ECE 9200 Assignment Sample Test Cases

Group

Name: Bao Nam Nguyen **– Student ID**: 251462627 **– Email**: bnguye88@uwo.ca

Name: Ahmad Abo Rawi **– Student ID**: 251469096 **– Email:** aaborawi@uwo.ca

Name: Zhiyuan Wu **– Student ID**: 251462080 **– Email:** zwu697@uwo.ca

Name: Ruilin Peng **– Student ID**: 251067921 **– Email:** rpeng25@uwo.ca

Simple 4-Function Calculator (Infix without Precedence)

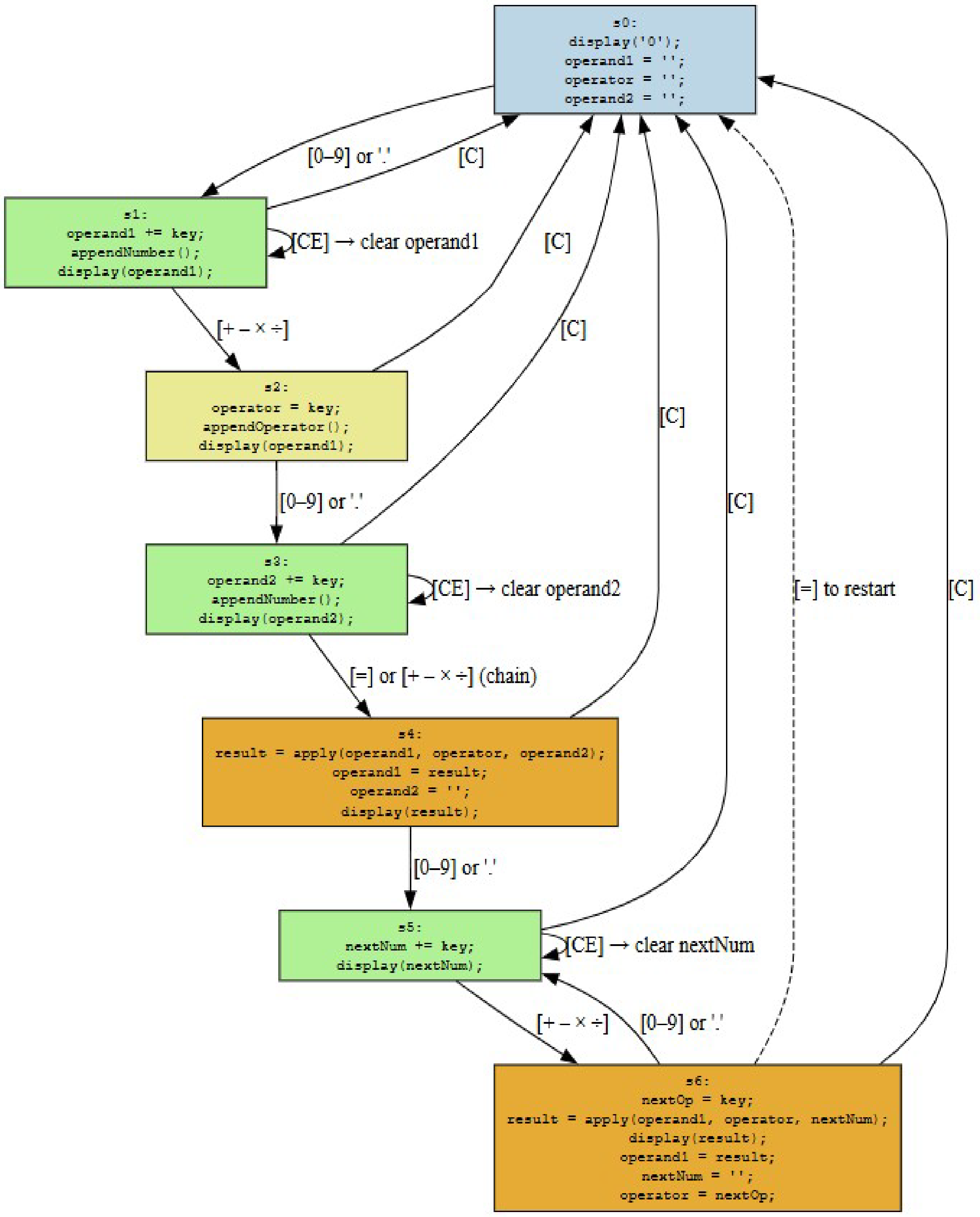
First, power on the calculator, 0 will be displayed as an indicator for the calculator to be ready for use "start". When the user presses a digit key from 0 to 9 to enter the first number, the display changes from zero to the pressed number. If the number is a floating number, the user presses the dot key to add to the current displayed number, and presses another digit from 0 to 9, and the display will update accordingly. Once the first number is entered, the user presses one of the operators ['+', '-', 'x', '/'], the display will continue showing the first number while storing the operator internally. Next, the user enters the second number by pressing the digit key or the decimal point, and the display shows the second number. If the user presses another operator instead of equals, the calculator computes the results of the previous operation and displays them, then stores the new operator for the next operation. When the user presses the equals operator, the calculator will compute the last operation and display the final result. At any point, if the user makes a mistake, they can press the clear entry button to erase the current number or the clear key to reset the entire calculation to 0. Throughout the process, each key entry is logged with a timestamp.

