COMP 354 Software Engineering

ASSIGNMENT 1: SOFTWARE REQUIREMENTS SPECIFICATIONS

FALL 2021

Annika Timermanis
George Mavroeidis
Jahrel Stewart
Axel Solano
Phuong Anh Trinh
Jordan Chan Kum Sang

11 October 2021

Dr. Rajagopalan Jayakumar
Farbod Farhour

Introduction

Our team's goal is to provide an environment that will allow young children to learn basic fundamentals of coding. Our team has chosen to develop an editor that is capable of transforming a natural language by passing keywords into predefined algorithm constructs, and returning the correct results. Our software will take sentences written in the English language as input, and return the equivalent code output. Our model software will target a calculator's functionality, and will be extendable. Our requirements are listed as follows.

Our Requirements

- General Algorithm Template: valid, error, etc.
  - Scan key words and use a dictionary
  - Unknown words or not find appropriate words
  - Mixture of words that cannot identify a calculation
- Variable assignment: Assign a word or value to a variable
  - If variable is not predefined → assignment error
  - If variable already exists → assignment is accepted
- Calculator-related constructs:
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Square-root
- Loop-based construct:
  - While loop
  - For loop

- Condition-based construct:
  - If a number is even or odd
- Additional features:
  - Option to exit
  - Option to enter
  - Option to modify

Intended Audience

The intended audience for this software development project is targeted towards children of all ages. The purpose of this project is to really provide an opportunity for kids who do not have any programming knowledge, and wish to learn some basic functionalities by simply typing in commands that are written in English sentences. As a team, we really want to immerse ourselves and hope to meet the requirements of both our target audience and other stakeholders in the process of developing our software.

Contribution Of Team

Pages 1-5 were completed by all team members.

Student Name - Student ID	**Pages Contributed**
Phuong Anh Trinh - 40069870	Page 6, 10
George Mavroeidis - 40065356	Page 7
Axel Solano - 40046154	Page 8
Annika Timermanis - 40131128	Page 9
Jordan Chan Kum Sang - 40125997	Page 11
Jahrel Stewart - 40115728	Page 12

Use Case

Use Case	• Understand the fundamentals of coding via calculator's functionality of text editor.
Primary actor	• Children.
Goal in context	• To experiment with algorithms by writing English sentences.
Preconditions	• Predefined set of algorithm constructs. • A library of executable implementations of these algorithm constructs. • An algorithm simulator which calls the above library to execute the algorithm. • A simple text editor to enter/modify the algorithm using the algorithm constructs and simulates/ executes the entered algorithm and displays the results.
Trigger	• The children decide to simulate algorithms by some calculator's operations.
Scenario	• Children enter single-lined-sentence written in English language, containing keywords (a list of predefined keywords will be provided to the child). • The system displays the a result of the input or a (mathematical) solution based on one of the predefined functionalities.
Exceptions	• The actor tries to assign a value to a variable that is not predefined - explored further in **General Algorithm.** • The actor enters gibberish, or unrecognizable english sentences -

