Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
Continuing example from last class
Scopes

records (in separate symbol table), sequence of declarations at start of sblock

dblocks (6.3.5 and 6.3.6)

Since declarations must be gathered together at the start of an sblock, and cannot themselves be directly nested: keep running offset, but remember old offset when entering embedded scope.

```
dblock → '[
    { Stack.push(offset); } declaration-list ']
    { offset=Stack.pop(); }
```

offset = 0

```
{ [ integer : x , y ]

push offset = 8 onto stack

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

pop offset = 8 from stack
push offset = 8 onto stack

offset = 8

offset = 9

offset = 10

{ [ real : x , z ] .... }

pop offset = 8 from stack
```

integer: x

integer: y

real: x

real: z

integer: x

integer: y

real: x

real: z

Boolean: y

character: z

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10

offset = 8

offset = 4

offset = 8

offset = 8

offset = 16

offset = 24

offset = 8

offset = 9

offset = 10
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

"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

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

\[ \{ \text{character} : d ; \text{integer} : r, s \} \ldots \} \]

\[ \{ \text{Boolean} : f, g ; \text{real} : t ; \text{character} h \} \ldots \} \]

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

Blocks are not aligned.

A block of size N bytes typically needs be aligned to an address divisible by N, where N is an integral power of 2 (1, 2, 4, 8)
Dealing with alignment

Blocks are aligned, but memory wasted to padding.

C will lay fields out in the order listed in the struct declaration.
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, no padding needed here.
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.
Offsets and alignment in the project

- The offsets for each variable in a scope is stored in its symbol table entry.
- The offsets must respect alignment constraints.
  - assume real is aligned to an 8-byte address boundary
  - assume int is aligned to a 4-byte boundary
  - assume smaller types can be at any address
  - assume reserve returns an address on an 8-byte boundary
Offsets and alignment in the project

- You may align using padding alone.

- You may align using a combination of re-organization of fields (large blocks before small blocks) and padding as necessary.
IR: a motivating example
Boolean expressions can be evaluated
- to determine the flow of control
- for their value
Boolean expressions

Examples: \( \neg X \), \( X \land Y \), \( X \lor Y \)
Boolean expressions

- Examples: \( \neg X \) \( X \land Y \) \( X \lor Y \)
- We will do short-circuit evaluation
Boolean expressions

- Examples: !X  X & Y  X | Y
- We will do short-circuit evaluation
- For example:
  
  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)
Boolean expressions

A concrete exercise - how is this translated?

if (r < s | (r = s & 0 < s)) then { A } else { B }
Boolean expressions

- A concrete exercise - how is this translated?
- if (r < s | (r = s & 0 < s)) then {A} else {B}

Here's a summary of the Intermediate Representation (IR) that we'll be using.

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

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
Boolean expressions

- A concrete exercise - how is this translated?
- \( \text{if } (r < s \mid (r = s \& 0 < s)) \text{ then } \{ A \} \text{ else } \{ B \} \)

Exercise: try to come up with a suitable translation.

Three address code instructions
(see 6.2.1, pages 364-5)

1. \( x = y \text{ op } z \)
2. \( x = \text{ op } y \) \hspace{1cm} (treat i2r and r2i as unary ops)
3. \( x = y \)
4. \( \text{goto L} \)
5. \( \text{if } x \text{ goto L / } \text{ifFalse x goto L} \)
6. \( \text{if } x \text{ relop y goto L} \)
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. \( x = \& y, x = *y, *x = y \)

We'll start with these.
We'll spend significant time on function calls later.
We'll explore these as needed later on.
Boolean expressions

A concrete exercise - how is this translated?

if \( r < s \lor (r = s \land 0 < s) \) then \{ A \} else \{ B \}

This has the same form as our example:

if \((X \lor (Y \land Z))\) 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)

Three address code instructions (see 6.2.1, pages 364-5)

1. \( x = y \; \text{op} \; z \)  
   (treat i2r and r2i as unary ops)
2. \( x = \text{op} \; y \)
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. \( x = \Delta y, \; x = *y, \; *x = y \)

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.
**Boolean expressions**

- A concrete exercise - how is this translated?

if ( \( r < s \) \( \mid (r = s \& 0 < s) \)) then \{ A \} else \{ B \}

This has the same form as our example:

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)

**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. \( x = &y, x = *y, *x = y \)

© 2020 Carl Alphonse - Reproduction of this material is prohibited without the author's consent
Boolean expressions