Translate the scenarios to tables:

input	Output
start	0
5	5
+	5
3	3
=	8

Input	Output
start	0
5	5
+	5
3	3
x	8
2	2
=	16

Input	Output
start	0
7	7
.	7.
5	7.5
+	7.5
1	1
CE/C	0



```
s0:
display('0');
operand1 = '';
operator = '';
operand2 = '';
```

```
s1:
operand1 += key;
appendNumber();
display(operand1);
```

```
s2:
operator = key;
appendOperator();
display(operand1);
```

```
s3:
operand2 += key;
appendNumber();
display(operand2);
```

```
s4:
result = apply(operand1, operator, operand2);
operand1 = result;
operand2 = '';
display(result);
```

```
s5:
nextNum += key;
display(nextNum);
```

```
s6:
nextOp = key;
result = apply(operand1, operator, nextNum);
display(result);
operand1 = result;
nextNum = '';
operator = nextOp;
```

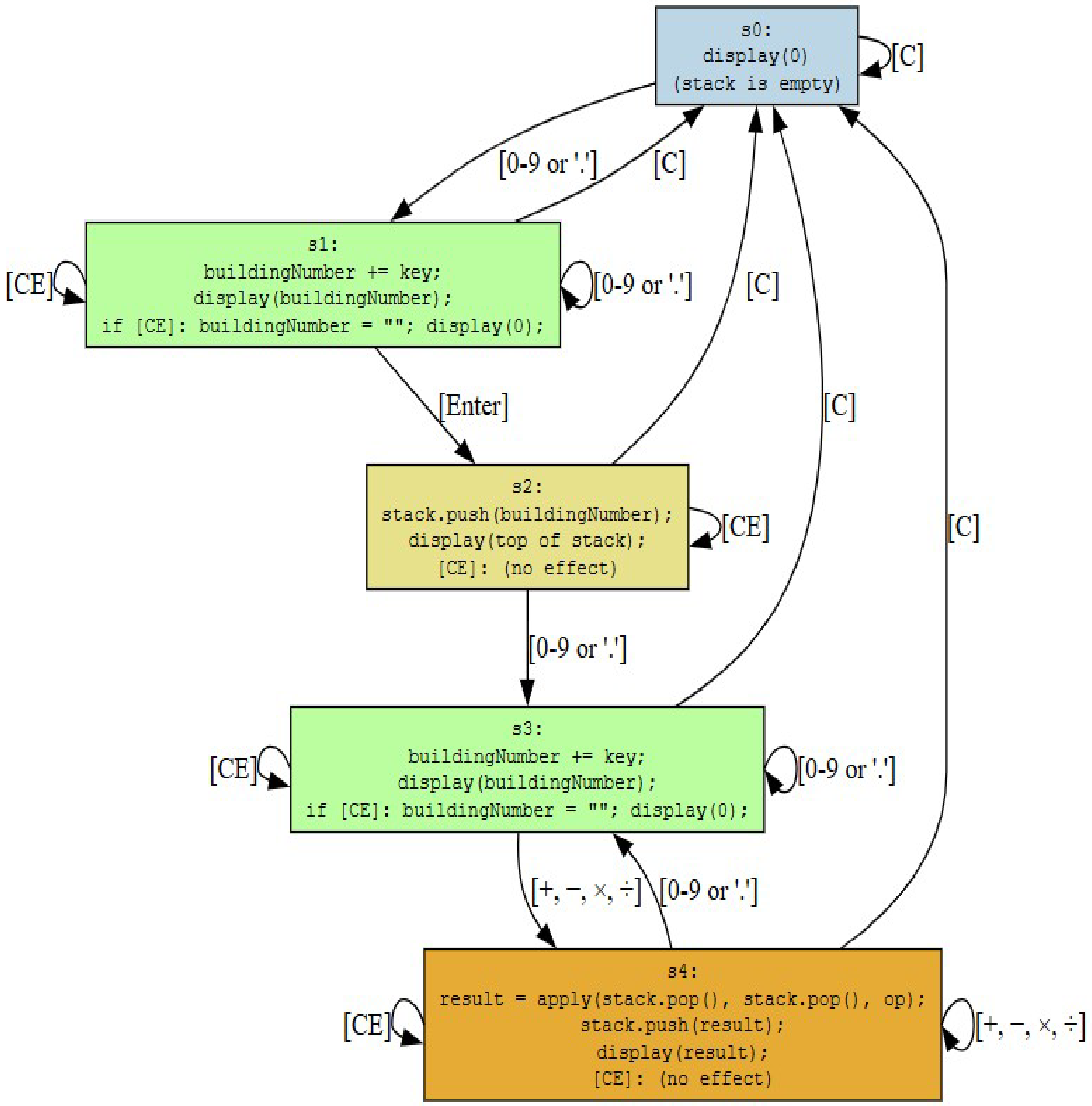
Simple 4-Function Calculator (RPN Mode)