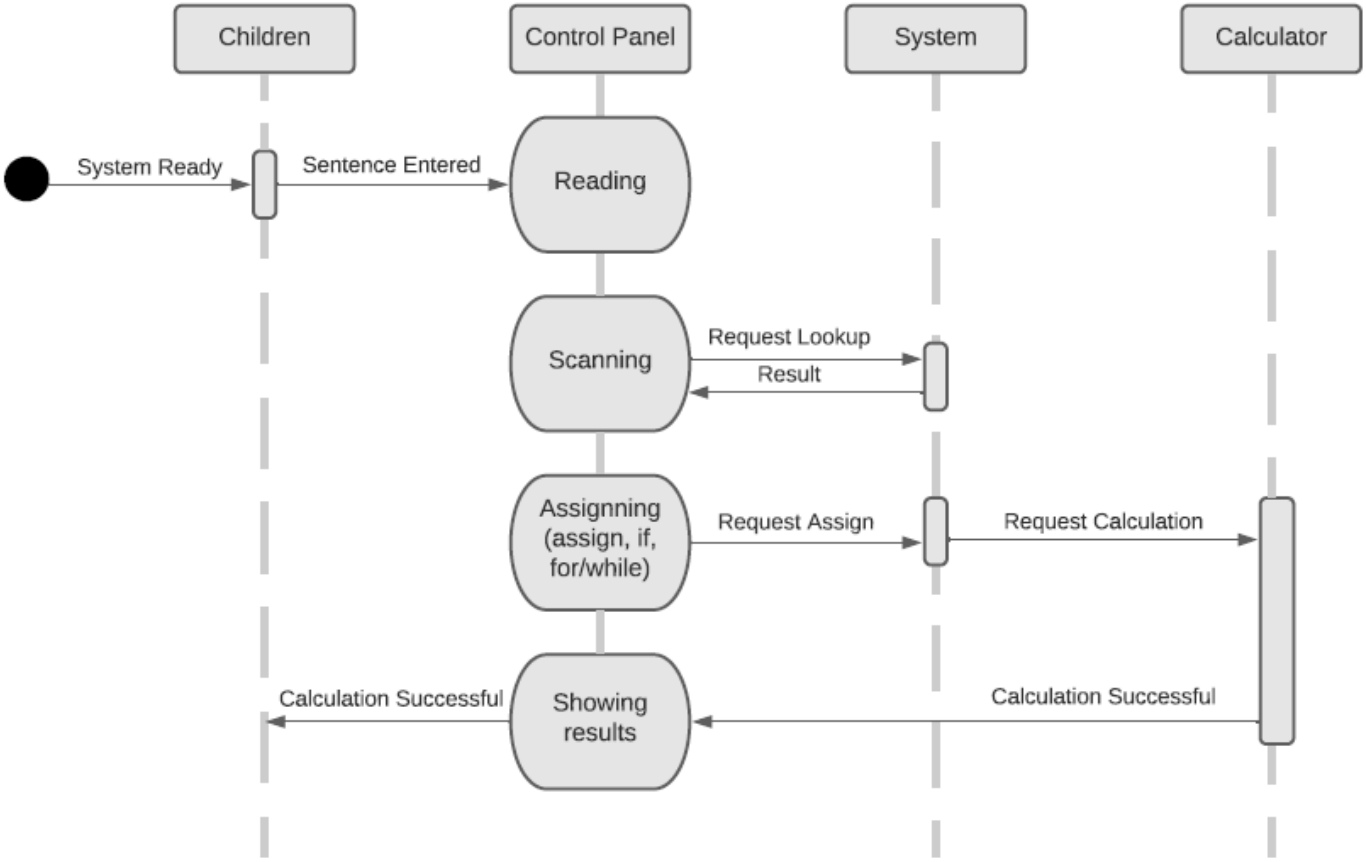
	explored further in **General Algorithm.**
Priority	• Modularity between the algorithms and the calculations/ features.
When available	• Our final assignment.
Frequency of use	• Tentative use, optional by the user.
Open issues	• If a user is particularly young, and may have difficulty with writing plain english, they might be prone to entering invalid English sentences or gibberish. • Will our software be easy enough to use even for younger children? • Will our editor be capable of redirecting and handling exceptions and issues regarding user input?
Will the actor have to inform the system about changes in the external environment?	• Not necessarily, as our software is directed at kids, it will be a very interactive child-friendly environment. • It will be designed to be extendable however.
What information does the actor desire from the system?	• The actor may wish to gain knowledge of programming fundamentals. • Learn the basics of coding.

Software Flow: UML Sequence Diagram

The first event, *system ready*, is derived from the external environment and channels behavior to the **Children** object. The child enters an English sentence. A *request lookup* event is passed to **System**, which looks up keywords in a simple database and returns a result to **Control Panel.** If the *scanning* state is completed, *assigning* state will be continued that activates a *request assign* to **System** and a *request calculation* to **Calculator** afterward.

