Project 10: IR Code Generation, Part 3

Finish IR Code Generation
Optional Extension: Peephole Optimizer

Files:

```
Generator.java
Peephole.java   -- "dummy stub"
tst/   -- Contains all of the p9 tests plus more
Main.java  }
Main.jar   }
makefile   }  Slight modifications
runAll     }
IR.java    }
<others>   -- unchanged
```

An Optimization in "genAssignStmt"

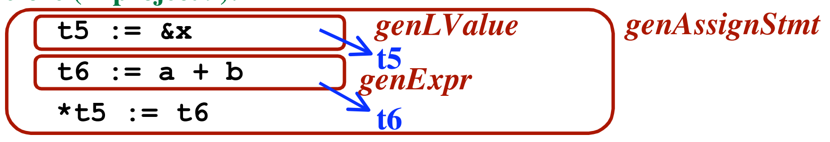
Want to reduce temporaries
Watch for special case:
"Lefthand side is a simple variable"
...and avoid calling genLValue()

Example:

PCAT Source:

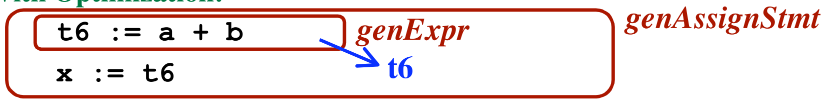
```
x := a + b;
```

Before (in project 9):



```
t5 := &x
t6 := a + b
*t5 := t6
```

With Optimization:



```
t6 := a + b
x := t6
```

If the lefthand side is not a simple variable...
Call genLValue() and generate "store" instruction.

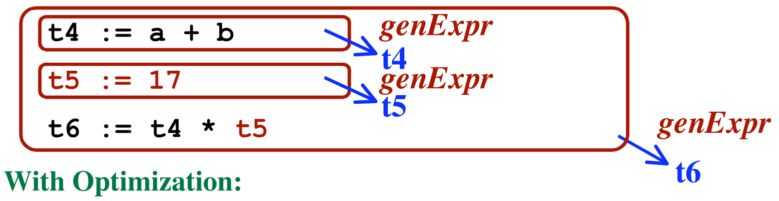
Optimization #2

Goal: Reduce temporaries associated with constants!

PCAT Source:

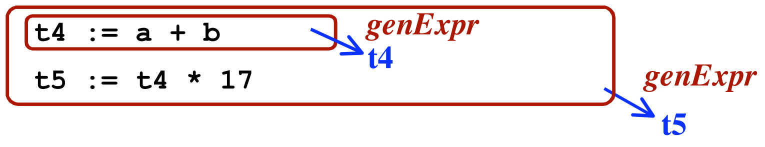
```
(a + b) * 17
```

Before (in project 9):



```
t4  := a + b
t5  := 17
t6  := t4 * t5
```

With Optimization:



```
t4  := a + b
t5  := t4 * 17
```

```
E.Code
E.Place
```

Before:

genExpr()

- creates a temporary
- generates code to move the value into the temporary
- returns the temporary

With Optimization:

genExpr()

When called on a constant...

- will return the value directly

genExpr() can now return...

- **A Variable**
  - `Ast.Formal`
  - `Ast.Variable`
- **A Value**
  - `Ast.IntegerConst`
  - `Ast.RealConst`

Runtime Errors

runtimeError1:
Heap allocation failed.

runtimeError2:
Pointer is NIL. (during dereferencing)

runtimeError3:
Read statement failed.

runtimeError4:
Array index is out of bounds.

```
0 <= index < N
```

runtimeError5:
In an array constructor, the count is <= 0.

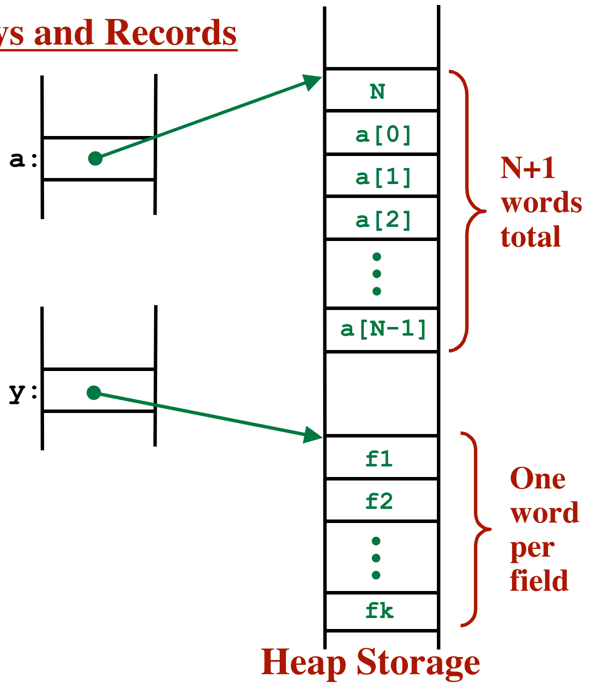
```
a := AType {{ 1, 2, 3, 4 }};
a := AType {{ 100 of 0, 200 of -1 }};
a := AType {{ i*10 of -1, 3, x+y, k of 0 }};
```