First, the user powers on the calculator, which will display zero. The user starts the calculation by pressing the digit key [0...9] to enter the first number, and the display will update to show the pressed number. For decimal values, the user adds the decimal point at the appropriate point, and the display will update accordingly to reflect the user's entry. Next, the user presses the Enter key to push the entry to the internal stack, and the display will continue showing the previous entry. The user then enters the second number by pressing a digit or the dot if it's a floating-point number. The display will then be updated to show this entry. To perform the calculation, the user will press one of the operator keys [+, -, x, /], which will pop the two numbers from the stack, apply the operation, push the result back, and the display will update to show the result. This process will be repeated for chained operations. There is no equal's key; the operations are calculated immediately after pressing the operator. If the user entered the wrong number, he could press the clear entry key "CE" to erase the current number, or the clear key to empty the stack, and the display will reset to zero. Throughout the process, each key entry is logged with a timestamp.

input	Output
start	0
5	5
enter	5
3	3
+	8

Input	Output
start	0
5	5
enter	5
3	3
+	8
2	2
x	16

Input	Output
start	0
7	7
.	7.
5	7.5
enter	7.5
3	3
C/CE	0


s0:
display(0)
(stack is empty)
[C]
[0-9 or '.']
[C]
s1:
buildingNumber += key;
display(buildingNumber);
if [CE]: buildingNumber = ""; display(0);
[CE]
[0-9 or '.']
[C]
[Enter]
[C]
s2:
stack.push(buildingNumber);
display(top of stack);
[CE]: (no effect)
[CE]
[C]
[0-9 or '.']
s3:
buildingNumber += key;
display(buildingNumber);
if [CE]: buildingNumber = ""; display(0);
[CE]
[0-9 or '.']
[+, −, ×, ÷]
[0-9 or '.']
s4:
result = apply(stack.pop(), stack.pop(), op);
stack.push(result);
display(result);
[CE]: (no effect)
[CE]
[+, −, ×, ÷]

4-Function Calculator with Precedence and Parentheses

First, the user powers on the calculator, which will display zero. The user starts the calculation by pressing a digit key [0...9] to input the first number, and the display will update immediately to reflect the user's entry. For floating point, the user will press the dot key, and the display will be updated accordingly. To override the precedence, the user presses the open parenthesis key. The display then clears the screen to 0. Once the user has finished entering numbers and operators, pressing the closing parenthesis key calculates the operations inside the parentheses and displays the results. If there are any operations before the parentheses, pressing the equals key will calculate the operations. Division and multiplication have higher precedence, so they are computed first unless parentheses dictate otherwise. This process can chain multiple operators and parentheses for more complex operations, with the calculator automatically handling precedence by evaluating higher-priority operations first. If the users made an error, the clear entry key removes the current entry, while the clear key resets the operations to zero. Throughout the process, each key entry is logged with a timestamp.