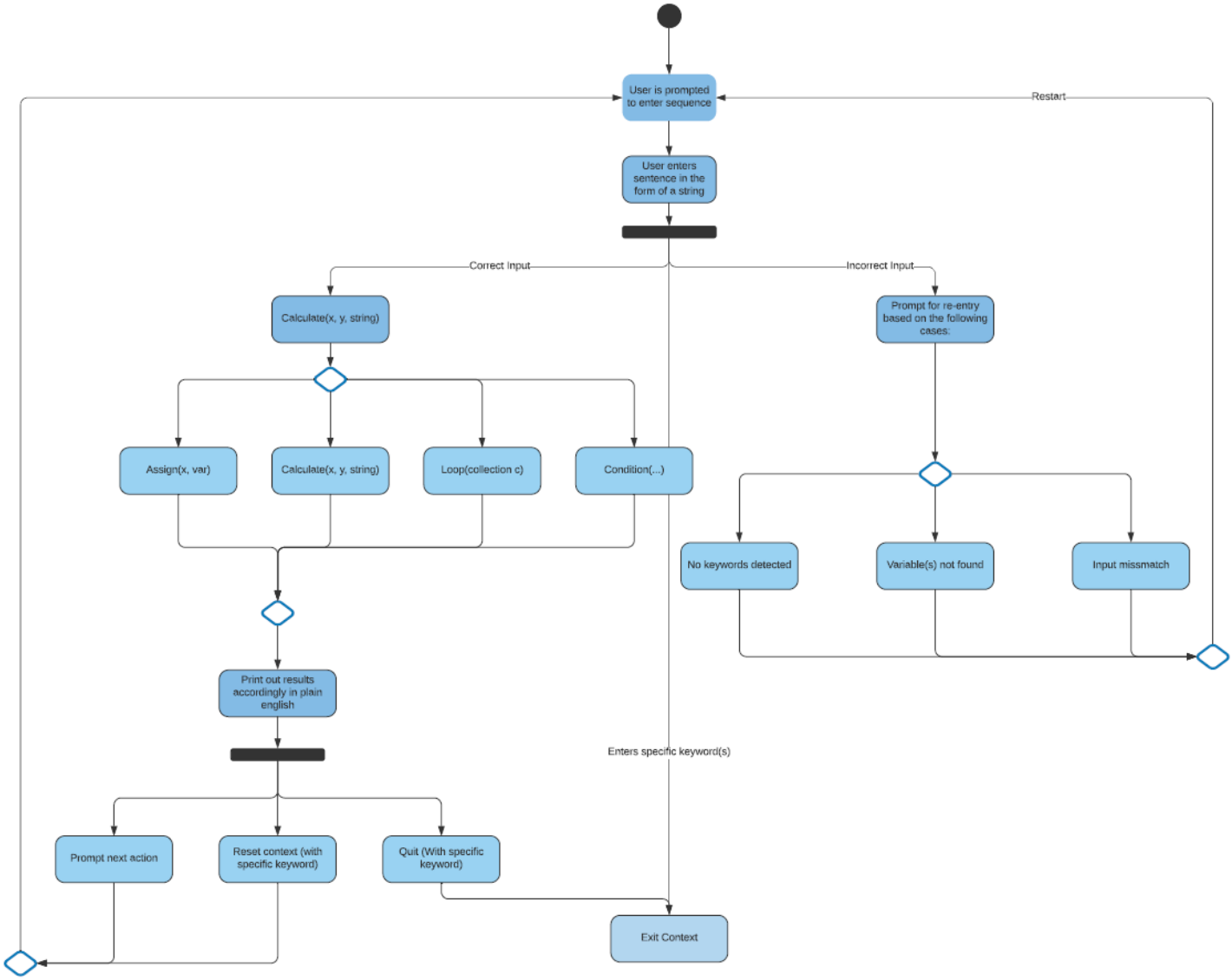
For example, those sentences could be entered:
“Assign x to 4 and divide by 2.”
“Add 8 to 5.”
“Multiply 2 to x for 10 times.”
“Square root of 9.”
“If x is odd/even, then subtract x to 6.”

If the calculation step is successful in **Calculator**, **Control Panel** will perform *showing results* state, and **Children** will be able to see them on screen.



