CSCC24 2023 Summer – Assignment 3
Due: Saturday July 22, 11:59PM
This assignment is worth 10% of the course grade.

In this assignment, you will implement a recursive descent parser in Haskell.
Code quality is worth 10% of the marks.

Question 1: Hajspy Parser [10 marks]

"A mad scientist was inspired by Python, Javascript, and Haskell. This was what happened."

In this assignment, we will write a parser for a language syntax described below. Here is part of the grammar in EBNF; ambiguities and omissions are resolved in words after. The start symbol is `expr`.

```
1  expr ::= lambda | ifelse
2  lambda ::= var "=>" expr                       -- inspired by JS
3  ifelse ::= arith [ "if" expr "else" ifelse ] -- inspired by Python but "enhanced"
4  arith ::= arith op arith                       -- source of ambiguity
5          | unary
6  unary ::= { "-" } app
7  app ::= app atom | atom                        -- inspired by Haskell, f x y z
8  atom ::= var
9         | "(" expr ")"
10 op ::= "+" | "-" | "*" | "/"
```

The following points resolve ambiguities and omissions:

- `var` is `identifier` from ParserLib, noting that `if` and `else` are reserved words.
- Line 4 is a source of ambiguity and left recursion that you need to fix. You need to implement an unambiguous, terminating parser based on these operator precedence levels, from highest to lowest:

operator	associativity
`* /`	left
`+ -`	left

  Note that line 6 means that the operands of `*` and `/` are `unary`.
- Whitespaces around tokens are possible.

Further explanations and hints:

- Parsing does not check types. The parser accepts what we may think of as ill-typed inputs, as long as they are allowed by the syntax.
- Line 3 is inspired by Python if-else expressions. And as the rule suggests, after an "`else`" you can have yet another if-else expression too! Example and translation to C and Haskell:

Python, Hajspy	`x if b else y if c else z`
C, Java	`b ? x : (c ? y : z)`
Haskell	`if b then x else if c then y else z`

But wait, there's more! Although Python fails to recognize the beauty of it, since the test condition is protected by a pair of matching "`if`" and "`else`" acting as parentheses, there is no ambiguity in allowing any expression, even yet another if-else expression!

Hajspy	`x if b if a else c else y`
C, Java	`(a ? b : c) ? x : y`
Haskell	`if if a then b else c then x else y`

:trollface:

- Line 6 is unary prefix negation. Note that zero or more minus signs are possible, e.g., "`- -x`". (Using `operator` from ParserLib, "`--x`" doesn't qualify. This is normal.)
- Line 7 is another source of left recursion. But it is just trying to say that function application is left-associative, with operands being `atom`.

Implement the parser as `expr` in Parser.hs. The abstract syntax tree to build is defined by `Expr` in Hajspy.hs. My tests will only test `expr` directly or via `run` in testParser.hs. You are free to organize your helper parsers.

Question 2: Type Inference [10 marks]

Show type inference steps for the following expression. The initial environment is empty.

You may omit detailed unification steps, but do show how `unify` calls `unify-intern` for clarity. (Similar to examples in the lecture.)

```
\f m -> case m of Nothing -> Nothing
                  Just x  -> Just (f x)
```

Please create text file q2.txt and hand it in.

End of questions.