Input	Output
start	0
5	5
+	5
3	3
x	3
2	2
=	11

Input	Output
start	0
(	0
5	5
+	5
3	3
)	8
x	8
2	2
=	16

Input	Output
start	0
(	0
7	7
.	7.
5	7.5
+	7.5
3	3
)	10.5
CE/C	16

Test case (a): **What is the difference between the following two numbers: 57 times 13 and 161 divided by 7.**

This means calculating as: (57 × 13) − (161 ÷ 7)

1. Compare the usability of three versions.

Version 1: INFIX Calculator

- Input Method: Enter 57, ×, 13, =, then write down the result, clear, enter 161, /, 7, =, then subtract manually. ({‘57’ ‘x’ ‘13’ ‘=’}, result: 741), ({‘161’ ‘/’ ‘7’ ‘=’}, result: 23), (741 – 23 = 718)
- Usability: Reasonable but lacks multi-step evaluation. Users must calculate sub-results one at a time, then subtract manually.
- Pro: Familiar design works like most standard calculators.
- Con: Easy to lose track of interim values.

Version 2: RPN (Reverse Polish Notation)

- Input Method:
    - {‘57’ ‘Enter’ ‘13’ ‘×’}, (result: 741)
    - {‘161’ ‘Enter’ ‘7’ ‘/’}, (result: 23),
    - then subtract using −: {‘741’ ‘enter’ ‘23’ ‘-‘}, (result: 718) .
- Usability: Efficient if user understands RPN logic.
- Pro: No need for parentheses; operations are stack-based.
- Con: Highly unintuitive for users unfamiliar with RPN. No direct guidance on expression structure.

Version 3: Order of Operations (with parentheses)

- Input Method: ‘(‘ ‘5’ ‘7’ ‘×’ ‘1’ ‘3’ ‘)’ ‘−‘ ‘(‘ ‘1’ ‘6’ ‘1’ ‘÷’ ‘7’ ‘)’ ‘=’.
- Usability: Most powerful and flexible. Allows exact input of expression as stated.
- Pro: Intuitive for those used to math notation. Handles complex expressions.
- Con: Requires care with parentheses and full expression entry.

2. Which interaction mode allows the user to perform their calculations with minimal errors?

Version 3 (Order of Operations)

- Why? It supports the full mathematical structure directly, reducing user memory burden and potential missteps.
- Parentheses clarify grouping, and operator precedence is handled internally.

3. Which interaction mode allows them to perform their calculations more rapidly?

Version 2 (RPN) – *for trained users*

- Why? Once you know the order (a b op), RPN can be faster because it eliminates the need for parentheses and interim equals.
- For non-expert users, Version 3 may be faster due to its intuitive layout.

4. Which do they find more satisfying?

Subjectively: Version 3 > Version 1 > Version 2

- Version 3 gives users confidence that their expression is structured exactly as intended.
- Version 1 feels familiar but clunky for multi-step problems.
- Version 2 may feel confusing or opaque to most casual users.

5. Which are unintuitive? In other words, which are more usable?
Most intuitive/usable → Least: Version 3 > Version 1 > Version 2

- Version 3 aligns with natural math notation – brackets, order of ops.
- Version 1 is standard but limited.
- Version 2 requires users to adopt a completely different model, making it the least intuitive.

Test case (b): **Double the number that results from the difference between 1023 and 127.**

This means calculating as: 2 × (1023 − 127)

1. Compare the usability of three versions.

Version 1 – INFIX Calculator

- Input method:
  - First: {'1023' '–' '127' '='} then store or remember result
  - Then: '×' '2' '='
- Usability:
  - Moderate. Requires intermediate result storage or quick mental tracking.
- Pro: Familiar interaction.
- Con: Requires *re-entering* previous result if not automatically carried forward.

