Testing Techniques
Assignment 1

Sasja Gillissen, Martin Huijben, Martijn Terpstra

September 28, 2016

1 Introduction

In this document we will describe bla bla intro

2 Test Goal

3 The Product

The system under test is a calculator with a command line interface (cli). This system is developed in Java and has no external dependencies. This product can be executed on all platforms supported by the Java Virtual Machine (JVM). The user can interface with the program using the command line, upon executing the program, it will show a prompt in which the user can type expressions or define variables and functions. The input will be parsed, evaluated and the result will be printed. This process repeats itself until te user quits the program.

4 The Specification

the program will continuously prompt the user for input. It will continue to prompt the user for input until it receives the exit function as input and then terminates.

Depending on the input given the program will either

- Evaluate an expression
- Assign a value to a variable and store it in memory
- Assign a value to a function and store it in memory

The full specification is included in appendixA

5 Risks

This product is advertised as a working calculator with function and variable memory. Every time the product does not comply with these expectations, this causes a risk, since people might not double check the answers, have another calculator or have a backup of the inserted data. This makes the following scenarios risks:

- The calculator gives a wrong answer. A risk since the user might not notice this.
- The calculator crashes. A risk since valuable function and variable memory can be deleted.
- The calculator gives an error. A minor risk since the user knows the product can not comply, no data is lost and often there is another calculator.

Finding those scenarios will be a main focus in our testing.

There are no risks in the development process. The development process is already finished.

Risks in the test process are scenarios which will make the test process to lengthy. Since we know an average test case will not take long (typing a few lines), the only risky scenario is when we have too many test cases. If this happens we will either start automating or lower our goal of testing every part. That being said, we do not expect this to happen.

6 Test Environment

The tests will be performed on a single PC with JRE 1.8. The SUT will be tested using manual tests. No other software is required.

7 Quality Characteristics

If we follow the quality characteristics of ISO 9126, then functionality and reliability would be our main focus. Here follows a per characteristic description of if we need testing for it:

Functionality will be tested thoroughly. We have a complete and detailed specification of the SUT, to which the SUT should comply.

Reliability will be a key point. Fault tolerance and recoverability are important to us, for the reason that on this area the SUT does not comply with our initial expectations. Some simple experimenting showed that our SUT could crash at some input, instead of just giving an error. We want to test in which cases this happens.

Usability is of little concern to us. Since the SUT is meant as a commandline calculator and is not meant for the larger public, the usability expectations are low. The SUT already surpasses our expectations.

Efficiency will not be tested. The SUT is so small that to the user every calculation will seem as instantaneous, regardless of the actual efficiency.

Maintainability will not be tested. The difficulty to adapt the SUT does not interest us.

Portability , the last characteristic. This will also not be tested. The reason for this is because we already know the portability. Every computer with command line can use the SUT.

8 Levels and Types of Testing

The SUT can be split into clear units. There is one unit for each predefined variable and function. Furthermore we have units for new variables, equality tests, new functions and expression parsing. These units can be tested separately and in combinations. Together they form the system, with its specification as counterpart.

We are only interested in the capabilities of our SUT, not in how one would fix found shortcomings. Therefore we do not test white box. We will be doing validation testing, using the specification.

9 Who will do the Testing

Test will be performed by independent testers. This is because little knowledge is needed to preform the test and those already familiar with the program may be biased.

10 Test Generation Techniques

The Black-Box tests will first be generated by error guessing based on the specification. The test suite will be expanded using partitioning of the input values, and applying boundary value analysis on these partitions.

11 Test Automation

Tests will be done manually. Each test is described in a spreadsheet.

To start a test, a new spreadsheet is copied from a template.

This template contains:

- One or more lines of input to be entered in the program

- A description of the expected output
- A field to insert the resulting output after entering the input.

The template is updated if and only if the tests changed.

If the resulting output is equal to the expected output, the test is successful. If not the test is unsuccessful.

The results of the test are recorded in the the spreadsheet. A new spreadsheet is created for each testing session.

After the testing session is over, the resulting spreadsheet shall be send via email to the developer team for review.

12 Exit Criteria

In the section 8 we told about the units our SUT could be split in. We consider every combination of units as a part that should be tested. Per such a part we will consider ourselves finished if we feel certain that we can predict every outcome for every input. Since the SUT is defined in a context-free grammar, this certainty can be achieved. When all parts have been tested in such a way, we consider ourselves finished.

13 Testware

As described in section 11, the results, with the description of the test are written to a spreadsheet. The resulting spreadsheet is the test product and shall be stored in a digital format.

14 Issue Registration

After a testing as described in section 11 the developer team receives an email with a spreadsheet attached.

This spreadsheet will inform the developer if any tests have failed and if so what tests have failed and what happened when they failed.

The developer then can make bug reports based on unsuccessful test and fix them in the future.

A Specification

A.1 Overall use of the program

the program will continuously prompt the user for input. It will continue to prompt the user for input until it receives the `exit()` function as input and then terminates.

On invalid input the program the program will show an error indicating the nature of the invalid input, but will not terminate.

Valid input is on of the following

- A function assignment
- A variables assignment
- An expression
- An equality test

A.2 Memory

The program will keep defined variables and function in memory until the program terminates.

The program will be initialized with the following variables

- `pi`, whose value is `3.141592653589793`
- `e`, whose value is `2.718281828459045`

The program will be initialized with the following functions

- `floor`, which rounds down
- `ceil`, which rounds up
- `round`, which rounds to the nearest whole number.
- `log`
- `ln`, which
- `sqrt`, which returns the square root of its
- `root`, which returns the Nth root of it first argument. (WARNING: does not work properly)
- `exit` , which will terminate the program

BTW floor,ceil and round work incorrectly for NEGATIVE numbers

A.3 Expression

A valid expression is defined using Context-free Grammar

Expression → Seperator Expression Seperator
Expression → (Expression)
Expression → Number
Expression → Variable
Expression → PrefixFunction Seperator (Expression)
Expression → Expression Seperator InfixFunction Seperator Expression
Seperator → SPACE — ϕ — Seperator Seperator
Number → 0 — 1 — 2 — 3 — 4 — 5 — 6 — 7 — 8 — 9 — Number Number
Variable → Name
PrefixFunction → Name
Name → SmallLetter — CapitalLetter — NameName
SmallLetter → a—b—c—d—e—f—g—h—i—j—k—l—m—n—o—p—q—r—s—t—u—v—w—x—y—
CapticalLetter → A—B—C—D—E—F—G—H—I—J—K—L—M—N—O—P—Q—R—S—T—U—
InfixFunction → / — * — - — + — ˆ
variable names have a max length

A.4 Variable assignments

A variable assignment is in the form
`VARIABLENAME = EXPRESSION`

Where a valid `VARIABLENAME` is a string of 1 or more letters and is case-sensitive.

A.5 Equality tests

An equality test is in the form
`EXPRESSION1 = EXPRESSION2`

Where `EXPRESSION1` and `EXPRESSION2` are valid expressions and `EXPRESSION1` is **not** the name of a variable

A.6 Functions assignments

A function assignment is in the form
`FUNCTIONNAME(ARGUMENT):=EXPRESSION`

A.7 Expression parsing

Intermediate steps and calculations are show The result of the expression is shown

For instance an expression in the form `(1 + 2) * 4` will require the following actions.

- Evaluate `1 + 2` and print its result, (`3` in this case).
- Evaluate `3 * 4` and print its result, the `3` being the result of the previous evaluation.
- Print the results of the entire expression, `12 in this case`, after all intermediate calculations have been done.