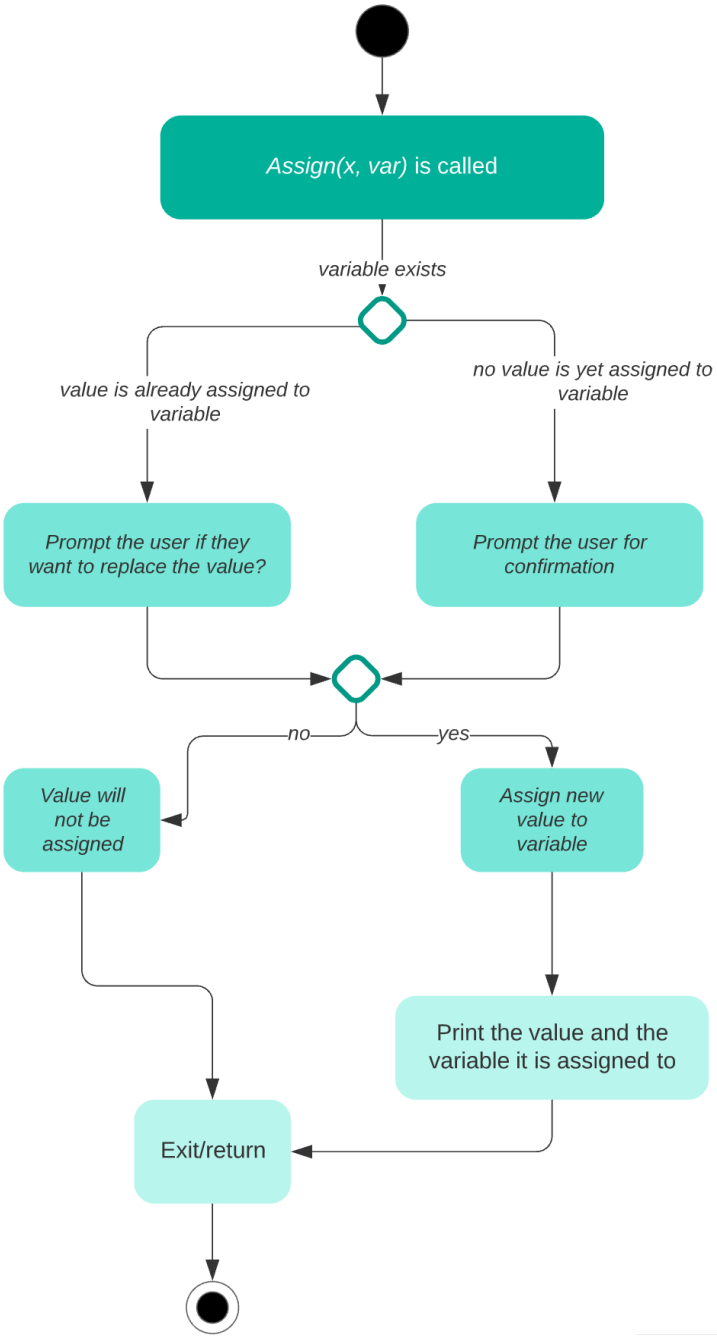
General Algorithm: UML Activity Diagram

To achieve modularity, a root template will be used to construct the calculator related constructs, using a dictionary of words to identify valid calculations. Different cases are handled, whether the input is valid, invalid or to execute a specific program operation. An option to close the program and restart the calculation context allows for a flexible use of the program by the user. There are three types of errors for all operations: 1) when a special keyword is not detected and cannot process any of the algorithm features 2) when a variable keyword used for operations does not exist or is invalid and 3) when there is an input mismatch during a variable calculation (ex: an integer subtracted from a string). Once these errors are detected, the user is prompted to re-enter his command, but in a correct manner. This will help kids understand what their error is and how to logically fix it. Once the user has successfully entered a calculation and outputs the desired results, a new choice of prompts will occur: 1) whether the user wants to make another calculation based on the context of the previous set of calculations 2) restart the calculation context by erasing the current memory and enter new variables and calculations 3) Quit the program.



Variable Assignment: UML Activity Diagram

In order to assign a value/word, or the result of a calculation, the function *Assign(x, var)* is called after having gone through the general algorithm that checks for valid inputs x and var. That means that the variable must already exist when it reaches this stage. There are two cases: the variable has already been assigned a value/word or the variable has not yet been assigned a value. If the variable has assigned a value, we ask the user if they want to replace it or not. If the variable has no value assigned to it, the user is asked to confirm the assignment. Depending on their choice, the value may or may not be assigned to the variable.



