6.110 Computer Language Engineering

Recitation 3: Project phase 2

February 23, 2024

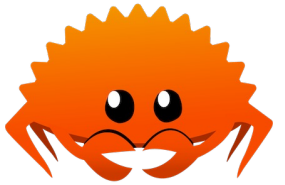
Weekly updates ←

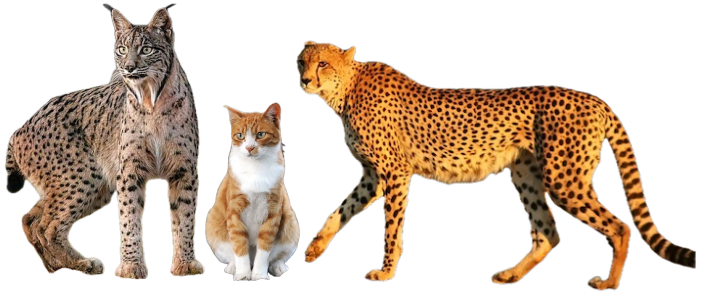
Phase 2 info

Phase 2 demo

Animals

(not to scale)





Wrapping up phase 1…

- Project phase 1 is due **today 11:59PM!!!**
  - We have OH from 2-7pm to help you
  - LLM questionnaire due Monday, February 26
- You are allowed to share your phase 1 code with potential teammates after the deadline.

Coming up soon...

- Team preference form due **Wednesday, February 28**
- Project phase 2 has been released, due **Friday, March 8** (in two weeks)
- Miniquiz 3 and Weekly Check-in 4 released, due Thursday, February 29.

Mon 2/26	Tue 2/27	Wed 2/28	Thu 2/29	Fri 3/1
No lectures next week Lectures will resume Mon 3/4				**Recitation** x86 Assembly
OH 4-6pm		**Re-lecture** for Week 3 lectures	**OH** 4-6pm	**OH** 2-7pm
		Due: Team preference form	**Due:** Mini-quiz, weekly check-in	

Weekly updates

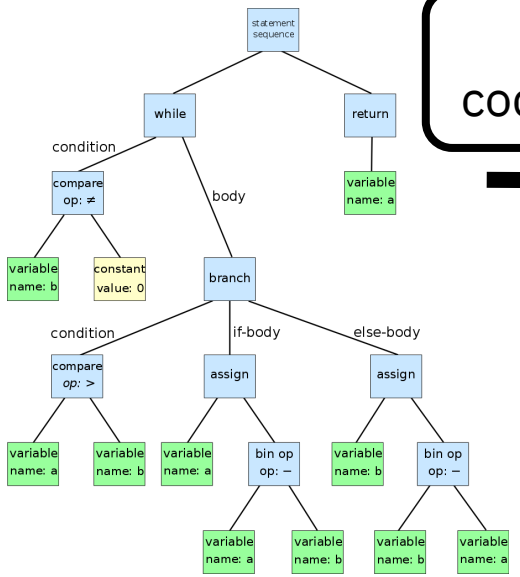
Phase 2 info ←

Phase 2 demo

```
import printf;

void main() {
…
```

Decaf source file



Internal representation

Phase 3
code generation

```
push %rbp
mov  %rsp, %rbp
…
```

x86-64 assembly

Phase 1. Does it have the right structure? (syntax)

Phase 2. Does it make sense? (semantics)

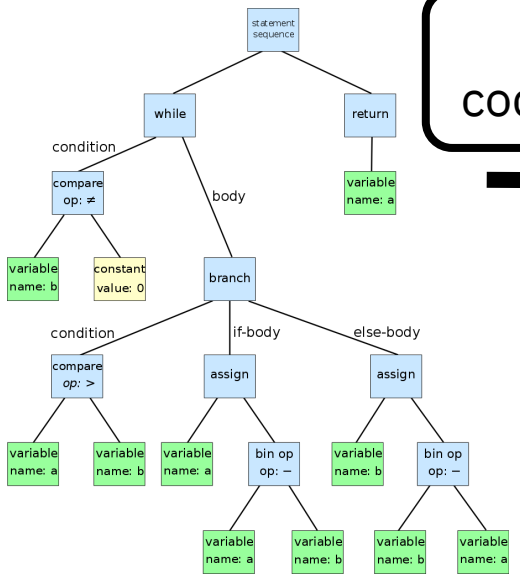
Phase 4. What can we learn about the program? (dataflow analysis)