Version 2 – RPN Calculator

- Input method:
  - {'1023' 'Enter' '127' '–'} → (result: 896)
  - {'Enter' '2' '×'} → final result
- Usability:
  - Efficient *if* the user knows RPN logic.
- Pro: Stack handles operations cleanly.
- Con: Still unintuitive for most users.

Version 3 – Order of Operations Calculator

- Input method:
  - {'2' '×' '(' '1' '0' '2' '3' '–' '1' '2' '7' ')' '='}
- Usability:
  - Excellent. Mirrors the original verbal structure exactly.
- Pro: Most natural and accurate way to enter this problem.

- Con: Slightly more effort in pressing parentheses.

2. Which interaction mode allows the user to perform their calculations with minimal errors?

Version 3 (Order of Operations)

- Supports grouping with parentheses.
- User directly mirrors the phrasing of the question.
- Less chance of user forgetting or mis-entering intermediate results.

3. Which interaction mode allows them to perform their calculations more rapidly?

Version 2 (RPN) – *for users experienced with stack logic*

- No need for parentheses or equals.
- Two operations executed sequentially.

Runner-up: Version 3 — intuitive but slightly slower due to expression length.

4. Which do they find more satisfying?

Version 3

- Users see and control the full expression.
- Result matches mental model.
- Confidence is high after execution.

5. Which are unintuitive? In other words, which are more usable?

From most intuitive/usable to least:

Rank	Version	Reasoning
1	**Version 3**	Matches natural phrasing, supports full expressions
2	**Version 1**	Familiar but clunky for nested expressions
3	**Version 2**	Efficient but requires conceptual shift in thinking

Test case (c): What is the area of a rectangle that has one side 789cm and the other side is 148cm longer than that amount?

This means calculating as: Area = 789 × (789 + 148) = 789 × 937

1. Compare the usability of three versions.

Version 1 – INFIX Calculator

- Input Method:
    - First: {‘789’ ‘+’ ‘148’ ‘=’} (get 937)
    - Then: {‘×’ ‘789’ ‘=’}, result: 739293
- Usability:
    - Fair. Requires tracking intermediate result manually or in memory.
- Pro: Familiar layout.
- Con: Multi-step logic requires re-entry of first operand unless user stores it mentally.

Version 2 – RPN Calculator

- Input Method:
    - {‘789’ ‘Enter’ ‘789’ ‘Enter’ ‘148’ ‘+’ ‘×’}
- Usability:
    - Clean execution once familiar with stack usage.
- Pro: No parentheses needed, operations stack in natural RPN order.
- Con: Least intuitive for general users, especially because it requires re-entering 789.

Version 3 – Order of Operations Calculator

- Input Method:
    - {‘7’ ‘8’ ‘9’ ‘×’ ‘(‘ ‘7’ ‘8’ ‘9’ ‘+’ ‘1’ ‘4’ ‘8’ ‘)’ ‘=’}
- Usability:
    - High. The entire expression can be entered in one pass.
- Pro: Closely mirrors the wording of the question.
- Con: Slightly slower entry due to longer expression and need for parentheses.

2. Which interaction mode allows the user to perform their calculations with minimal errors?

Version 3 (Order of Operations)

- Parentheses prevent misinterpretation.
- Reduces memory and re-entry errors.
- Expression maps directly from word problem.

3. Which interaction mode allows them to perform their calculations more rapidly?

Version 2 (RPN) – *again, for users fluent in stack logic*

- Can compute in as little as 5 button presses:
    - 789, Enter, 789, Enter, 148, +, ×
- No switching context or clearing is needed.

4. Which do they find more satisfying?

Version 3

- Why? You feel in control, seeing the full math logic unfold.
- High alignment with standard math phrasing, very little guesswork.

5. Which are unintuitive? In other words, which are more usable?

From most to least intuitive/usable:

Rank	**Version**	**Reasoning**
1	Version 3	Natural math structure; less mental overhead
2	Version 1	Familiar flow, but mentally fragile
3	Version 2	Precise, but cognitively demanding if unfamiliar

Test case (d): If the tax on gasoline is 9 cents per litre, and last week you consumed 135 litres of gasoline, but this week, you only consumed 117 litres of gasoline, then what is the total amount of tax that you saved?