Four Calculator-Related Constructs: UML Activity Diagram

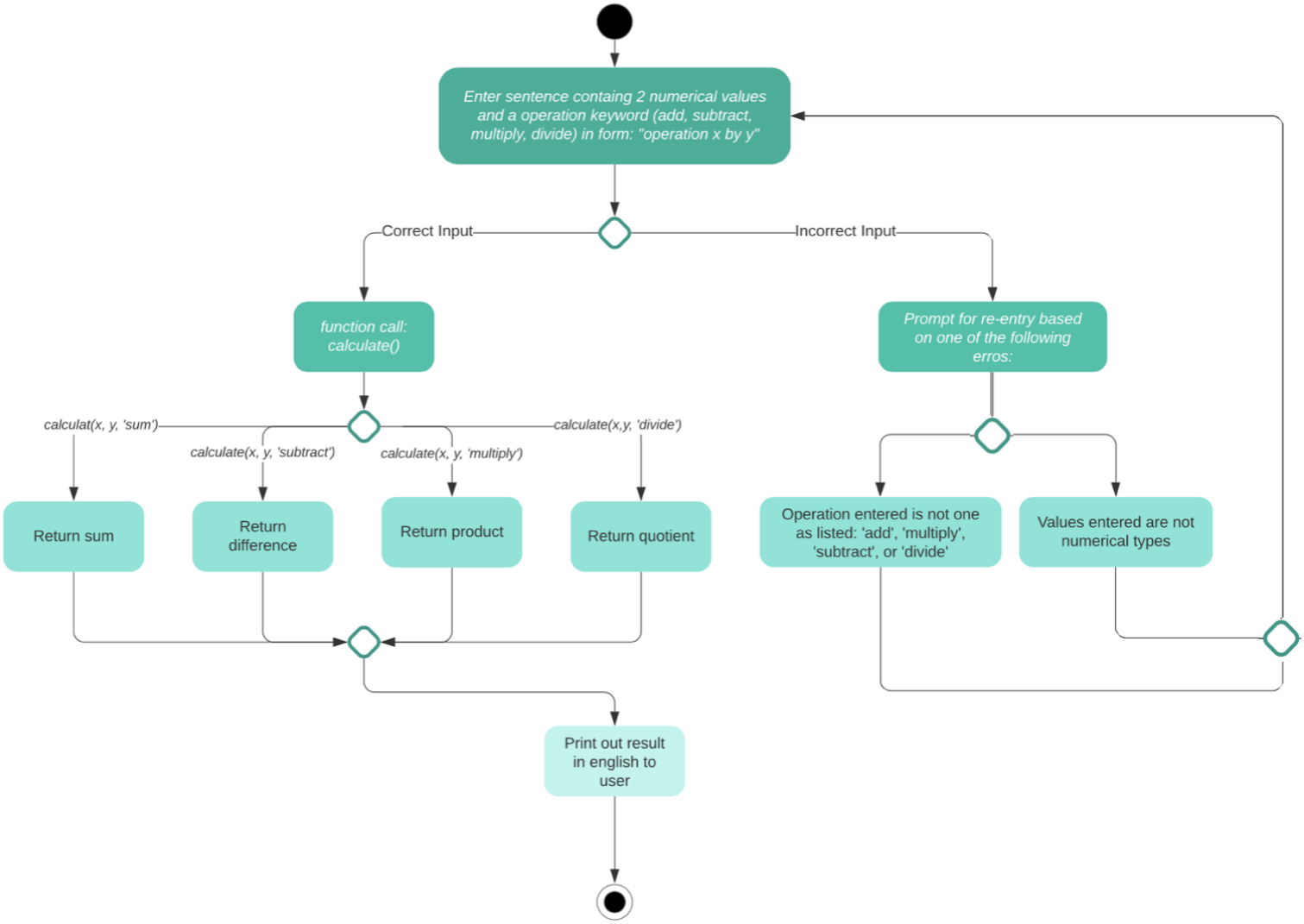
For our four main calculator functionalities, we will need to be able to take input of two numerical values and either add, subtract, multiply or divide them, and return the result to the user. A UML activity diagram can be used to represent processing details, and provides a graphical representation of the flow of interaction within a specific scenario. Here I will highlight our four scenarios grouped under a single activity. We can express this with code using a single calculate() function, which takes two numerical (integer or double) values, *x* and *y,* as well as a string, *operation* (‘add’, ‘subtract’, ‘multiply’, ‘divide’), as inputs:

def calculate(x, y, operation):

This function will first check the string passed, to determine which out of the four operations it will be performing, then return the result of the passed operation to the user in english.