Phase 5. How can we make the output code faster?

```
import printf;

void main() {
…
```

Decaf source file



Internal representation

Phase 3
code generation

```
push %rbp
mov  %rsp, %rbp
…
```

x86-64 assembly

~~**Phase 1.** Does it have the right structure? (syntax)~~

Phase 2. Does it make sense? (semantics)

Phase 4. What can we learn about the program? (dataflow analysis)

Phase 5. How can we make the output code faster?

Phase 2 overview

- Group project, in teams of 3-4. You'll keep working with the same group for all the remaining phases.
- **Goal:** have a working compiler frontend that can determine whether each input Decaf code is *semantically* valid or not.

Team formation process

- Submit team preference form on Gradescope **as a group with your preferred teammates.**
- We'll match you up with other students/groups to form groups of 3-4.
  - Matching will be based on preferred language.
  - You can also opt out of the matching process, but note that there will be a lot of work per person for smaller groups.

Specifications

- When running `./run.sh <filename> -t inter` on **a semantically valid input file:**
  - Exits with return code 0 (OK)
  - Produce no output
- You can decide how you want to implement your IR and semantic checker.

Specifications

- When running `./run.sh <filename> -t inter` on **a semantically invalid input file:**
  - Exit with nonzero return code (i.e. error)
  - Outputs reasonable error messages to stderr. (should include line/column number and the identifier that caused the error)
- We'll manually check your error messages.
  - As long as they are reasonable, you'll get full credit.

Submission and grading

- Phase 2 is worth **5%** of the overall grade, due **Friday, March 8.**
- Three items to be submitted on Gradescope
  - Code submission
    - Autograded tests: **2.5%**
    - Error messages: **1%**
  - Report: **1.5%**
    - Overview of approach, team status report, LLM

Getting started

- Once teams have been assigned, we will create team repositories for you.
  - We'll initially use a placeholder name for your team repository.
  - If you'd like to name your team, please let us know and we'll change your repository name.
- You are allowed to use your team members' phase 1 code.

Suggested approach

1. Convert parse tree or AST to a high-level IR by traversing AST nodes and constructing **symbol tables**.
2. Once you've finished constructing the IR, perform **semantic checks** by traversing your IR.

Symbol tables

- Stores relevant information about each identifier

identifier →	*descriptor*
`x`	local variable id 1, type int
`f`	method id 3, type bool → int

