Check-in

Review: Semantic Analysis

```
 1. int a;
 2. bool f;
 3. int m(int arg){
 4.     int b;
 5.     return arg + 1;
 6. }
 7.
 8. int g(){
 9.     int c;
10.     int d;
11.     if (a){
12.         int d;
13.         int f;
14.         int g;
15.     }
16. }
```

Show the symbol table
After line 12 but before line 13

University of Kansas | Drew Davidson

EECS 665 COMPILER CONSTRUCTION

Type systems

Last Time
Semantic Analysis

Name Analysis

- Enforcing scope

Symbol Table

- What it is
- What it does

You should know

Name analysis

- What it is
- What it does
- How it works

Semantics

Lecture Outline

Type Systems

Discuss Type Systems

- What they are
- Why we use them

Type Specification

- Formally communicating type systems

Our type system (for the project)

Semantics

Specification vs Implementation

Discussing Type Systems

A big idea in compilers

- Thinking at different layers of abstraction
- Types are a nice instance
  (so were syntax and tokenization)
- Today: Specification
- Next time: Implementation

Recall: Aim of Semantic Analysis

Type Systems – Rationale

Deduce what the programmer meant

- Philosophy: don’t let bugs get by
- Give programmer means to express intent

Types as Hints From Programmer

Type Systems – Rationale

Types Communicate programmer intention

- Compiler can choose the appropriate operation
- Compiler can tell if the operations are sensible

What we Mean by "Type"

Short for "data type"

- Classification for various kinds of data
- A set of possible values which a variable can possess

May imply representation

(perhaps in memory)

- int32

Type Systems: The Context for Types

Type Systems

Type System: lists types and describes how they may be used

- What operations that can be done on member values
- How type system may be extended

Components of a Type System

Type Systems

- Base types and means of building aggregate types
  - int, bool, void, class, function, struct, pointer, reference
- A means of determining if types are compatible
  - Can disparate types be combined? How?
- Rules for inferring the type of an expression

Type Rules

Type Systems

- For every operator (including assignment)…
  - What types can the operand have?
  - What type is the result?
- Example:

```
double a;
int b;
a = b;
b = a;
```

`a = b;` Legal in Java, C++

`b = a;` Legal in C++, Illegal in Java

c = a = b;

Type Conversion

Type Systems

Defn: Using One Type as a Different Type

- May require explicit acknowledgement by user (e.g. casting)

Type Coercion
Type Systems

Defn: *Implicit* cast from one data type to another

- For example:

`int` to `unsigned int`

```
#include <stdio.h>
int main(){
		unsigned int a = 1;
		int b = -1;
		if (a * b < 0){
				printf("NEG");
		} else {
				printf("NON-NEG");
		}
}
```

Type Promotion

Type Systems

A narrow form of coercion

- When destination type can represent the source type without loss of precision
- float to double (ok)
- double to float (not ok)