Boilerplate

Canned, fixed material inserted into the SPARC output target file.

Layout of Arrays and Records

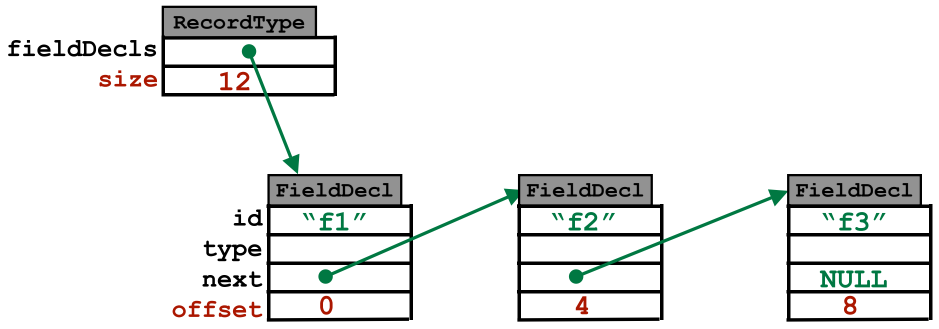
```
var a: array of ...;

var y: record
         f1: ...;
         f2: ...;
         ...
         fk: ...;
       end;
```



Every data value is 1 word long (4 bytes)

Record Sizes and Field Offsets

Each field is 4 bytes.
Compute and set: `RecordType.size` and `FieldDecl.offset`
Must take a look at `TypeDecl.compoundType`.



Dealing With L-Values

`x`	Variable
`a[...expr...]`	ArrayDeref
`r.fieldName`	RecordDeref

} Can be used as
- L-Value
  `genLValue()`
- R-Value
  `genValueOf()`

Dealing With L-Values

`x`	Variable	Can be used as L-Value `genLValue()` R-Value `genValueOf()`
`a[...expr...]`	ArrayDeref	
`r.fieldName`	RecordDeref	

How we deal with...
- Variable
- ArrayDeref
- RecordDeref

Will differ depending on whether it is used as...
- L-Value
  - Generate code to move an ***address*** into a temp.
- R-Value
  - Generate code to move a ***value*** into a temp.

Idea:
- Eliminate **genVariable**, **genArrayDeref**, **genRecordDeref**.
- Include code directly in **genLValue** and **genValueOf**
  - ... since it will be slightly different in each.

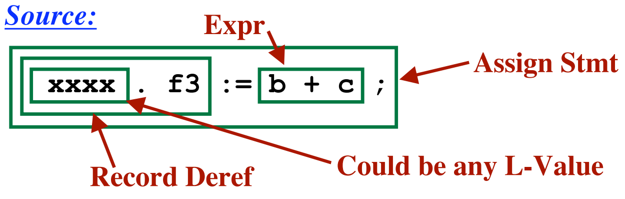
```
genValueOf (ValueOf p,...) {
  lv = p.lValue;
  if lv instanceOf Variable {
    ...
  } else if lv instanceOf RecordDeref {
    ...
  } else if lv instanceOf ArrayDeref {
    ...
  }
}
```

```
genLValue (LValue p) {
  if p instanceOf Variable {
    ...
  } else if p instanceOf RecordDeref {
    ...
  } else if p instanceOf ArrayDeref {
    ...
  }
}
```

Record Dereferencing (as an L-Value)

Field	Offset
f1	0
f2	4
f3	8
⋮	⋮

Source:

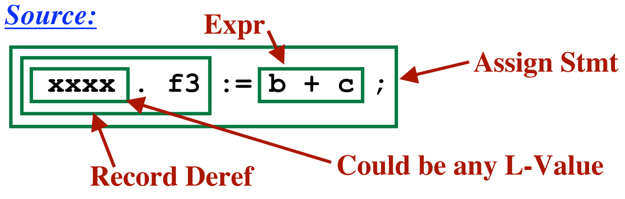


```
xxxx . f3 := b + c ;
```

Record Dereferencing (as an L-Value)

