CS 536 Announcements for Thursday, April 7, 2022

Last Time
- name analysis
- static vs dynamic scoping
- scoping issues to consider
- name analysis in minim
  - scoping rules
  - symbol table
  - handling structs

Today
- type checking
- type-system concepts
- type-system vocabulary
- minim
  - type rules
  - how to apply type rules

Next Time
- runtime environments

Name analysis and structs

Defining a struct

Declaration of a variable of a struct type

Accessing fields of a struct

- Compiler needs to
  - for each field: determine type, size, and offset with the structure
  - determine overall size of structure
  - verify declarations and uses of something of a struct type are valid

- Idea: each struct type definition contains its own symbol table for its field declarations
  - associated with the main symbol table entry for that struct's name
Handling Classes

Similar to handling aggregate data structures
• also need to be able to search the class hierarchy

Idea:
Symbol table for each class with two nesting hierarchies

1)
2)

To resolve a name

What is a type?

Short for data type
• classification identifying kinds of data
• a set of possible values that a variable can possess
• operations that can be done on member values
• a representation (perhaps in memory)

Type intuition – is the following allowed?
```c
int a = 0;
int *pointer = &a;
float fraction = 1.2;
a = pointer + fraction;
```
Components of a type system

base types (built-in/primitive)

rules for constructing types

means of determining if types are compatible or equivalent

rules for inferring the type of an expression

Type rules of a language specify
- what types the operands of an operator must be

Example:
```c
double a;
int b;
a = b;
b = a;
```

- what type the result of an operator is

Type coercion
  - implicit cast from one data type to another

  - type promotion
Type checking: *when* do we check?

static typing

dynamic typing

combination of the two

Static vs dynamic trade-offs

- static

- dynamic
Type checking: what do we check?

strong vs weak typing
- degree to which type checks are performed
- degree to which type errors are allowed to happened at runtime

General principles
- statically type → 
- more implicit casting allowed → 
- fewer checks performed at runtime →

Example

Type safety
- All successful operations must be allowed by the type system
- Java is explicitly designed to be type safe

- C is not

- C++ is a little better
Type system of minim

minim's type system
- primitive types
- type constructors
- coercion

Type errors in minim

Operators applied to operands of wrong type
- arithmetic operators
- logical operators
- equality operators
- other relational operators
- assignment operator

Expressions that, because of context, must be a particular type but are not
- expressions that must be boolean (in minim)
- reading
- writing
Type errors in minim (cont.)

Related to function calls
- invoking (i.e., calling) something that is not a function
- invoking a function with
  - wrong number of arguments
  - wrong types of arguments
- returning a value from a void function
- not returning a value from a non-void function
- returning wrong type of value in a non-void function

Type checking
Recursively walks the AST to
- determine the type of each expression and sub-expression using the type rules of the language
- find type errors

Add a `typeCheck` method to AST nodes

Type checking: binary operator

Type "checking": literal

Type checking: IdNode
Type checking (cont.)

Type checking: others

- call to function \( f \)
  - get type of each actual parameter of \( f \)
  - match against type of corresponding formal parameter of \( f \)
  - pass \( f \)'s return type up the tree

- statement \( s \)
  - type check constituents of \( s \)

Type checking: errors

Goals:
- report as many distinct errors as possible
- don't report same error multiple times – avoid error cascading

Introduce internal error type
- when type incompatibility is discovered
  - report the error
  - pass error up the tree
- when a type check gets error as an operand
  - don't (re)report an error
  - pass error up the tree

Example:
```plaintext
int a;
bool b;
a = true + 1 + 2 + b;
b = 2;
```