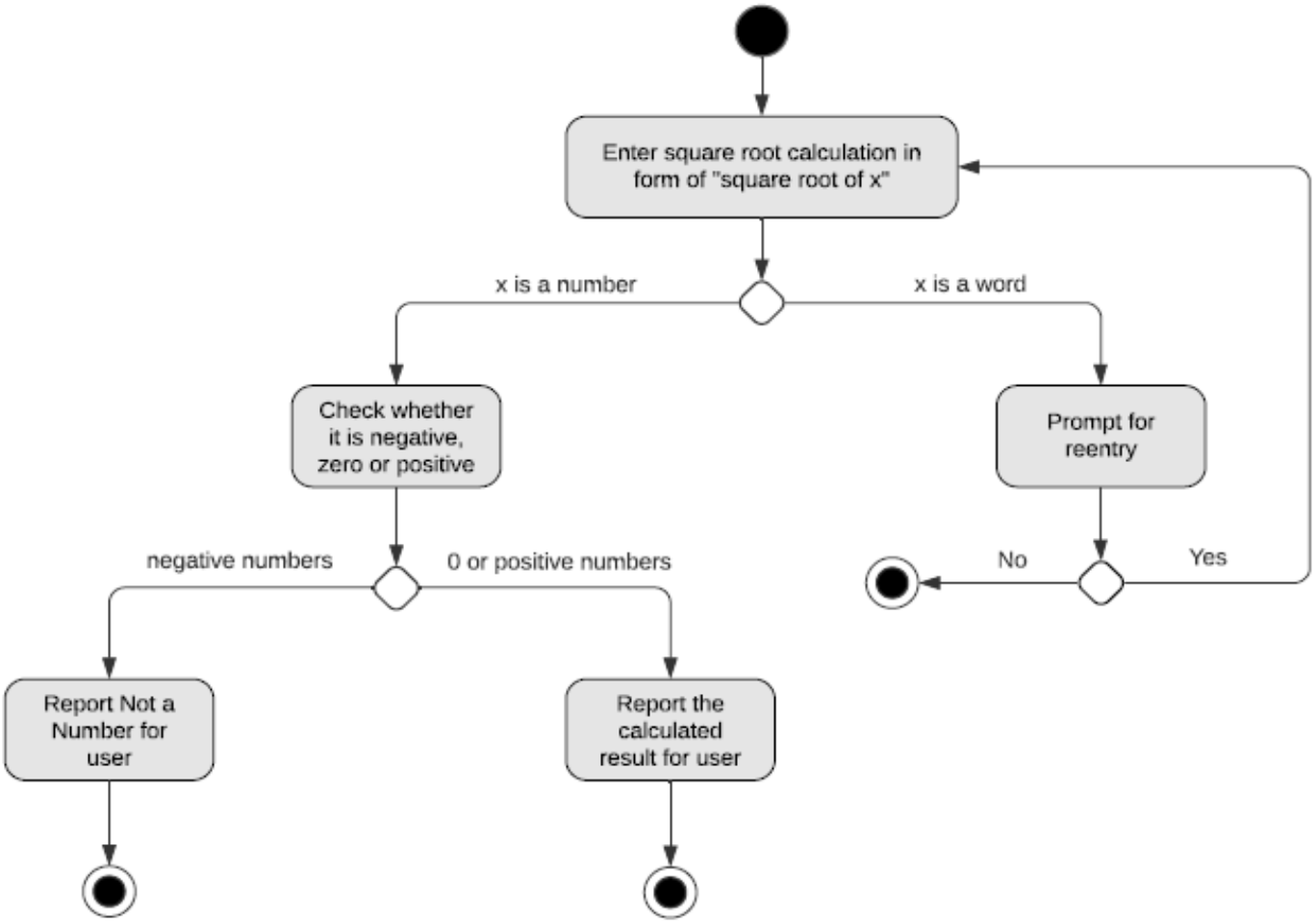
Examples of acceptable english inputs by user:

“Operation x by y”
“Operation x to y”
“Operation x and y”