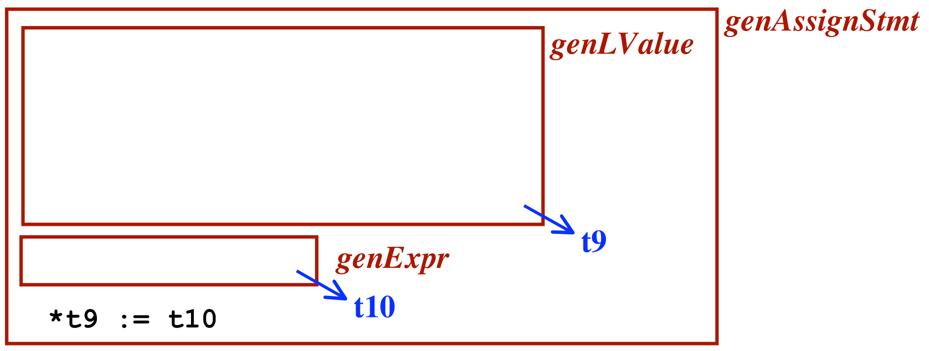
Field	Offset
f1	0
f2	4
f3	8
⋮	⋮

Source:



```
xxxx . f3 := b + c ;
```

IR Code:

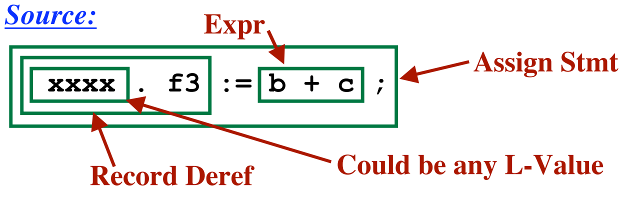


```
*t9 := t10
```

Field	Offset
f1	0
f2	4
f3	8
⋮	⋮

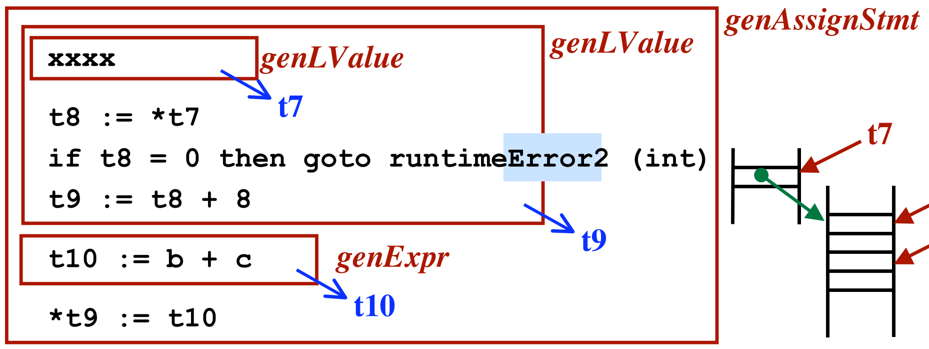
Record Dereferencing (as an L-Value)

Source:

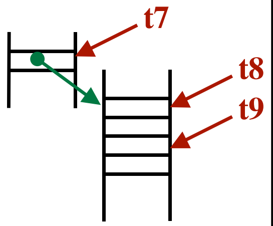


```
xxxx . f3 := b + c ;
```

IR Code:



```
xxxx
t8  := *t7
if t8 = 0 then goto runtimeError2 (int)
t9  := t8 + 8
t10  := b + c
*t9  := t10
```



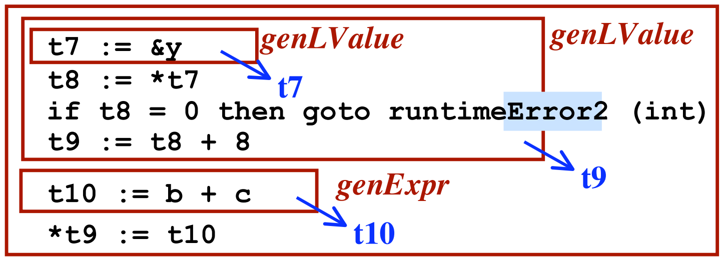
Field	Offset
f1	0
f2	4
f3	8
⋮	⋮

Example

Source:

```
y . f3 := b + c ;
```

IR Code:



```
t7  := &y
t8  := *t7
if t8 = 0 then goto runtimeError2 (int)
t9  := t8 + 8
t10  := b + c
*t9  := t10
```

Field	Offset
f1	0
f2	4
f3	8
⋮	⋮

Example

Source:

```
y . f3 := b + c ;
```

IR Code:

```
t7  := &y
t8  := *t7
if t8 = 0 then goto runtimeError2 (int)
t9  := t8 + 8
t10  := b + c
*t9  := t10
```

genLValue → t7

genLValue → t9

genExpr → t10

Ughhh!!!

