In-Class Exercises

Compilers

Version: 2025-12-26

Download exercises

1. Go to

   https://ligerlabs.org/compilers.html

2. Download the file

   `compilers-1-exercises.zip`

3. Open up a terminal and Unzip the file

```
> unzip compilers-1-exercises.zip
> cd compilers-1-exercises
> ls
```

Task 1 (a)

- Go to `https://godbolt.org` and enter this program:

C

```
int f(int n) {
    return (n+1) * (n+1);
}
```

1. Select the "`mips gcc 15.2.0`" compiler. Can you recognize what the assembly code is doing?
2. Under the "`+`" menu, select "`GCC Tree/RTL`". Select the pass "cfg (tree)". Does the IR make sense?
3. Enter the Compiler Option "`-O`" (optimization). What is the assembly code doing now?

Task 1 (b)

- Go to `https://godbolt.org` and enter this program:

Java

```
class F {
    int f(int n) {
        return (n+1) * (n+1);
    }
}
```

1. Select the "`Java`" language and the "`idk 25.0.1`" compiler. Can you guess what the Java byte code is doing?
2. Under the "`+`" menu, select "`Claude Explain`". Does the LLM help you understand the code?

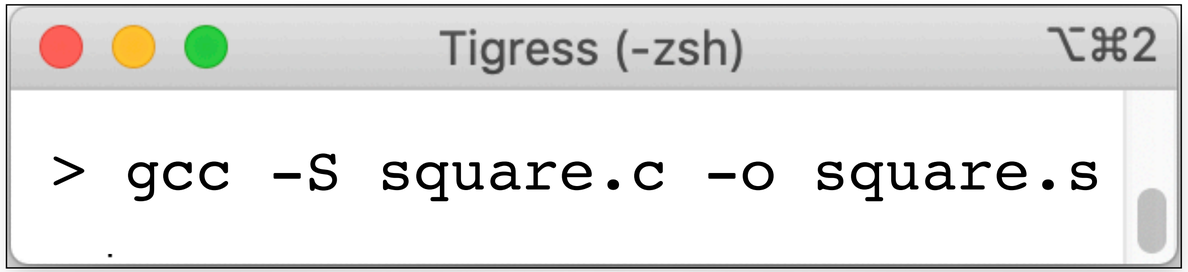
Task 2

- Say that you're writing a compiler for a new language L.
- L has an operator "#" that squares its argument:

L

```
var x : integer = 5;
print #x;
```

- But, what code should you generate for this operator>
- Try this common trick: write a C function that performs this operation, compile it with with gcc, and then examine the generated assembly code:



Task 3 (a)

- Consider the following program:

```
PROCEDURE Foo ();
VAR i : INTEGER;
BEGIN
    i := 1;
    IF i < 2.0 THEN
      i := i + 1;
END
```

