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/iybn4ndq1s3ei
survey

- good feedback (please complete by 5:00 this afternoon!)

- one recurring theme:
  - more hands-on / coding discussion
Workshop Wednesdays

- Wednesdays we'll devote class time to working on project-related issues.
- We have 6 teams:
  - ML: //TODO and compilerclub
  - C: J.R.R.Token, Not Google, sanjargaret+moo, and UBComputing
- We'll work in groups of teams
- In general, post issues by Sunday night in Piazza to focus Wednesday activities
PR02 issues (~20 minutes) - discuss one+ of:
- asc format (esp. scope id)
- reporting multiple errors
- meaningful error messages

PR03 (~30 minutes)
- tackling type checking (for

Can continue cross-team discussions in recitations this week: don't share code, but brainstorm solutions to problems you're having.
Grading status

- PR01 - graded
- PR02 - graded (*)
- HW1 - in progress
Current assignments

- HW2 - due Wednesday, March 29 @ 5

- 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
Machine independent optimizations

Machine dependent optimizations

Target 1

Target 2

Target n
"Many of the translation schemes can be implemented during either bottom-up or top-down parsing [...] All schemes can be implemented by creating a syntax tree and then walking the tree." [p. 357]
Directed Acyclic Graph (DAG)

- Similar to a syntax tree
- No repeated nodes: structure sharing
Ex. 6.1 [p 359]

\[ a + a \cdot (b - c) + (b - c) \cdot d \]
Ex. 6.1 [p 359]

\[ a + a \cdot (b - c) + (b - c) \cdot d \]
Ex. 6.1 [p 359]

\[ a + a \times (b - c) + (b - c) \times d \]
Ex. 6.1 [p 359]

\[ a + a \times (b - c) + (b - c) \times d \]

Things can be more complicated if expressions have side effects
Value-number method
Algorithm 6.3 [p. 361]

- Input: label op, node l, node r
- Output: The value number of a node in the array with signature \(<op,l,r>\)
- Method: Search the array for a node M with signature \(<op,l,r>\). If there is such a node, return the value number of M. If not, create in the array a new node N with signature \(<op,l,r>\) and return its value number.
Revisiting 6.1
see construction steps in figure 6.5 [p. 360]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
<td>$\rightarrow$ to ST entry for a</td>
</tr>
<tr>
<td>2</td>
<td>id</td>
<td>$\rightarrow$ to ST entry for b</td>
</tr>
<tr>
<td>3</td>
<td>id</td>
<td>$\rightarrow$ to ST entry for c</td>
</tr>
<tr>
<td>4</td>
<td>$-$</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>$*$</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$+$</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>id</td>
<td>$\rightarrow$ to ST entry for d</td>
</tr>
<tr>
<td>8</td>
<td>$*$</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>$+$</td>
<td>6</td>
</tr>
</tbody>
</table>
Three address code representation

\[ t_1 = b - c \]
\[ t_2 = a \times t_1 \]
\[ t_3 = a + t_2 \]
\[ t_4 = t_1 \times d \]
\[ t_5 = t_3 - t_4 \]
Three address code instructions (see 6.2.1, pages 364-5)

1. \( x = y \ op \ z \)
2. \( x = \ op \ y \) (treat i2r and r2i as unary ops)
3. \( x = y \)
4. goto L
5. if x goto L / ifFalse x goto L
6. if x relop y goto L
7. function calls:
   - param x
   - call p, n
   - y = call p
   - return y
8. \( x = y[i] \) 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]
Quadruples

Instructions have four fields:

\[ \text{op, arg1, arg2, result} \]

Example: \( t_3 = a + t_2 \) is represented as

<table>
<thead>
<tr>
<th>op</th>
<th>arg1</th>
<th>arg2</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>a</td>
<td>( t_2 )</td>
<td>( t_3 )</td>
</tr>
</tbody>
</table>

Example: \( t_4 = - c \) is represented as

<table>
<thead>
<tr>
<th>op</th>
<th>arg1</th>
<th>arg2</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>minus</td>
<td>( c )</td>
<td></td>
<td>( t_4 )</td>
</tr>
</tbody>
</table>
Quadruples

Identifiers would be pointers to symbol table entries. Compiler-introduced temporaries can be added to the symbol table.

<table>
<thead>
<tr>
<th>op</th>
<th>arg1</th>
<th>arg2</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>→ entry for a</td>
<td>→ entry for t_2</td>
<td>→ entry for t_3</td>
</tr>
</tbody>
</table>
**Triples**

Instructions have three fields:
- op, arg1, arg2

Example:
- \( t_2 = \ldots \)
- \( t_3 = a + t_2 \) is represented as

<table>
<thead>
<tr>
<th>line</th>
<th>op</th>
<th>arg1</th>
<th>arg2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>computation of ( t_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>+</td>
<td>( a )</td>
<td>( (5) )</td>
</tr>
</tbody>
</table>
Indirect triples

Because order matters (due to embedded references instead of explicit variables) it is more challenging to rearrange instructions with triples than with quadruples.

Indirect triples allow for easier reordering (see page 369).