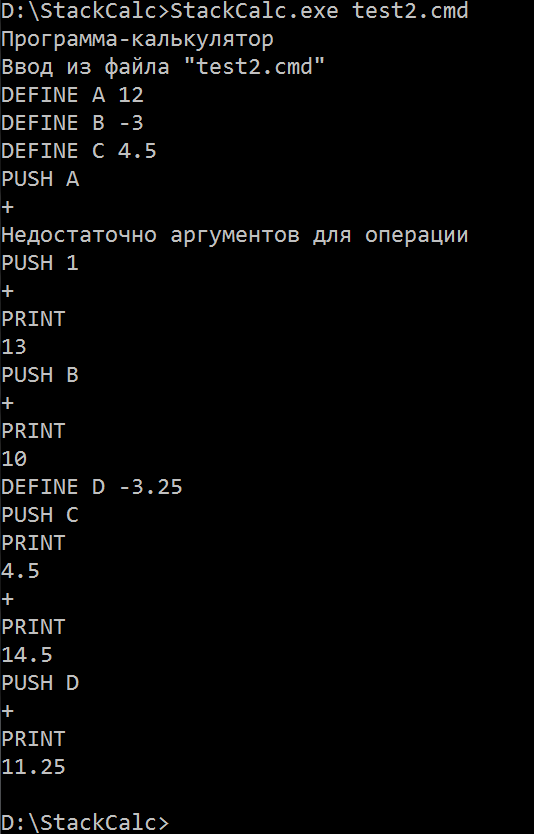


Рисунок 2. Тестирование операции сложения

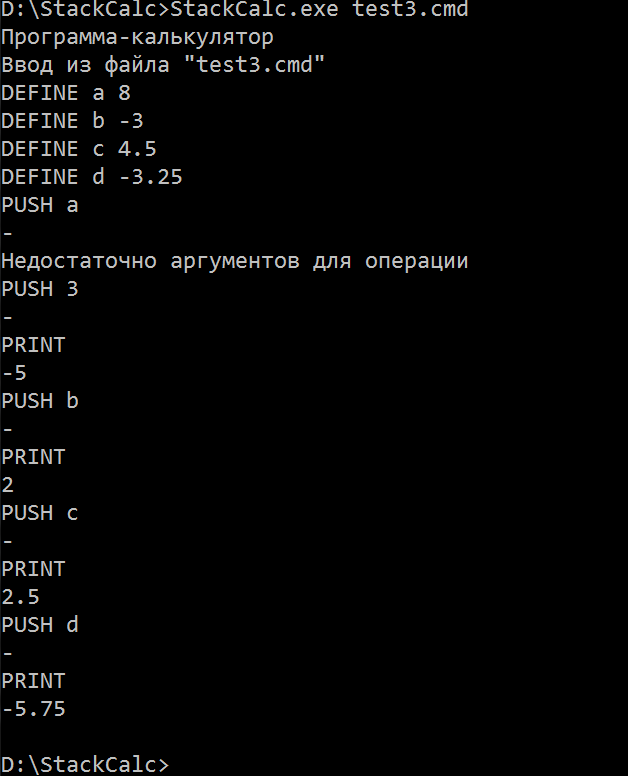


Рисунок 3. Тестирование операции вычитания

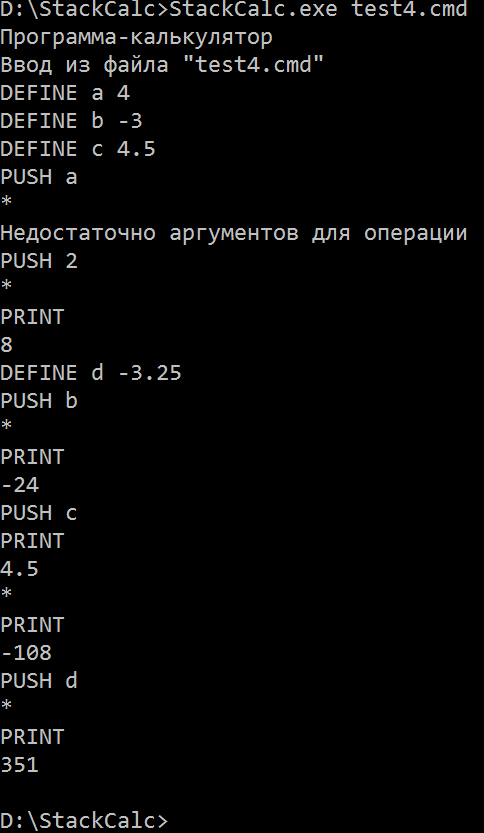


Рисунок 4. Тестирование операции умножения

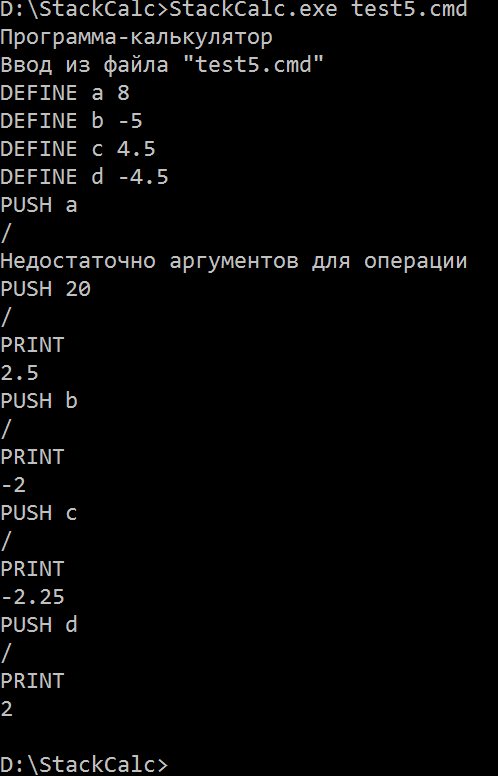


Рисунок 5. Тестирование операции деления

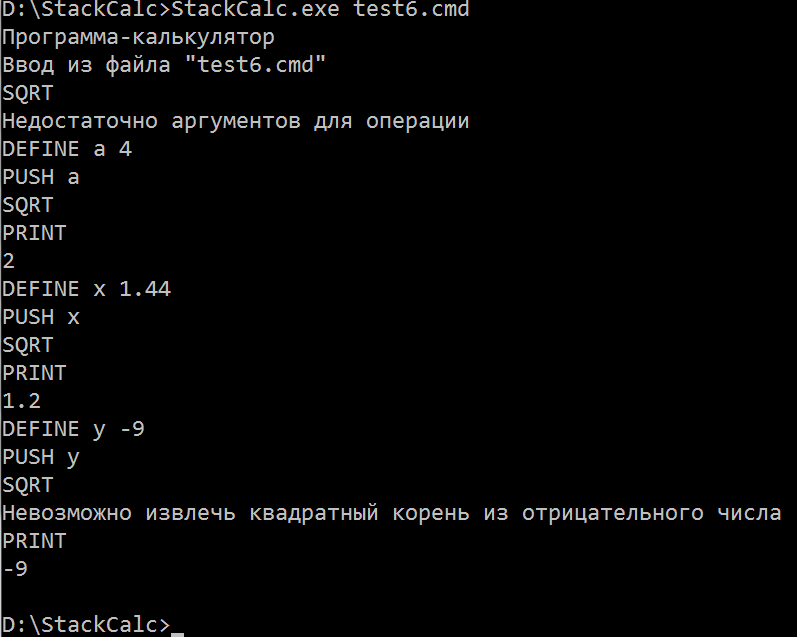


Рисунок 6. Тестирование операции извлечения корня

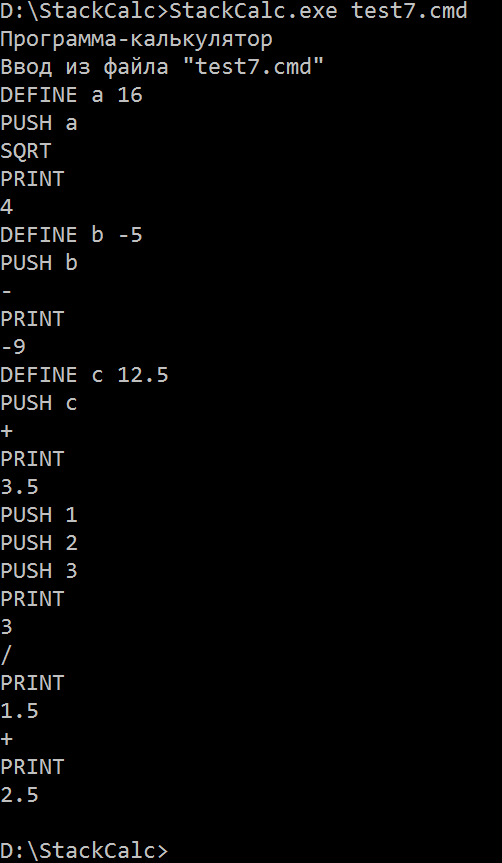


Рисунок 7. Комплексное тестирование арифметических операций

