1 Introduction to Compilers

Some examples of compilers we've used in this class include

`gcc`	C to binary
`g++`	C++ to binary
`rustc`	Rust to binary
`javac`	Java to JVM bytecode
`ghc`	Haskell to Haskell magic
`tsc`	TypeScript to JavaScript

Essentially, a compiler takes some program and produces an output program, one that is easier for us to run in some environment. In this course, we're going to create a compiler `ucsdc` that takes in `snek` files and will produce `x86_64` binaries.

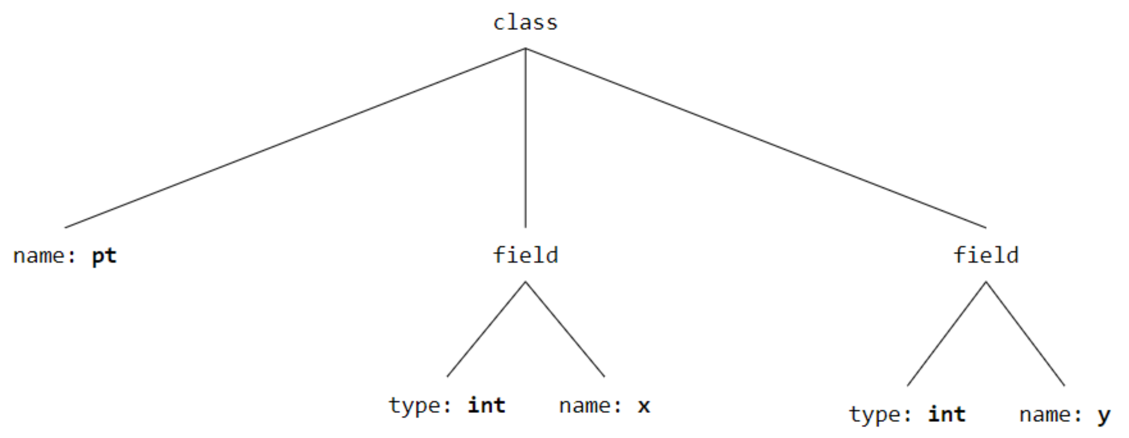
To be more specific, our compiler should do the following: given a `.snek` text file,

- parse[1] the text into an abstract syntax tree (AST),

(Example.) For example, consider the following Java class.

```
class Pt {
    int x;
    int y;
}
```

Its AST representation might look like



- and either
  - generates assembly (or some other output program), or
  - generate an error message.

We are not interested in generating the most optimized assembly, just that it works. Most of the time spent in this course will be on the code generating part (i.e., from AST to assembly). Towards the *end* of the course, we'll work with libraries/runtime.

[1]Known to be very boring.

1.1 Course Syllabus

The course webpage can be found here: https://ucsd-compilers-s23.github.io/index.html.

- Two exams during discussion (week 5, week 9).
- The final exam is skippable. The final exam can be used to improve your scores on the two exams. If you did well on the two exams, you don't need to do the final exam.
- Engagement usually means weekly quiz on Gradescope and lecture handouts. They may help make the difference between an A and A+.
- Graduate students may be asked to do some additional things for the later projects.