Scope

global scope

```
import printf;
int x = 0;
```

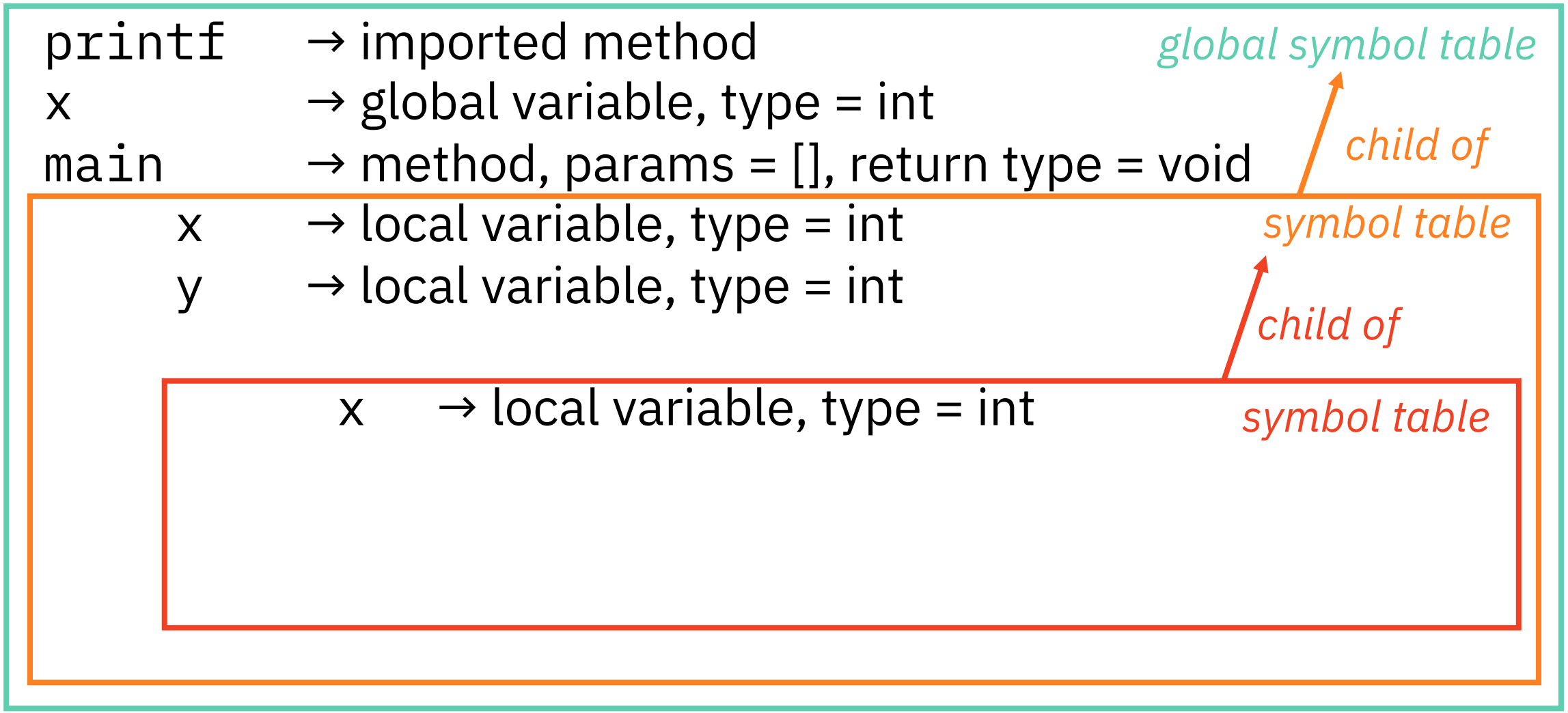
method scope

```
void main() {
    int x = 1, y = 2;
    if (x > 0)
```

block scope

```
    {
        int x = 3;
        printf("%d %d", x + y);
    }
}
```

Symbol tables



Scope

global scope

```
import printf;
int x = 0;
```

method scope

```
void main() {
    int x = 1, y = 2;
    if (x > 0)
    {
        int x = 3;
        printf("%d %d", x + y);
    }
}
```

block scope

Symbol tables: summary

- One symbol table per scope
  - Each symbol table links to symbol table of parent scope
- First search for identifier in current scope
  - If not found, go to parent symbol table
  - If not found in any table, *semantic error!*

Semantic checks

- Here are some types of semantic rules

Name issues

```
void main() {
    int x, x; // R1: x defined twice
                 in same scope
    y = 0;    // R2: y not defined
}
```

Type errors: expressions

```
x[true]      // R14 : array index
                     must be int
4 + true     // R17 : <arith_op> takes
                     two ints
false == 1   // R18 : <eq_op> takes
                     same type
4 && 5       // R19 : <cond_op> takes
                     two bools
```

Type errors: assignments

```
int i, arr[];
bool b;
arr = 0;    // R12 : cannot assign
                 to array
i = true;   // R20 : assignment type
                 must match
b++;        // R21 : can only
                 increment ints
```

Miscellaneous rules

```
int arr[-1]; // R6: array size must
                    be positive


9223372036854775808
// R25: int must be in bounds
```

Semantic checks

- Here are some types of semantic rules
  - Name issues
  - Type errors
  - Miscellaneous rules
- For the full list of rules, **check the Decaf spec!**

Weekly updates

Phase 2 info

Phase 2 demo ←

IR and semantic checking demo

Code available at:

https://github.com/6110-sp24/recitation3