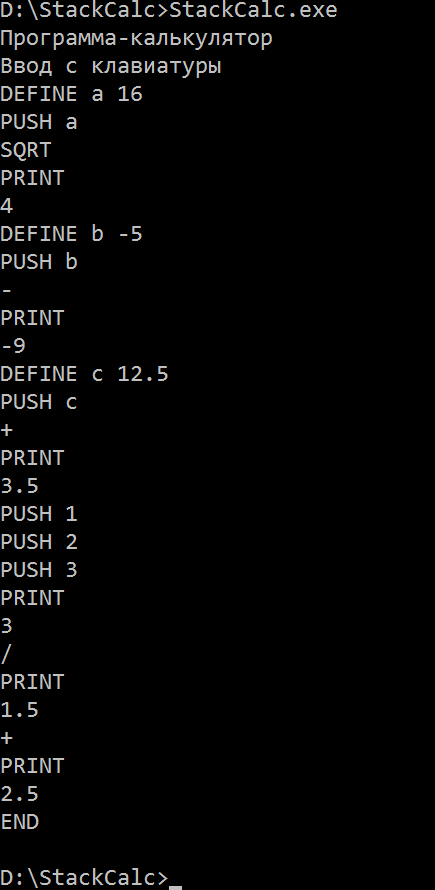


Рисунок 8. Повторение предыдущего теста с клавиатуры

# 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 (ассоциативная коллекция) 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. Исходные текст 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 функця (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

// Возвражаемое 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. Команды тестов

**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