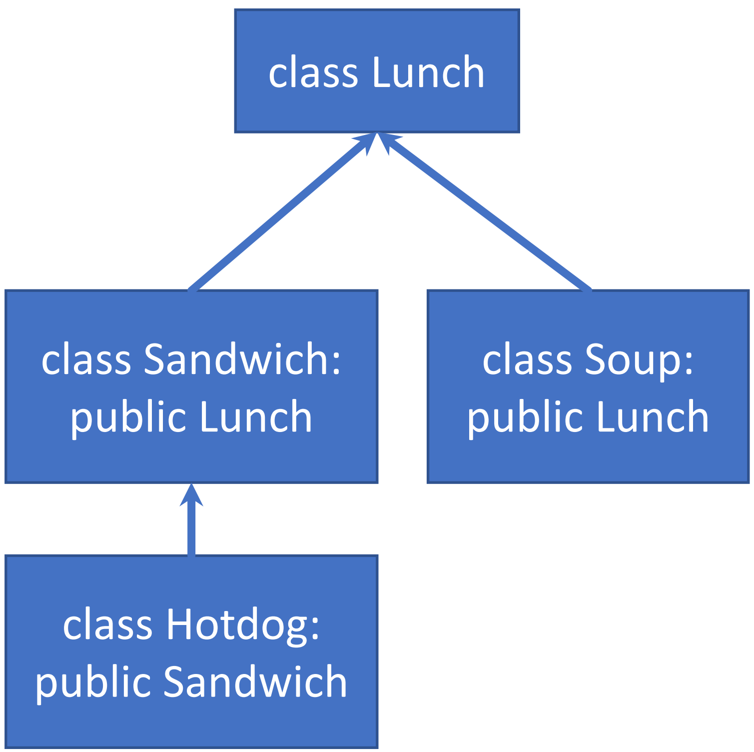
A promotion ceremony

Subtyping

Type Systems

When a more narrow type can be used in place of a another

- Explicit inheritance / class hierarchy



Duck Typing

Type Systems

Defn: Type is defined by the methods and properties

"If it walks like a duck and talks like a duck, it's a duck"

Duck Typing: Example


```
class Duck:
      def quack(): print(“quack”)
class Rando:
      def quack(): print(“QUACK”)

def processDuck(Duck d) { … }
Rando r = new Rando();
processDuck(r);
```

Duck Punching

Type Systems

Defn: Type defined by the methods/properties *at time of use*

"If it walks like a duck but isn't giving you the noise you want, punch it until it quacks. Now it's a duck"

Brief Aside: Duck Punching

Type Systems

Also sometimes called gorilla typing

guerilla (as in covert/secret)

gorilla (sounds like guerilla)

Duck Punching: Example

Type Systems

```
class Duck:
      def quack(): print("quack")
class MechaBird:
      def squak(): print("101001…")

def processDuck(Duck d) { … }
MechaBird m = new MechaBird();
m.quack = m.squak;
processDuck(m);
```

Let's Talk about The Type System Used in the Projects

Type Checking

Our Type System: Fundamentals

Type Checking

- Primitive Types
  - int, bool, short, string, void
- Aggregate types
  - pointers, functions
- Coercion
  - Bool cannot be used as an int (nor vice-versa)
  - Short can be promoted to int
  - Int cannot be demoted to short

Our Type Rules

Our Type System

- Arithmetic operators must have **int** or **short** operands
- Equality operators **==** and **!=**
  - Operands must have same type
    - CANNOT be applied to functions
    - CAN be applied to function results
- Other relational operators must have **int** or **short** operands
- Logical operators must have **bool** operands

Type Errors II

Our Type System

- Assignment operator
  - Must have operands of the same type
  - Can’t be applied to functions
    - Functions (but CAN be applied to function results)
- For sending data to the console
  - x must be an rval (usable on RHS of an assignment)
- For reading data from the console
  - x must be an lval (usable on LHS of an assignment)
- Condition of **if** and condition of **while** must be boolean

Type Errors III

Our Type System

- Invoking (calling) something that's not a function
- Invoking a function with
  - Wrong number of args
  - Wrong type of args
- Returning a value from a void function
- Not returning a value in a non-void function
- Returning a wrong type of value in a non-void function

Summary
Type Systems

- Invoking (calling) something that's not a function
- Invoking a function with
  - Wrong number of args
  - Wrong type of args
- Returning a value from a void function
- Not returning a value in a non-void function
- Returning a wrong type of value in a non-void function

Upcoming Project: P3

Type Systems

Implement name analysis

Formalizing Type Systems

Detour: Ungraded Material

Representing Type Systems

Formal Type Systems

Particular formalism: Judgements + rules

Judgements:

$$\Gamma \vdash \mathfrak{J}$$

$\mathfrak{J}$ is an assertion;
Free variables in $\mathfrak{J}$ are declared in Γ

Rules:

(rule name)

$$\frac{\Gamma_1 \vdash \mathfrak{J}_1 \quad \ldots \quad \Gamma_n \vdash \mathfrak{J}_n}{\Gamma \vdash \mathfrak{J}} \quad \text{(annotations)}$$

Judgements

Formal Type Systems

$$\Gamma \vdash \mathfrak{J}$$

$\mathfrak{J}$ is an assertion;
Free variables in $\mathfrak{J}$ are declared in Γ

Example Judgements

$$\Gamma \vdash \lozenge$$

$$\Gamma \vdash M : A$$

$$\emptyset \vdash true : bool$$

$$\emptyset, x{:}\, int \vdash x + 1 : int$$

Rules

Formal Type Systems

DETOUR

(rule name)

$$\frac{\Gamma_1 \vdash \mathfrak{J}_1 \quad \ldots \quad \Gamma_n \vdash \mathfrak{J}_n}{\Gamma \vdash \mathfrak{J}}$$

