THE ENVIRONMENT

Live Coding

Code developed in class.

HOMEWORK

EMAIL

- Send me an email stating "I pinky swear to have a good time for at least part of spring break".

- That’s it. Enjoy your spring break

EXAMS


PAR J. B. NOLIN GEOGRAPHE
A PARIS
1755
MER DU NORD
GRAND Ocean
OCEAN
ETHIOPIEN
AFRIQUE
BARBARIE
NIGRITIE
TERRES AUS

HANDING EXAMS BACK

- I'll ask you to please say your name so I can easily hand the correct exam to you
- Correction / Curve as follows:
  - 90+: no curve available
  - 79-90: Correct for half points, to a maximum score of 90. Email me for details.
  - Below 79: On Thursday morning, come discuss a personal plan to help recover your grade

```
#[derive(Clone)]
#[derive(Debug)]
pub enum Expression {
    Add(Vec<Expression>),
    Multiply(Vec<Expression>),
    Subtract(Vec<Expression>),
    Variable(String),
    Number(f64),
}
```

```
pub struct Environment {
    key: String,
    value: Expression
}
```

```
impl Environment {
    fn value_for_key(self: &Environment, key: &String) → &Expression {
        if &self.key == key {
            &self.value
        } else {
            panic!("key not found in environment");
        }
    }
    fn new() → Environment {
        Environment{ key: String::from(""), value: Expression::Number(0.0) }
    }
}
```

```
pub fn evaluate_addition(add: &Expression,
environment: &Environment) → f64 {
    if let Expression::Add(expressions) = add {
        let iter = expressions.iter();
        iter.fold(0.0, |total, next| total +
evaluate(next, environment))
    } else {
        panic!("Addition not provided")
    }
}
```

```
pub fn evaluate_multiplication(mult:
&Expression, environment: &Environment) → f64 {
    if let Expression::Multiply(expressions) =
mult {
        let iter = expressions.iter();
        iter.fold(1.0, |total, next| total *
evaluate(next, environment))
    } else {
        panic!("Multiply not provided")
    }
}
```

```
pub fn evaluate_subtraction(sub: &Expression,
environment: &Environment) → f64 {
    if let Expression::Subtract(expressions) = sub {
        let mut iter = expressions.iter();
        let first = iter.next().unwrap();
        iter.fold(evaluate(first, environment), |
total, next| total - evaluate(next, environment))
    } else {
        panic!("Subtract not provided")
    }
}
```

```
fn evaluate(expression: &Expression, environment:
&Environment) → f64 {
    match expression {
        Expression::Add(_) ⇒ evaluate_addition(expression,
environment),
        Expression::Multiply(_) ⇒
evaluate_multiplication(expression, environment),
        Expression::Subtract(_) ⇒
evaluate_subtraction(expression, environment),
        Expression::Variable(key) ⇒
evaluate(&environment.value_for_key(key), environment),
        Expression::Number(val) ⇒ *val,
    }
}
```

```
    #[test]
    fn simple_environment() {
        // arrange
        let environment = Environment { key:
String::from("x"), value: Expression::Number(5.0) };
        let values = vec![Expression::Number(2.0),
Expression::Variable(String::from("x"))];
        // act
        let sum = evaluate(&Expression::Add(values),
&environment);
        // assert
        assert_eq!(7.0, sum);
    }
```