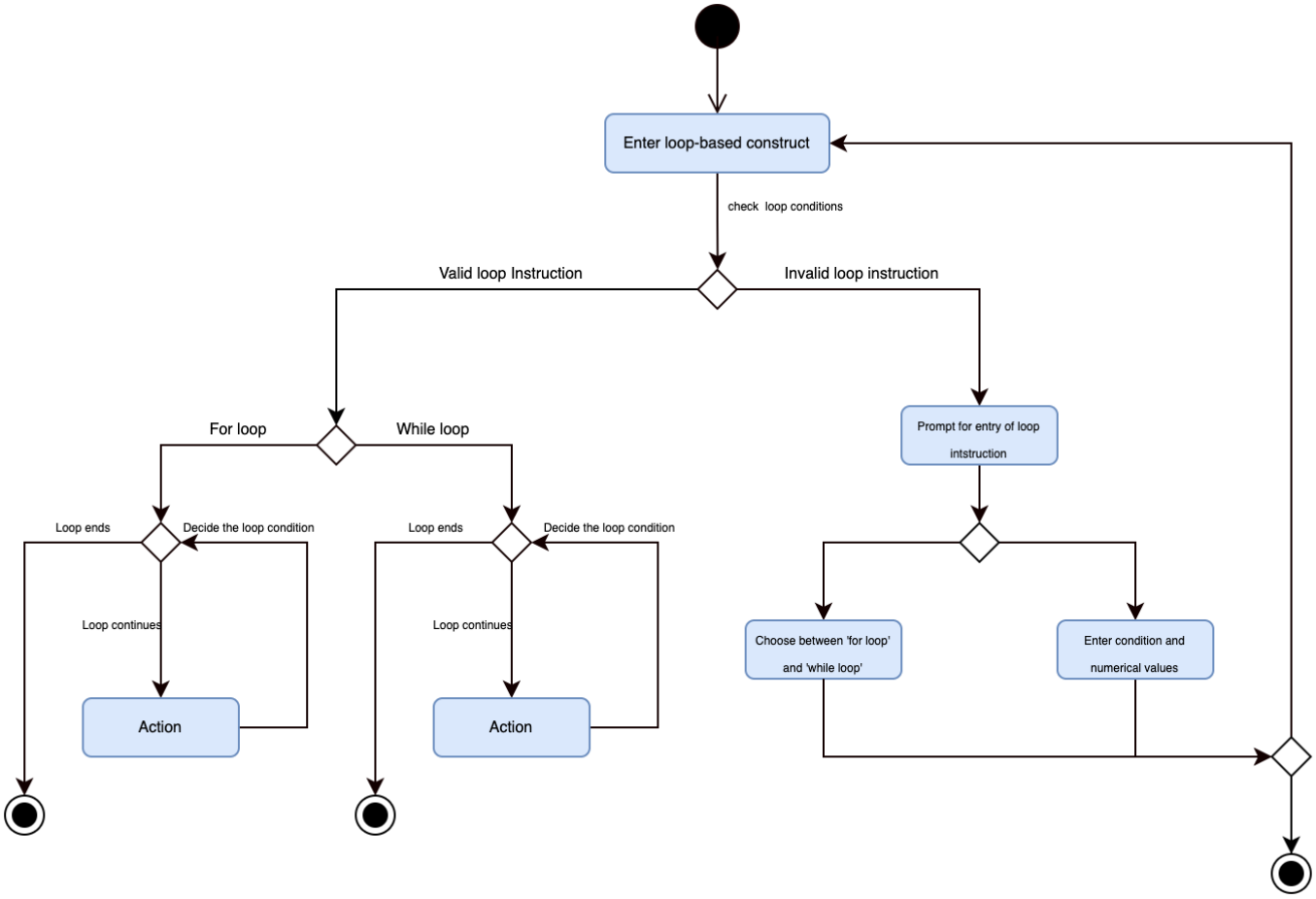
Square-Root Functionality: UML Activity Diagram

In order to perform square root operation, a sentence in the form of “Square root of x” has to be written. If the user enters x as a word, the text editor will prompt them to reentry the sentence. If the sentence is written in the right form, the input will be checked whether it is a positive, zero or negative number. In case x is negative, “Not a number” will be reported on screen. Otherwise, the calculated results will be shown.



Loop-Based Construct: UML Activity Diagram

The loop-based construct consists of two types of loops: for loop and while loop. The user has the choice between the two loops to perform an operation. Firstly, the user must enter either for loop or while loop with a valid condition else the program will prompt for input again. The algorithm will then proceed to execution of calculator instructions for the for loop or while loop. After each iteration loop, the condition is going to be compared until it becomes ‘false’ and the loop terminates.



