

Video submission – 40%

Together with the code, you should submit a video screen-cast (**through Gradescope**) that explains your implementation. The video will claim points for each expression in your submission as follows by showing its implementation and a sample program that uses the expression. The sample program should make it clear that an expression is correctly implemented (e.g., conditional execution only executes one side).

Besides claiming points for implemented expressions, your video submission should also describe:

- How do you represent expressions? 10%
- Do you implement lexical or dynamic scope? 10%
- How hard/easy is the language you choose versus Java? 10%
- Video is within the time limit. 10%

The maximum length for the video is 10 minutes, instructors will stop watching at the 10 minute mark (nothing past that point in the video will be graded). This video should be a screencast of your IDE open on the code submitted, and you should highlight the code. Note that longer videos are not better videos, and you should record a video as short as needed to show all the expressions and answer the questions above.

You can record such a video without installing any software by using the following website:

<https://screenapp.io/#/>

Language Features

Inheritance and reflection

To use the solution presented in class directly in another language, that language needs to have the following features: **Inheritance** (to extend `Expression`), **reflection** (to get the name of each expression class), and **switching on strings** (to evaluate each expression based on its name).

Enums and arrays

A simpler implementation can use **enums** and **arrays**. You can **represent each expression as an array**. The **first element of this array is an enum** that tells you what the type of the expression is. The **rest of the elements are the data** that you need to evaluate that expression. Note that you can have **nested expressions**.

For instance, you can start by translating the code below to the language you choose:

```
enum Expression { INT_CONST, BIN_OP }
enum Operation { PLUS, MINUS, TIMES, DIV }

Value eval(Object[] c) {
    Expression e = (Expression) c[0];
    switch (e) {
        case INT_CONST: {
            int value = (Integer) c[1];
            return new IntValue(value);
        }
        case BIN_OP: {
            Operation op = (Operation) c[1];
            Value left = eval(c[2]);
            Value right = eval(c[3]);
            switch (op) {
                case PLUS: return new IntValue(left+right);
                ...
            }
        }
    }
}

// Sample program
Object[] p = new Object[]{
    BIN_OP,
    PLUS,
    new Object[]{ INT_CONST, 400 },
    new Object[]{ INT_CONST, 74 }
};

// Should get an IntValue with 474
eval(p);
```

Tagged unions and Pattern-Matching

Choosing a language that supports tagged unions and pattern-matching results in the most compact submission. These are sophisticated language features that minimize the boilerplate code needed for your submission. The code below shows an example in F#. This code has a simple error that you have to fix by finishing the implementation of `BinOp` for all the operations.

```
type Operator = PLUS | MINUS | TIMES | DIV

type Expression =
    | IntConstant of int32
    | BinOp of Operator * Expression * Expression

type Value =
    | IntValue of int32

let rec eval c =
    match c with
    | IntConstant(value) -> (IntValue value)
    | BinOp(op, left, right) ->
        let (IntValue l) = eval left
        let (IntValue r) = eval right
        match op with
        | MINUS -> IntValue (l - r)
        // Finish implementing BinOp here

let p = BinOp(
    PLUS,
    IntConstant(400),
    IntConstant(74))

printfn "Value: %A" (eval p)
```

Memory management

You are **discouraged from using** a language with manual memory management for this assignment, such as **C** or **C++**. Choosing such a language will make your implementation much more complicated than it needs to be.

Sample Programs

This section shows some sample programs (in pseudo-code) that you can translate to your interpreter to show different features. Note that each program shows many features, and you can show a single program in your submission video (the highest possible number that your submission supports).

Program name	Features	Program	Result
P1	<ul style="list-style-type: none">• Constants	474	474
P2	<ul style="list-style-type: none">• Constants• Binary operation	$(400 + 74) / 3$	158
P3	<ul style="list-style-type: none">• Constants• Binary operation• Comparison	$((400 + 74) / 3) == 158$	True
P4	<ul style="list-style-type: none">• Constants• Binary operation• Comparison• Conditional execution	<pre>if (((400 + 74) / 3) == 158) then 474 else 474/0</pre>	474, no divide by zero error
P5	<ul style="list-style-type: none">• Constants• Binary operation• Comparison• Conditional execution• Variables	<pre>let bot = 3 in (let bot = 2 in bot) + (if (bot == 0) then 474/0 else (400+74)/bot)</pre>	160 or 239, no divide by zero error
P6	<ul style="list-style-type: none">• Constants• Binary operation• Comparison• Conditional execution• Variables• Functions	<pre>function f(top,bot) : if (bot == 0) then 0 else top/bot let bot = 3 in (let bot = 2 in bot) + (f(top,bot) + f(top,0))</pre>	160 or 239, no divide by zero error

Due Date

This assignment is due on **December 5 2020 (Saturday)** at **5pm CST**. There is no late policy. There is no opportunity to resubmit this assignment. Together with your submission, you will have to submit (on Gradescope) a screen-cast as described above.

Early Partial Submission Bonus – 10%

This assignment carries a 10% bonus for early submissions. To claim it, you have to submit the implementation of the first two expressions (constants and binary operations), together with a program that uses them, by **Tuesday 24 2020 (Saturday)** at **12h30noon CST** (before class).

Graduate students cannot use the early partial submission bonus.

Piazza Bonus Points – 10%

This assignment has a total of **10% bonus points**, which you can earn by using Piazza as described in the syllabus. Your posts should be public, tagged with the `assignment6` label, and non-anonymous to the instructors to count towards the bonus.

Submission and Grading

This assignment is submitted Gradescope, and graded manually after the submission date.

Errors and Omissions

If you find an error or an omission, please post it on Piazza as soon as you find it.

Hardcoding and Academic Integrity

Any hardcoding will result in a 0% grade. Hardcoding is when you submit code that detects which test is being run, and simply outputs the expected result. For instance, detecting that test 22 is running, and replacing the usual execution of your submission with `System.out.println("expected result")`.

The academic integrity policy described in the syllabus applies to this assignment. You are responsible for writing all the code that you submit. We will use an automatic tool that detects plagiarism on all submitted code, and we will investigate all instances where plagiarism is more than likely.

Please refer to the syllabus for the full academic integrity policy.