CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

http://www.cse.buffalo.edu/faculty/alphonce/SP17/CSE443/index.php
https://piazza.com/class/iybn4ndqal3ei
Workshop Wednesdays

- Post topics for WW @ 4/5 to Piazza by Sunday evening.
- Post as a response to post @112:

  WW @ 4/5 Discussion topic suggestions
Current assignments

- PRO3 - due Monday, April 3 @ 5
  - for coercion, just print message when it should occur (no IR generation yet)

- HW3 - due Tuesday, April 4 @ 5
Upcoming project stages

- PR04 Intermediate code generation
  - Target due date: 4/14 (tentative)

- PR05 Machine independent optimizations
  - Target due date: 4/28 (tentative)

- PR06 Machine code generation
  - Target due date: 5/8 (tentative)
Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
Three address code instructions (see 6.2.1, pages 364-5)

1. \( x = y \text{ op } z \)
2. \( x = \text{ op } y \)  (treat \text{i2r} and \text{r2i} as unary ops)
3. \( x = y \)
4. \( \text{goto L} \)
5. \( \text{if x goto L} / \text{ifFalse x goto L} \)
6. \( \text{if x relop y goto L} \)
7. function calls:
   - \( \text{param x} \)
   - \( \text{call p, n} \)
   - \( y = \text{call p} \)
   - \( \text{return y} \)
8. \( x = y[i] \text{ and } x[i] = y \)
9. \( x = \&y, x = *y, *x = y \)
"The description of three-address instructions specifies the components of each type of instruction, but it does not specify the representation of these instructions in a data structure."

[p. 366]
Static Single Assignment (SSA) 
an additional constraint on the three address code

1) Each variable is assigned to exactly once.

\[ x = r + 1 \]
\[ y = s \times 2 \]
\[ x = 2 \times x + y \]
\[ y = y + 1 \]

\[ x_1 = r + 1 \]
\[ y_1 = s \times 2 \]
\[ x_2 = 2 \times x_1 + y_1 \]
\[ y_2 = y_1 + 1 \]
Static Single Assignment (SSA)
an additional constraint on the three address code

1) Each variable is assigned to exactly once.

2) Need $\phi$ function to merge split variables:

\[
\text{if (e) then } \{ x = a \} \text{ else } \{ x = b \}
\]
\[
y = x
\]

With SSA:
\[
\text{if (e) then } \{ x_1 = a \} \text{ else } \{ x_2 = b \}
\]
\[
y = \phi( x_1, x_2 )
\]
Type equivalence

Name equivalence: two types are equivalent if and only if they have the same name.

Structural equivalence: two types are equivalent if and only if they have the same structure. A type is structurally equivalent to itself (i.e. int is both name equivalent and structurally equivalent to int)
Name equivalence

```java
int x = 3;
int y = 5;
int z = x * y;
```

The type of `z` is `int`.
The type of `x * y` is `int`.
The names of the types are the same, so the assignment is legal.
Structural equivalence

struct S { int v; double w; };
struct T { int v; double w; };

int main() {
    struct S x;
    x.v = 1; x.w = 4.5;
    struct T y;
    x = y;
    return 0;
}

Under name equivalence the assignment is disallowed.

Under structural equivalence the assignment is permitted.

What does C do?
C does not allow the assignment

cat -3.2$ gcc type.c
type.c:9:5: error: assigning to 'struct S' from incompatible type
'struct T'
    x = y;
        ^  ~
1 error generated.
**Structural equivalence**

struct S { int v; double w; };  
struct T { int a; double b; };  

int main() {  
  struct S x;  
  x.v = 1; x.w = 4.5;  
  struct T y;  
  x = y;  
  return 0;  
}  

Should this be allowed?

Types and order of fields align, but names differ.
Consider...

```c
struct Rectangular { double x; double y; };
struct Polar { double r; double theta; };

int main() {
    struct Rectangular p;
    p.x = 3.14; x.y = 3.14;
    struct Polar q;
    q = p;
    return 0;
}
```

Should this be allowed?
Interpretation matters

polar interpretation

rectangular interpretation
Our Language

- primitive types: integer, real, Boolean, character, string
- user-defined types:
  - record types have names
    - type rec : ( real x y ; ) : ( x := 0 y := 0)
  - array types have names
    - type arr : 10 -> string : ( "" )
  - function types do not have names
    - function f : ( real : x ) -> rec { ... }