This means calculating as: (135 − 117) × 0.09 = 18 × 0.09 = 1.62

1. Compare the usability of three versions.

Version 1 – INFIX Calculator

- Input Method:
    - First: {'135' '−' '117' '='} (get 18)
    - Then: {'×' '0.09' '='}
- Usability:
    - Simple and intuitive for this two-step calculation.
- Pro: Familiar layout; efficient for short sequences.
- Con: Needs mental tracking or retyping of interim value.

Version 2 – RPN Calculator

- Input Method:
    - {'135' 'Enter' '117' '−'}, (result: 18)
    - {'Enter' '0.09' '×'}
- Usability:
    - Direct and efficient.
- Pro: Stack-based logic is excellent for linear numeric operations like this.
- Con: Still less intuitive for general users due to required understanding of Enter and operand order.

Version 3 – Order of Operations Calculator

- Input Method:
    - {'(' '135' '−' '117' ')' '×' '0.09' '='}
- Usability:

    - Very strong — expression maps directly to natural-language question.

- Pro: Clear grouping via parentheses; accurate expression.
- Con: Slightly longer to enter than Version 1 or 2.

2. Which interaction mode allows the user to perform their calculations with minimal errors?

Version 3 (Order of Operations)

- Why? Allows user to explicitly enter full expression as they understand it from the question.
- Less cognitive burden = fewer mistakes.

3. Which interaction mode allows them to perform their calculations more rapidly?

Version 2 (RPN) – *for stack-fluent users*

- Fewer buttons, fewer steps.
- 135 Enter 117 − Enter 0.09 × = Done.

4. Which do they find more satisfying?

Version 3

- Clarity, accuracy, and confidence all contribute to user satisfaction.
- Seeing (difference) × tax rate reinforces user understanding.

5. Which are unintuitive? In other words, which are more usable?

From most to least intuitive:

Rank	**Version**	**Reasoning**
1	Version 3	Expression matches question phrasing and supports grouping
2	Version 1	Familiar, but more error-prone in multi-step problems
3	Version 2	Precise, but conceptually unintuitive for most users

Test case (e): If two numbers differ by 154, but the ratio of these two numbers is 15, then what is the smaller of the two numbers?

This means calculating as (let the smaller number be x. Then the larger is 15x.):

$15x - x = 154 \rightarrow 14x = 154 \rightarrow x = 154 / 14 = 11$

1. Compare the usability of three versions.

Version 1 – INFIX Calculator

- Input Method:
  - {'154' '/' '14' '='}
- Usability:
  - Excellent for this single operation once the user has worked out the formula.
- Pro: Fast and familiar.
- Con: User must deduce 14 from 15x − x themselves — calculator does no symbolic assistance.

Version 2 – RPN Calculator

- Input Method:
  - {'154' 'Enter' '14' '/'}
- Usability:
  - Very efficient if user understands order.
- Pro: Fast, clean input for division.
- Con: Requires RPN understanding; user still has to derive the formula mentally.

Version 3 – Order of Operations Calculator

- Input Method:
  - {'1' '5' '−' '1' '='} (to get 14), then {'1' '5' '4' '/' '1' '4' '='}
  - Or directly: {'1' '5' '4' '/' '1' '4' '='}
- Usability:
  - Strong. Allows user to enter the exact math as discovered.
- Pro: Can break it into steps or do all at once.

- Con: May require two sub-expressions (if user doesn't pre-compute 15 − 1 = 14 mentally).

2. Which interaction mode allows the user to perform their calculations with minimal errors?

Version 3 (Order of Operations)

- Allows breaking down steps if needed (e.g., first compute 15 − 1, then divide).
- Supports clear entry of logical sequence.

3. Which interaction mode allows them to perform their calculations more rapidly?

Version 2 (RPN) – for experienced users

- 154 Enter 14 / → done
- Fastest sequence once you know what to compute.

4. Which do they find more satisfying?

Version 3

- Gives room to enter the expression clearly, see intermediate steps.
- More confidence when replicating the logic of a word problem.

5. Which are unintuitive? In other words, which are more usable?

From most intuitive/usable to least:

Rank	**Version**	**Reasoning**
1	Version 3	Allows partial or full step-by-step calculation and matches logic
2	Version 1	Familiar but rigid; minimal assistance
3	Version 2	Still least intuitive due to operand-first format