Title

Sasja Gillissen, Martin Huijben, Martijn Terpstra

September 26, 2016

Contents

1 Tasks

1.1 Write requirements

1.1.1 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

1.1.2 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

1.1.3 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|z
CapticalLetter → 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|Z
InfixFunction → / | * | - | + | ˆ
variable names have a max length

1.1.4 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.

1.1.5 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

1.1.6 Functions assignments

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

1.1.7 Expression parsing

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

1.2 Design test cases

1.2.1 Functions assignments

1.2.2 Variable assignments

1.2.3 Equality tests

1.2.4 Expression parsing

1.3 Assign Test cases

1.4 Execute test cases (automation?)

2 Short requirements

From this description we can deduce that a test approach must at least describe:

- the product that will be tested,
- the controlled environment in which it will be tested,
- the specified procedure following which it will be tested,
- the quality characteristics that will be tested, and
- the specification.

3 Template

3.1 Introduction

State the objectives and overview of the document at a high-level.

3.2 Test Goal

What is the overall goal of the testing effort, what are the final deliverables, who are the stakeholders, i.e., for whom are you doing it, applicable laws and (international) standards.

3.3 The Product

Identification of the SUT: What is the product (SUT System Under Test) being tested, its version, its operation context, required platform, its interfaces, and how is it executed.

3.4 The Specification

What is the test basis, i.e., its specification, and all documentation describing what the SUT shall do. (Do not include specification documents, but refer to them.)

3.5 Risks

What are the risks of the product (at a high level), of the development process, and of the test process. How are risks handled and mitigated.

3.6 Test Environment

What is the (controlled) environment in which experiments are performed, what is the test architecture, i.e., how are SUT and test system positioned and connected, which environment and infrastructure (hardware, software, middleware, databases, libraries, . . .) are required for testing, how to access the SUT and its interfaces, which stubs and drivers are needed, are tests performed in a laboratory, production, or user environment.

3.7 Quality Characteristics

Which quality characteristics are tested (IS 9126 or other quality model: functionality, reli- ability, usability, . . .),

3.8 Levels and Types of Testing

Which levels and types of tests are performed: (V-model: unit, integration, module, sys- tem, acceptance, . . .), which units, components, subsystems, . . . are tested and for what, accessibility (white/black box), verification vs. validation tests, ...

3.9 Who will do the Testing

Who tests what, and what are the roles: developer, (independent) tester, user, alpha, certi- fication, . . .),

3.10 Test Generation Techniques

As far as already known or required, e.g., by applicable standards: black-box (equivalence partitioning, boundary value analysis, error guessing, cause-effect graphing, decision tables, state transitions, use case testing, exploratory testing, . . .), white-box (path, statement, (multiple) condition, decision/branch, function, call, loop, MC/DC coverage, . . .), mutation testing, combinatorial testing, ...

3.11 Test Automation

As far as applicable, which parts of the testing will be automated, which test tools will be used in the various phases of the testing process (planning, preparation, test generation, test execution, completion), which tests are performed manually, what is automated, and which tools have to be obtained or developed.

3.12 Exit Criteria

What are the criteria for going from one test phase to the next, when is testing finished, when is the product considered sufficiently tested, what are the (final) evaluation criteria.

3.13 Testware

Which test products are recorded, consolidated, and kept for reuse.

3.14 Issue Registration

How are issues (defects) registered, analysed, reported, and handled.

4 Observations martijn

4.1 Overall

- Comments only in util/BigFunctions.java

4.2 Expression that crash

should the program CRASH on wrong input?

- $1/0$
- $log(-1)$
- $\phi = 7$

- Using TABS in any expression
- using arrow keys

The program crashes on some but not all wrong input.

- $seven + eight$ does not crash.

4.3 Expression with a wrong result

- $(1/300) * 300 = 0.9999999999999900.$ should be 1
- $2^(- 1) = 0.$ should be 0.5
- $log(10^1234)$ outputs an intermediate result, then crashes. should output 1234
- 0^0 return 1, is undefined

4.4 Other

- Input reading is primitive, cant go back without deleting.
- `ln(log( 10^e ))` throws an error Could not convert bigInteger into long, but still gives the correct answer
- `1 Banana \phi = 1`
- Control + \ gives a lot of debugging information
- `x=7 7(x)` gives `7`, not `49`
- `f(x):=7`, works fine `f(x):=f(x)+1`, work if the function f has previously been defined `f(1)` crashed due to an infinite loop
- Using 9999 character variable names gives an exception, 999 characters is fine.
- `x(z):=z f(x):=x() f(7)` crashes
- `f(x):=1 f(x):=f(2) f(3)`, crashes, expected `1`
- Equality?, can either return -1,0, or 1
- `z=7` $\neq$ `7=z`