Field	Offset
f1	0
f2	4
f3	8
⋮	⋮

Example

Source:

```
y . f3 := b + c ;
```

IR Code:

```
t7  := &y
t8  := *t7
if t8 = 0 then goto runtimeError2 (int)
t9  := t8 + 8
t10  := b + c
*t9  := t10
```

genLValue → t7

genLValue → t9

genExpr → t10

Ughhh!!!

Optimization: Watch for special case RecordDeref.lValue is a simple variable!

```
if y = 0 then goto runtimeError2 (int)
t9  := y + 8
t10  := b + c
*t9  := t10
```

genLValue → t9

genExpr → t10

genLValue

```
genLValue (LValue p) {
  if p instanceOf Variable {
    :  generate "t3 := &x"
    :  return t3
  } else if p instanceOf RecordDeref {
    :
    :



    :
    :
  } else if p instanceOf ArrayDeref {
    :
    :
  }
}
```

genLValue

```
genLValue (LValue p) {
  if p instanceOf Variable {
    :  generate "t3 := &x"
    :  return t3
  } else if p instanceOf RecordDeref {
    :
    :
    if p.lValue instanceOf Variable {
      :
      :  generate optimized version
    } else {
      :  call genLValue
      :  generate LoadIndirect instruction
    }
    :
    :
  } else if p instanceOf ArrayDeref {
    :  Do the same optimization for ArrayDeref
    :
  }
}
```

Record Deref Optimization #2

Field	Offset
f1	0
f2	4
f3	8
⋮	⋮

Source:

```
xxxx . f1 := b + c ;
```

A field with offset = 0

IR Code:

```
xxxx
t8  := *t7
if t8 = 0 then goto runtimeError2 (int)
t9  := t8 + 0
t10  := b + c
*t9  := t10
```

t7

t9

t10

Ughhh!!!

Record Deref Optimization #2

Field	Offset
f1	0
f2	4
f3	8
⋮	⋮

Source:

```
xxxx . f1 := b + c ;
```

A field with offset = 0

IR Code:

```
xxxx
t8  := *t7
if t8 = 0 then goto runtimeError2 (int)
t9  := t8 + 0
t10  := b + c
*t9  := t10
```

t7

t9

t10

Ughhh!!!

Optimization:

Eliminate the ADD instruction
Avoid creating the temporary (t9)
Just return this variable

Example

Source:

```
y.f1 := a + b;
```

Both optimizations apply!

IR Code:

```
if y = 0 then goto runtimeError2
t10 := a + b
*y := t10
```

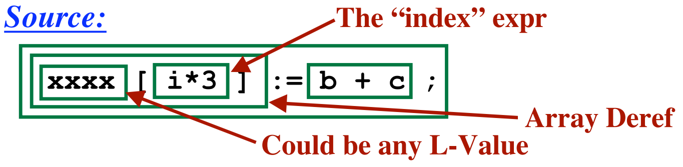
Array Dereferencing (as an L-Value)

Source:

```
xxxx [ i*3 ] := b + c ;
```

Array Dereferencing (as an L-Value)

Source:

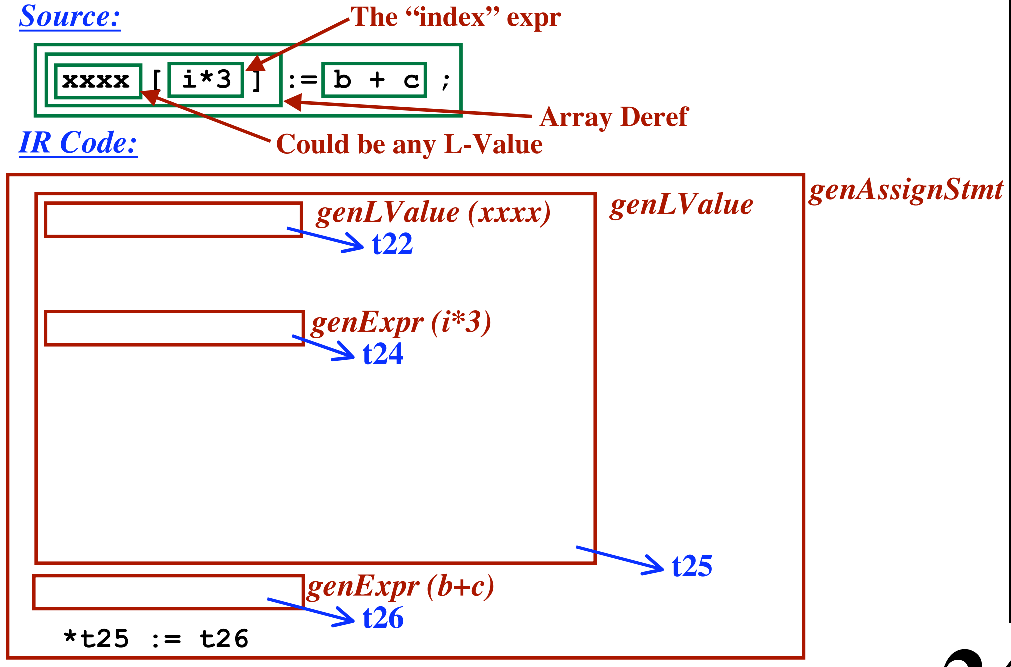


```
      a[i*3]
    r.f[i*3]
a[j*7][i*3]
```

Array Dereferencing (as an L-Value)

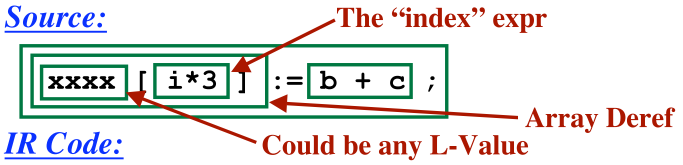
Source:

IR Code:

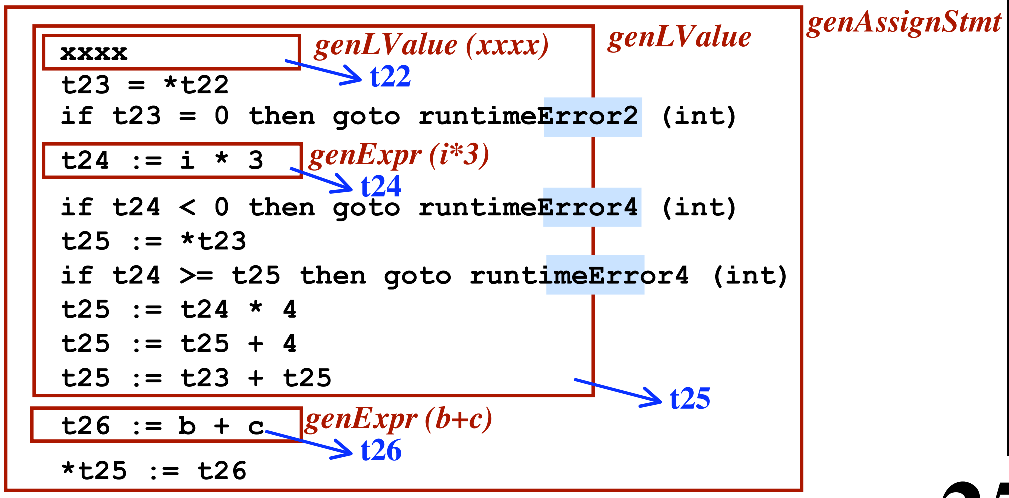


Array Dereferencing (as an L-Value)

Source:



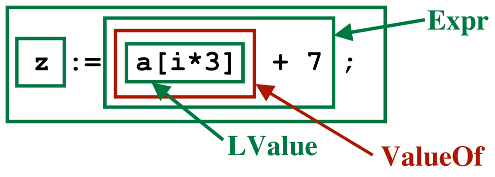
IR Code:



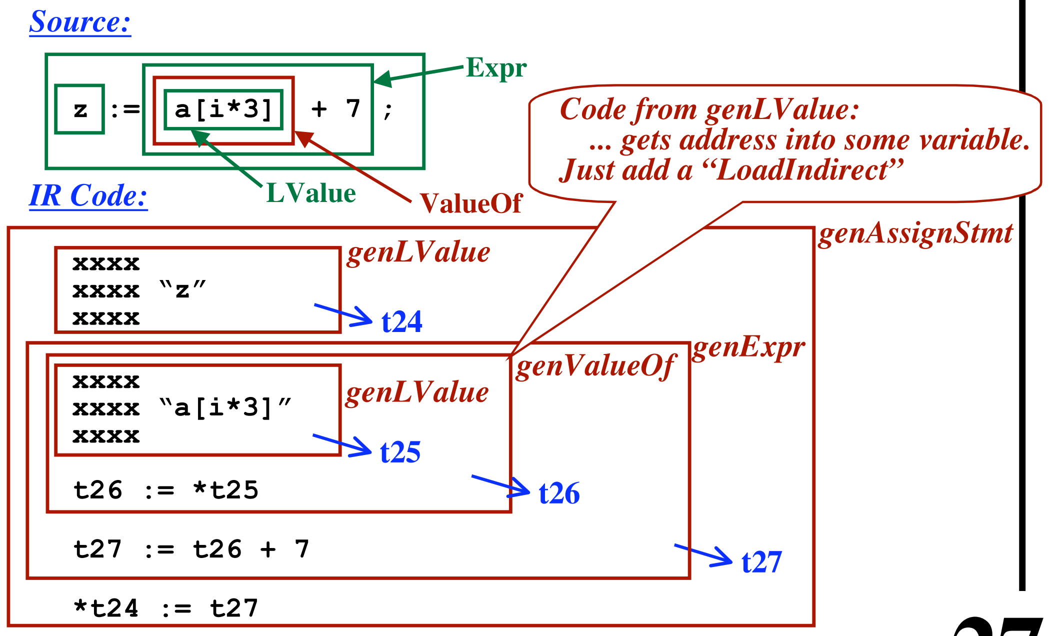
```
xxxx
t23 = *t22
if t23 = 0 then goto runtimeError2 (int)
t24 := i * 3
if t24 < 0 then goto runtimeError4 (int)
t25 := *t23
if t24 >= t25 then goto runtimeError4 (int)
t25 := t24 * 4
t25 := t25 + 4
t25 := t23 + t25
t26 := b + c
*t25 := t26
```

Dealing With R-Values: ArrayDeref and RecordDeref

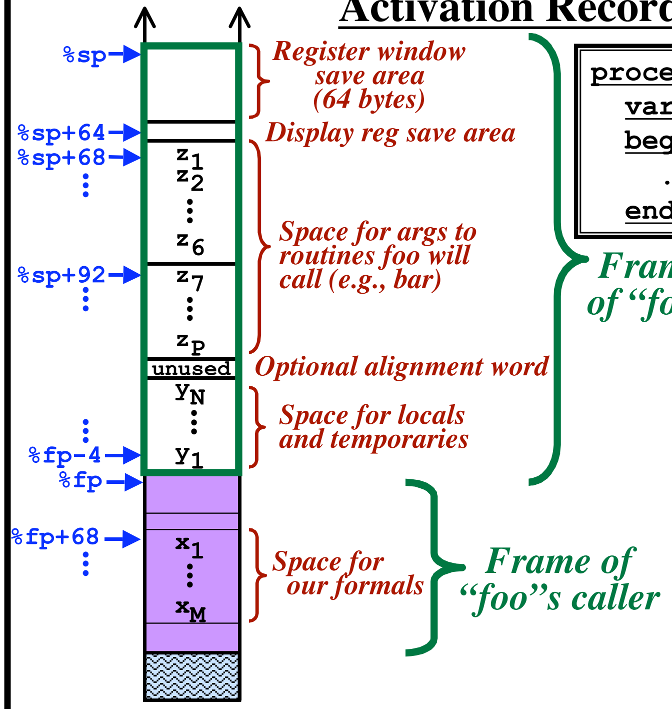
Source:



Dealing With R-Values: ArrayDeref and RecordDeref



Activation Record Layout



```
procedure foo (x1,x2,...xM)
   var y1,y2,...yN;
   begin
      ... bar(z1,z2,...zP)...
   end;
```

New Fields:

```
   VarDecl.offset
   Formal.offset
   Body.frameSize
```

You must fill these in

It walks the AST, visiting all
```
Bodys
ProcDecls
VarDecls
Formals
```

Method "printOffsets()" has been added to IR.java

Computation of frameSize...

numberOfLocals
maxNumberOfArgsUsed

Constants in Generator.java

```
static final int INITIAL_VARIABLE_OFFSET      =  -4;
static final int VARIABLE_OFFSET_INCR         =  -4;
static final int INITIAL_FORMAL_OFFSET        = +68;
static final int FORMAL_OFFSET_INCR           =  +4;
static final int REGISTER_SAVE_AREA_SIZE      = +64;
static final int DISPLAY_REG_SAVE_AREA_OFFSET = +64;
```

Don't Forget:
If frame size is not a multiple of 8....
then add 4 (the optional, unused alignment word)
to make it a multiple of 8.

