

Compiler Construction

~ The Tigrou Project ~

Goals

Introduction to transpilation &
compilation techniques

Define & execute a mini-language
in one week!

Discover & play with (a subset of)
Tiger's language

Helper for choosing the main project of
the semester: Tiger or Spider

Tigrou's Backus–Naur Form (BNF)

```
number ::= [0-9]+
id ::= [a-zA-Z][a-zA-Z_0-9]*

exp ::=  exp "+" exp
      |  exp "-" exp
      |  exp "*" exp
      |  exp "/" exp
      |  "(" exp ")"
      |  number
      |  "let" "var" id ":@" exp
      |  "in" exp "end"
      |  "print" "(" exp ")"
      |  identifier
```

Restrictions

Name reusability

A variable name cannot be reused !

Syntactically correct

Even if `1 + print(2)` is syntactically correct, it will failed to type check

No sequence of instructions

Only one expression is available in a given scope *but it can be simulated using nesting*

Stages

Stage 1:

Scanner, Parser, and AST

Stage 2:

Binder & Type Checker

Stage 3:

GNU C Code generation
(compound statements)

Stage 1: Scanner, Parser & AST

Two difficulties:

- combine Flex/Bison with the AST generation
- pretty-printer

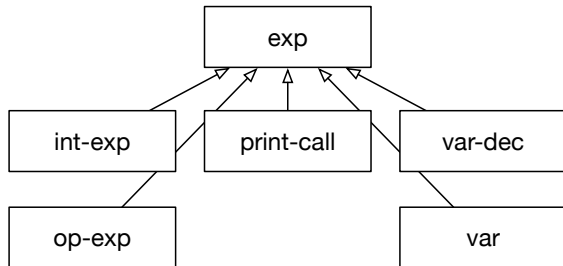
Start from your THL calculator!

Stage 1: Scanner, Parser & AST

Two difficulties:

- combine Flex/Bison with the AST generation
- pretty-printer

Start from your THL calculator!



Stage 2: Binder & Type Checker

Binder

Only check that a variable is not used twice!

TypeChecker

Check for illegal combination in expressions

Technical detail

Use virtual dispatch to walk the AST

Stage 3: Code Generation

Input

```
let var a := 2
in
  print(a + 2)
end
```


Output

```
#include <stdio.h>
int main(void) {
  ({
    int a = 2;
    printf("%d\n", (a + 2));
  })
}
```

Summary



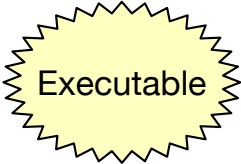
Simple
project



Full front-
end



Transpil.
techniques



Executable