Condition-Based Construct: UML Activity Diagram

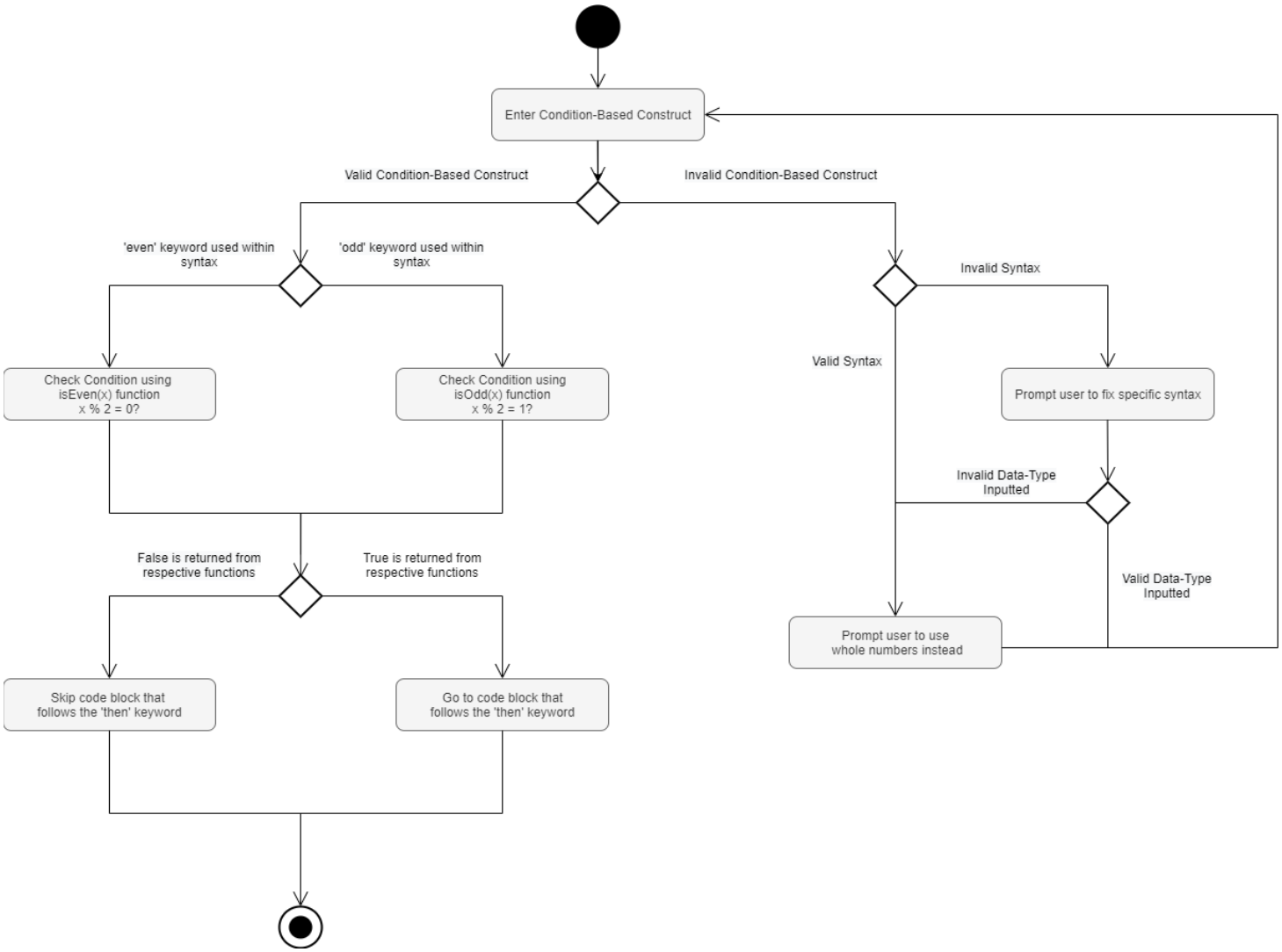
To perform a condition-based construct catered towards even/odd conditions, the user should have their condition, written as *[integer] is even* or *[integer] is odd*, placed between *'if'* and *'then'*. Once that is done, a check would be performed to assess the validity of the condition. If this check returns true, the english-like code following the *'then'* keyword is executed. If false, it will skip the code following the *'then'* keyword, executing only what comes after the entire condition-based construct.

Examples of acceptable english inputs(code) by user:

- If 4 is even then [arbitrary code]
- If 1 is odd then [arbitrary code]
- If 5 is even then [arbitrary code]

Examples of unacceptable english inputs(code) by user:

- If 0.54 is even then [arbitrary code]
- If “hey” is odd then [arbitrary code]
- 5 is even then [arbitrary code]
- If 5 is even [arbitrary code]
- If 5 is even then
- 5 is odd



References

[1] Roger Pressman and Bruce Maxim, Software Engineering: A Practitioner's Approach, 9th Edition, McGraw Hill, 2020, ISBN 13: 9781259872976.

[2] Diagrams created in Lucidchart, www.lucidchart.com