New IR Instruction: alloc

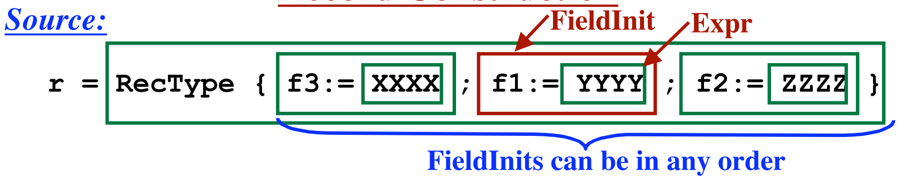
```
t3 := alloc (n)
```

Result
Arg 1

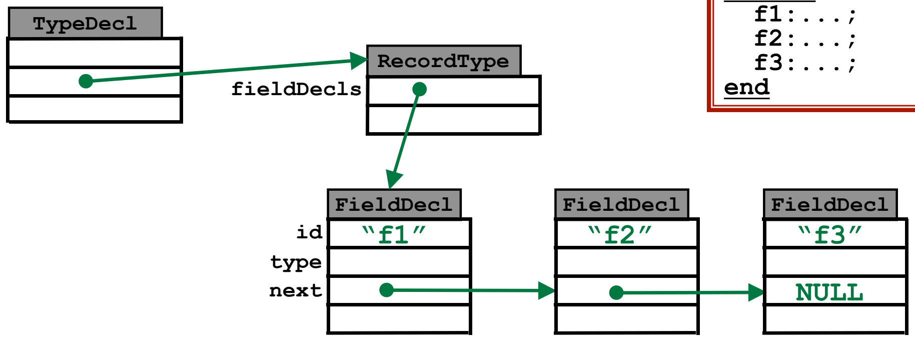
Allocate "n" bytes on the heap
Set "result" to a pointer to the memory
    Save addr in t3
...or set to zero if problems.
Will call "calloc" from library.
Used for
- Array Constructors
- Record Constructors

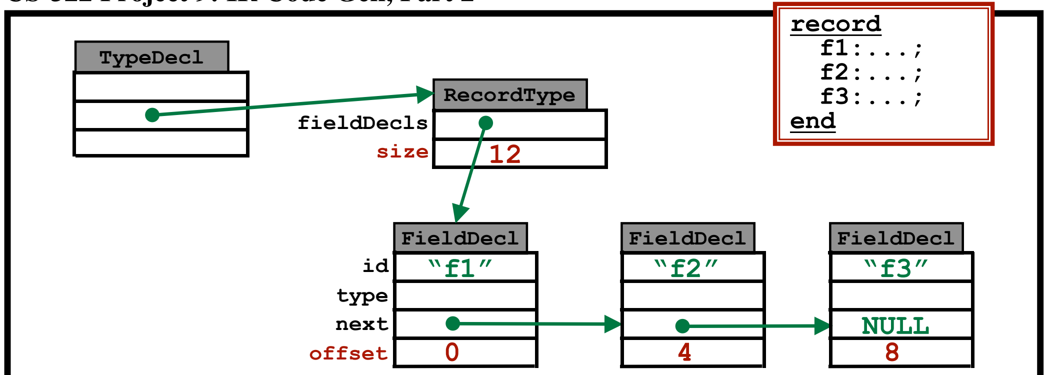
Record Construction

Source:


FieldInit
Expr
r = RecType { f3:= XXXX ; f1:= YYYY ; f2:= ZZZZ } ;
FieldInits can be in any order