- A concrete exercise - how is this translated?

\[
\text{if ( } r < s \lor (r = s \land 0 < s) \text{) then } \{ A \} \text{ else } \{ B \}
\]

This has the same form as our example:

\[
\text{if ( } r < s \lor (Y \land Z) \text{) then } \{ A \} \text{ else } \{ B \}
\]

\[
\text{if } r < s \text{ goto LA}
\]
\[
\text{ifFalse } Y \text{ goto LB}
\]
\[
\text{ifFalse } Z \text{ goto LB}
\]

LA:  A

goto END

LB:  B

END: (next instruction)

Three address code instructions (see 6.2.1, pages 364-5)

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

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.
Boolean expressions

- A concrete exercise - how is this translated?

- if \( r < s \lor (r = s \land 0 < s) \) then \{ A \} else \{ B \}
- if \( X \lor (Y \land Z) \) then \{ A \} else \{ B \}

This has the same form as our example:

if \( r < s \lor (r = s \land 0 < s) \) then \{ A \} else \{ B \}

is translated as

if \( r < s \) goto LA
ifFalse \( r = s \) goto LB
ifFalse \( Z \) goto LB
LA:   A
goTo END
LB:   B
END: (next instruction)

Three address code instructions
(see 6.2.1, pages 364-5)

1. \( x = y \text{ op } z \)
2. \( x = \text{ op } y \) \[\text{(treat i2r and r2i as unary ops)}\]
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 \( p, n \)
   - \( y = \text{ call } p \)
   - return \( y \)
8. \( x = y[i] \) and \( x[i] = y \)
9. \( x = \& y, x = \ast y, \ast x = y \)

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.
Boolean expressions

- A concrete exercise - how is this translated?

- if \( r < s \) or \((r = s \& 0 < s)\) then \{ A \} else \{ B \}
- if \( x \) \& \( y \) \& \( z \) then \{ A \} else \{ B \}

This has the same form as our example:

if \( r < s \) or \((r = s \& 0 < s)\) 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)

Three address code instructions
(see 6.2.1, pages 364-5)

1. \( x = y \) op \( z \)
2. \( x = y \) op \( y \)  \hspace{1cm} \text{(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. \( x = \& y, x = *y, *x = y \)

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.
IR: the general case
Flow-of-Control (6.3.3)

if ( B ) then S1 else S2

Let's generalize from the previous concrete example to one with an arbitrary Boolean expression B.

We assume that IR instructions are placed into an array.
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

going to LS1

ifTrue:
goto LS1
ifFalse:
goto LS2

goto END

END
Flow-of-Control (6.3.3)

\[ S \rightarrow \text{if ( } B \text{ ) then } S1 \]

\[ B\.true = \text{newlabel()} \]
\[ B\.false = S\.next = S1\.next \]
\[ S\.code = B\.code || \]
\[ \text{label}(B\.true) || S1\.code \]
Flow-of-Control (6.3.3)

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

B.code

ifTrue:
goto LS1

ifFalse:
goto END

LS1

S1.code

goto BEGIN

END
Boolean expressions: value or control flow?
6.6.6 Boolean values and jumping code

"S → id = E; | if (E) S | while (E) S | S S

Nonterminal E governs the flow on control in S → while (E) S1. The same nonterminal E denotes a value in S → id = E; [...]

[p. 408]
6.6.6 Boolean values and jumping code

"Suppose that attribute E.n denotes the syntax-tree node for an expression E and that nodes are objects. Let method jump generate jumping code at an expression node, and let method rvalue generate code to compute the value of the node into a temporary."

[p. 408]
Value of Boolean expression

"When E appears in S -> while (E) S1, method jump is called at node E.n
[...] When E appears in S -> id = E;, method rvalue is called at node E.n" [p. 408]
"If E has the form E1 + E2, the method call E.n.rvalue() generates code as discussed in section 6.4." [p. 408]

"E-> E1 + E2
E.addr = new Temp()
E.code = E1.code || E2.code || gen(E.addr '=' E1.addr '+' E2.addr)" [p. 379]

"If E has the form E1 && E2 we first generate jumping code for E and then assign true or false to a new temporary t at the true and false exits, respectively, from the jumping code." [p. 408]

Translation of: x = a<b && c<d
    ifFalse a < b goto L1
    ifFalse c < d goto L1
    t = true
goto L2
L1:   t = false
L2:   x = t