Midterms

Grades posted
Tests will be returned tomorrow in class

Mean: 90.2
Median: 94.5
Std Dev: 11.75

Bring questions to my office hours after class tomorrow in CSE 218
Project Part III Overview

Due: Friday, March 1
● Build symbol tables
● Calculate type information
● Perform error checking
● Print symbol tables

Lots of hints in the writeup:
● Use (many) visitors
● Use helper methods
● ...
Tests

Some generally good advice:
● Write your tests first
● Keep unit tests small
● Only test one thing per test

Some options for running tests:
● Write JUnit tests
● Write a script to run your tests (possibly via ant) and check exit codes
Test Driven Development

For each semantic error you need to catch:
- Write a minijava program containing that error
- Check that your compiler fails to catch the error
- Update the compiler to detect the error
- Check that your compiler catches the error

Advantages:
- Tests are written first
- Tests are small
- Test coverage is good (regression testing)
MiniJava Symbol Tables

Global Table: Map class names to class tables

Class Tables: Map methods and fields to type information, storage locations, etc.

Method Tables: Map variables and parameters to type information, storage locations, etc.

You will probably want to persist tables over multiple compiler passes
Types in Minijava

Types are not AST nodes!
● Create your own "type" class hierarchy
● Use singletons for base types (int, ...)

Use helpers: assignmentCompatible(Type, Type)

See lecture slides for more hints

Real Java has coercions, casting, ...
x86 Highlights

label: op dst, src ;comment

up to one memory address per instruction

caller saved: eax, ecx, edx
callee saved: ebx, esi, edi
ebp (stack frame base)
esp (last occupied, aligned stack entry)
x86 Highlights

mov eax, 17  mov eax, [ebp+8]
mov eax, ecx  mov [ebp-12], eax

[basereg + indexreg * \{2,4,8\} + constant]

binary ops: mov, add, sub, imul, and, or, xor
unary ops: inc, dec, neg, not
x86 Highlights

lea dst, src;  dst <- address of src
src should be a memory address computation
The & operator in C

jmp dst

cmp dst,src;  sets eflags
je, jne, jz, jnz, jg, jng, jg, jnge, jl, jnl, jle, jnle
x86 Highlights

push src;  esp <- esp - 4; memory[esp] <- src
pop dst;  dst <- memory[esp]; esp <- esp + 4
call label;  esp <- esp - 4; memory[esp] <- eip
ret;  eip <- memory[esp]; esp <- esp + 4
leave;  mov esp,ebp; pop ebp
x86 Highlights

Function Caller:
- Push args (from right to left)
- Execute call
- Pop args

Function Callee:
- Save/spill registers and allocate stack frame
- Execute function (leave result in eax)
- Restore registers and pop stack frame
- Return
Code Generation

Generate code for AST using a visitor

- Visit children as necessary

For simple binary operations:

- Visit left child and save result
- Visit right child
- Apply operation to results

Tip: Keep trees in mind
Code Shape: Simple Operations

Local variable access:

```assembly
mov eax, [ebp+16]
```

Location of variable stored in symbol table
Offsets are stored for objects
Code Shape: While Statements

while (cond) stmt

l1:  <compute cond>
    j_false l2
    <compute stmt>
    jmp l1

l2:
Code Shape: If-Else Statements

if (cond) stmt1 else stmt2

<compute cond>
j_false l3
<compute stmt1>
jmp l4
l3: <compute stmt2>
l4:
Code Shape: Conditionals

Conditionals are annoying in x86:

- There is no j_false operation
- Use cmp and conditional jumps instead
  - Don't always want the result of boolean operations left in a register
  - Requires special conditional processing
- You can still have boolean variables, so you still need the regular processing (leaving results in registers)
Code Shape: Switch Statements

switch (exp) { case 10: x = 11; case 12: x = 13; }

Could generate:
<evaluate exp into eax>
<jmp default if no table entry exists for value in eax>
mov eax, switch_table[eax*4-40]
jmp eax
L10: <code for x = 11>
L12: <code for x = 12>

What does switch_table need to look like?
Code Shape: Switch Statements

switch (exp) { case 10: x = 11; case 12: x = 13; }

...  
    mov eax, switch_table[eax*4-40]
...  

.data switch_table
    dd L10
    dd L_default
    dd L12
    jmp eax
Code Shape: Arrays

\[ \text{exp1[exp2]} \]

<evaluate exp1 into eax>
<evaluate exp2 into edx>
mv eax, [eax+4*edx]

Multidimensional arrays are more complicated
- Don't exist in Java
More Complex Generation for OO Code

Coming up in next lectures/sections
Questions?