1. Given the token definitions on the next page, list the tokens that the lexical analysis would generate.

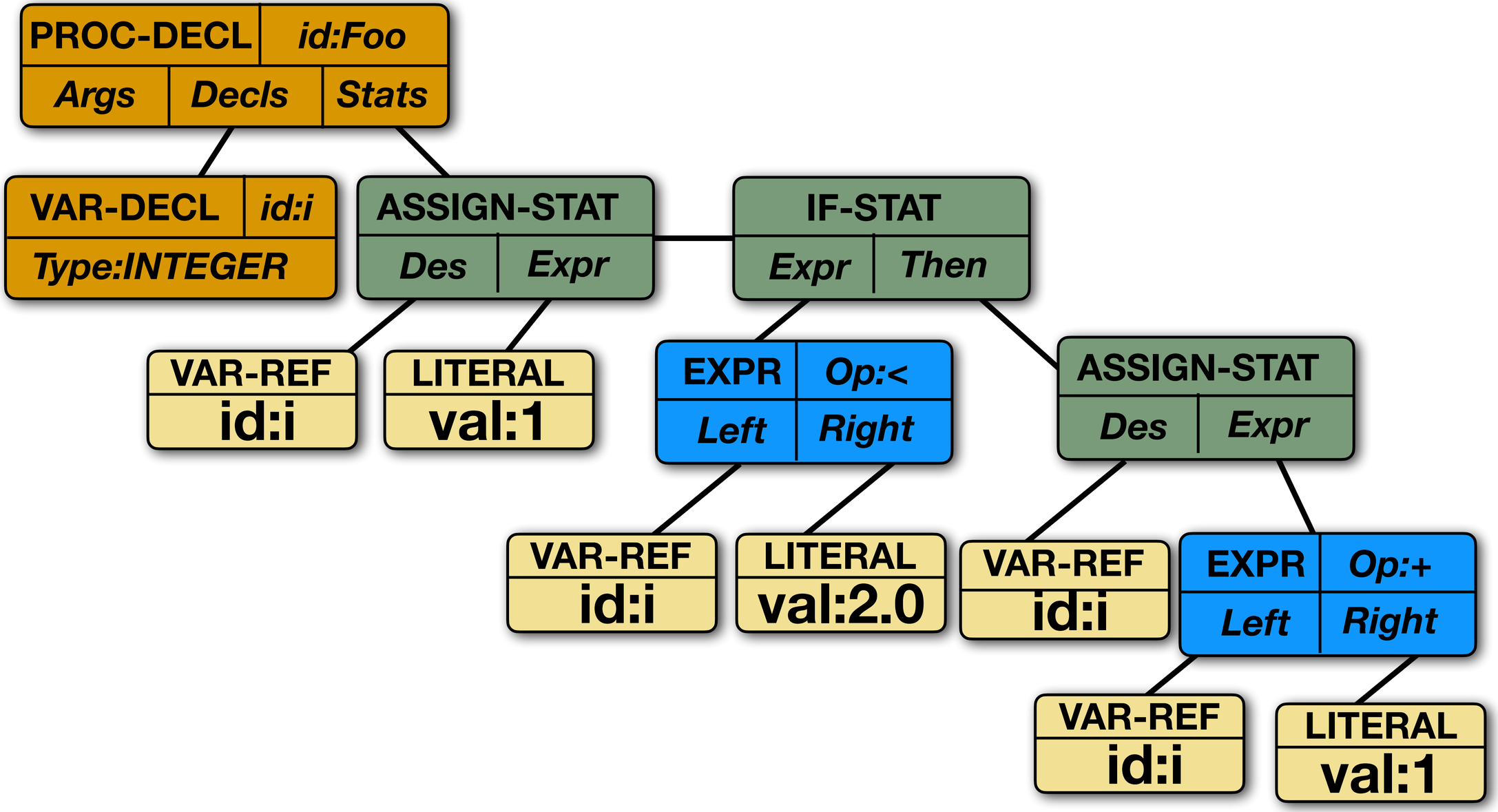
String	Token
"PROCEDURE"	PROCEDURE
identifier	<id,Foo>
"("	LPAR
")"	RPAR
";"	SC
"VAR"	VAR
":"	COLON
"BEGIN"	BEGIN
":="	CEQ

String	Token
integer literal	<int,42>
"IF"	IF
"<"	LT
float literal	<flt,4.2>
"PRINT"	PRINT
"*"	MUL
"+"	PLUS
"THEN"	THEN
"END"	END

Task 3 (b)

- Consider the Abstract Syntax tree on the next page that was generated from the Foo function.

2. Decorate this tree by adding an attribute "`Type`" to each relevant tree node, and set this type to one of "`Int`" or "`Float`".


PROC-DECL
id:Foo
Args
Decls
Stats
VAR-DECL
id:i
Type:INTEGER
ASSIGN-STAT
Des
Expr
IF-STAT
Expr
Then
VAR-REF
id:i
LITERAL
val:1
EXPR
Op:<
Left
Right
ASSIGN-STAT
Des
Expr
VAR-REF
id:i
LITERAL
val:2.0
VAR-REF
id:i
EXPR
Op:+
Left
Right
VAR-REF
id:i
LITERAL
val:1

Task 3 (c)

- Consider the intermediate code instructions on the next page.

3. Generate intermediate code instructions from the abstract syntax tree for the function Foo.

IR	Defintion
ASSIGN A,B	$A \leftarrow B$
BRGE A,B,L	IF ($A \geq B$) THEN continue at instruction L
MUL A,B,C	$A \leftarrow B * C$
ADD A,B,C	$A \leftarrow B + C$
SHL A,B,C	A ← shift B left C steps
PRINT A	Print A and a newline
JUMP L	Continue at instruction L

Task 4

- Search the internet for
  - job postings for compiler engineers
  - salary estimates for compiler engineers
  - PhD student research assistantships with an emphasis on compilers
- Report your findings to the class.

- Some relevant links:
  - `https://glassdoor.com`
  - `https://indeed.com`

collberg.cs.arizona.edu

ligerlabs.org

Supported by

NSF SATC/TTP-1525820

SATC/EDU-2029632

Christian Collberg

University of Arizona