Example type rules

(Env ∅)

$$\frac{}{\emptyset \vdash \Diamond}$$

(Val 1)

$$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash 1:\text{int}}$$

(Val +)

$$\frac{\Gamma \vdash M : \text{int} \qquad \Gamma \vdash N : \text{int}}{\Gamma \vdash M+N:\text{int}}$$

Proof Trees

Formal Type Systems

$$\dfrac{\dfrac{\overline{\emptyset \vdash \Diamond}\ \text{By (Env } \emptyset)}{\emptyset \vdash 1{:}\text{int}}\ \text{By (Val 1)} \qquad \dfrac{\overline{\emptyset \vdash \Diamond}\ \text{By (Env } \emptyset)}{\emptyset \vdash 1{:}\text{int}}\ \text{By (Val 1)}}{\emptyset \vdash 1+1{:}\text{int}}\ \text{By (Val +)}$$

Example type rules

(Env Ø)

$$\dfrac{}{\emptyset \vdash \Diamond}$$

(Val 1)

$$\dfrac{\Gamma \vdash \Diamond}{\Gamma \vdash 1{:}\text{int}}$$

(Val +)

$$\dfrac{\Gamma \vdash M : \text{int} \qquad \Gamma \vdash N : \text{int}}{\Gamma \vdash M+N{:}\text{int}}$$

Well-Typedness

Formal Type Systems

DETOUR

A way to express that the program can be correctly typed

Basic Scheme

- State rules for language constructs
- Well-typed if it can be placed at root of a complete proof tree

Hypothetical proof tree

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \Diamond}\ \text{By (Env } \emptyset)}{\emptyset \vdash 1{:}\text{int}}\ \text{By (Val 1)} \qquad \dfrac{\dfrac{}{\emptyset \vdash \Diamond}\ \text{By (Env } \emptyset)}{\emptyset \vdash 1{:}\text{int}}\ \text{By (Val 1)}}{\emptyset \vdash 1+1{:}\text{int}}\ \text{By (Val +)}$$

Example Type Rules

Formal Type Systems

DETOUR

(val arr-len)

$$\frac{\Gamma \vdash E : T[\,]}{\Gamma \vdash E.\text{length} : \text{int}}$$

(val arr-elt)

$$\frac{\Gamma \vdash E_0 : T[\,] \qquad \Gamma \vdash E_1 : \text{int}}{\Gamma \vdash E_0[E_1] : T}$$

(val arr-alloc)

$$\frac{\Gamma \vdash E : \text{int}}{\Gamma \vdash \text{new } T[E] : T[\,]}$$

Example Type Rules

Formal Type Systems

(val stmt)

$$\frac{\Gamma \vdash E : T}{\Gamma \vdash S : \text{void}}$$

Where statement S contains only expression E

Example Type Rules

Formal Type Systems

(val sequence)

$$\frac{\Gamma \vdash S_1 : \mathrm{T}_1 \qquad \Gamma \vdash (\,S_2; \ldots; S_\mathrm{n}):\ T_\mathrm{n}}{\Gamma \vdash (\,S_1; S_2; \ldots; S_\mathrm{n}):\ T_\mathrm{n}}$$

Example Type Rules

Formal Type Systems

(val declaration)

$$\frac{\Gamma \vdash E : T \qquad\qquad \Gamma, \mathrm{id}: T \ \vdash (S_2 \,;\ \ldots ; Sn) : T'}{\Gamma \vdash (\mathrm{id} : T = E \,; S_2 \,;\ \ldots ; Sn) : T'}$$

Example Type Rules

Formal Type Systems

(val fn-call)

$$\frac{\Gamma \vdash E_1 : T_1 \times \cdots \times \mathrm{E_n} : \mathrm{Tn} \rightarrow Tr \qquad \Gamma \vdash E_i : Ti \; (i \in 1..n)}{\Gamma \vdash \mathrm{E(E_1\,,\ \ldots, En) : Tr}}$$

Formal Type Systems

End Detour: Done with Ungraded Material