Weekly team meetings with me:
- Doodle poll link in Piazza
- A few teams have not yet signed up

Wednesday (4/4) will be a workshop

- Post questions you’d like addressed in Piazza by Sunday (4/1) evening
- Post in @98

https://piazza.com/class/jcavahwt1zc181?cid=98
Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
Dealing with alignment

"On many machines, instructions [...] may expect integers to be aligned, that is, placed at an address divisible by 4" [p. 428]
Dealing with alignment

\[
\begin{align*}
\{ & \text{[ Boolean : } a \text{ ; integer : } x \text{ ; character } c \text{; real : } y \} \\
\{ & \text{[ character : } d \text{ ; integer : } r, s \} \ldots \} \\
\{ & \text{[ Boolean : } f, g \text{; real : } t \text{ ; character } h \} \ldots \} \\
\}
\end{align*}
\]

"On many machines, instructions [...] may expect integers to be aligned, that is, placed at an address divisible by 4" [p. 428]
Dealing with alignment

```
{ [ Boolean : a ; integer : x ; character c ; real : y ]

[ [ character : d ; integer : r , s ] ... ]

[ [ Boolean : f , g ; real : t ; character h ] ... ]

}
```

"On many machines, instructions [...] may expect integers to be aligned, that is, placed at an address divisible by 4" [p. 428]
Dealing with alignment

```
{ [ Boolean : a ; integer : x ; character c ; real : y ]

{ [ character : d ; integer : r , s ] ... }

{ [ Boolean : f , g ; real : t ; character h ] ... }
}
```

"On many machines, instructions [...] may expect integers to be aligned, that is, placed at an address divisible by 4" [p. 428]
Dealing with alignment

{ [ Boolean : a ; integer : x ; character c ; real : y ]

[ [ character : d ; integer : r , s ] ... ]

[ [ Boolean : f , g ; real : t ; character h ] ... ]

}

Blocks are aligned, but memory wasted to padding
Dealing with alignment

{ [ Boolean : a ; integer : x ; character c ; real : y ]

[ character : d ; integer : r , s ] ... }

{ [ Boolean : f , g ; real : t ; character h ] ... }

}
Dealing with alignment

{ [ Boolean : a ; integer : x ; character c ; real : y ]

{ [ character : d ; integer : r , s ] ... }

{ [ Boolean : f , g ; real : t ; character h ] ... }

}

Blocks are aligned, padding needed before embedded scope block.
Three address code instructions (see 6.2.1, pages 364-5)

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

Need operators for various types: let's use \( <t> op \), as in \( i+ \) or \( r* \) or \( b< \)
resolving overloaded operators

Fig 6.27, p. 390: after type checking and possibly coercions, then 6.6.3 choose correct operation to perform

\[ E \rightarrow E_1 + E_2 \]
\[
\{ \text{E.type} = \text{max(E1.type, E2.type)}; \\
\text{a1} = \text{widen(E1.addr, E1.type, E.type)}; \\
\text{a2} = \text{widen(E2.addr, E2.type, E.type)}; \\
\text{E.addr} = \text{new Temp}(); \\
\text{gen(E.addr} \ 'i= \ a1 \ '+ \ a2); \} \]
resolving overloaded operators

Fig 6.27, p. 390: after type checking and possibly coercions, then 6.5.3 choose correct operation to perform

\[
E \rightarrow E_1 + E_2 \\
\{ \text{E.type} = \text{max}(E_1\text{.type}, E_2\text{.type}); \\
a_1 = \text{widen}(E_1\text{.addr, E_1\text{.type, E.type}}); \\
a_2 = \text{widen}(E_2\text{.addr, E_2\text{.type, E.type}}); \\
E\text{.addr} = \text{new Temp}(); \\
g\text{en}(E\text{.addr} '=' a_1 '+' a_2); \}
\]

Are we doing int addition or floating point addition? i+ vs. f+ ??
Skip 6.5.4-6.5.6
Control flow

- Booleans to control flow
- Booleans as values
Boolean expressions

¬x  x & y  x ∣ y

We will do short-circuit evaluation

if (x ∣ y & z) then { a } else { b } is translated as

if x goto LA
ifFalse y goto LB
ifFalse z goto LB

LA:  A
    goto END

LB:  B

END: (next instruction)
A more concrete example:

```plaintext
if ( r < s | r = s & 0 < s) then { A } else { B } is translated as

if r < s goto LA
ifFalse r = s goto LB
ifFalse 0 < s goto LB

LA:   A
goto END

LB:   B

END: (next instruction)
```
Flow-of-Control (6.3.3)

if ( B ) then S1 else S2

B.true =NewLabel()
B.false =NewLabel()
S.next = S1.next = S2.next
S.code = B.code || label(B.true) || S1.code || gen('goto' S.next) || label(B.false) || S2.code
Flow-of-Control (6.3.3)

$S \rightarrow \text{if ( B ) then } S_1$

$B.true = \text{newlabel()}$

$B.false = S.next = S_1.next$

$S.code = B.code \parallel \text{label(B.true)} \parallel S_1.code$

ifTrue:
  goto LS1

ifFalse:
  goto END

LS1

B.code

S1.code

END
while ( B ) then S1

begin = newlabel()
B.true = newlabel()
B.false = S.next
S1.next = begin
S.code = label(begin) || B.code || label(B.true) || S1.code || gen('goto' begin)

BEGIN

ifTrue:
goto LS1

ifFalse:
goto END

LS1

S1.code

goto BEGIN

END
Value of Boolean expression

"When \( E \) appears in \( S \rightarrow \text{while} (E) \ S_1 \), method \textit{jump} is called at node \( E.n \) [...] When \( E \) appears in \( S \rightarrow \text{id} = E; \), method \textit{rvalue} is called at node \( E.n \)" [p. 408]
Translation of: $x = a < b \land \land c < d$

```plaintext
ifFalse a < b goto L1
ifFalse c < d goto L1
t = true
goto L2
L1: t = false
L2: x = t
```