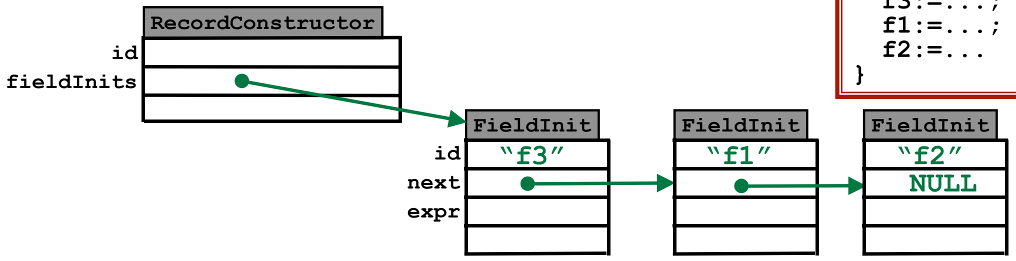
```
record
   f1:...;
   f2:...;
   f3:...;
end
```

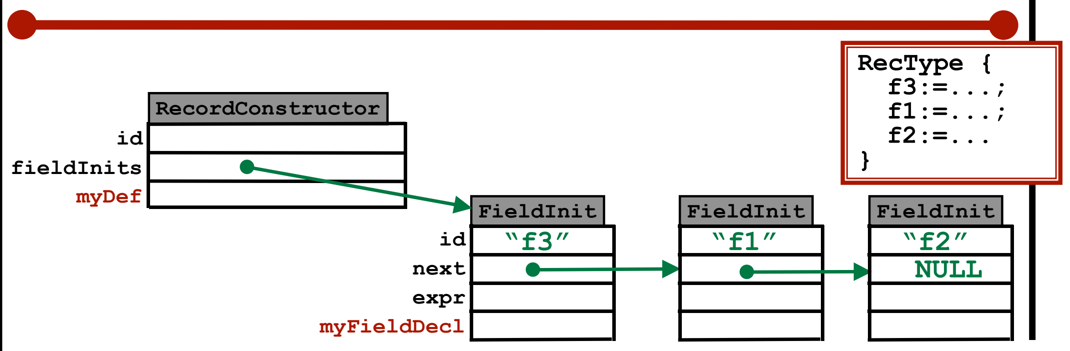




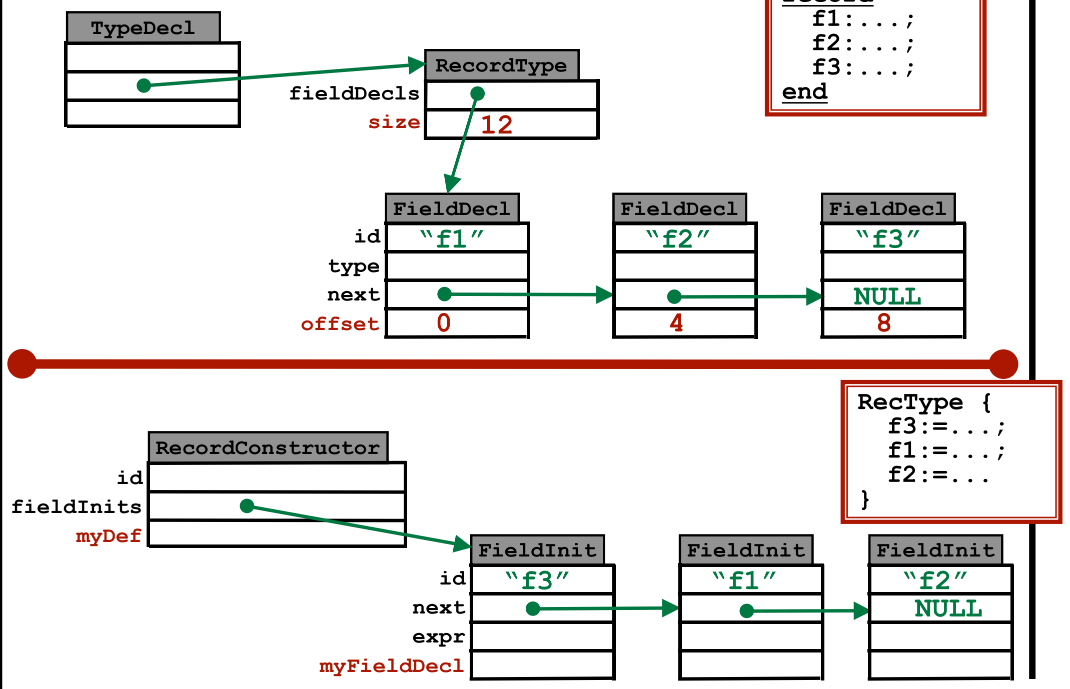
```
record
   f1:...;
   f2:...;
   f3:...;
end
```

```
RecType {
   f3:=...;
   f1:=...;
   f2:=...
}
```



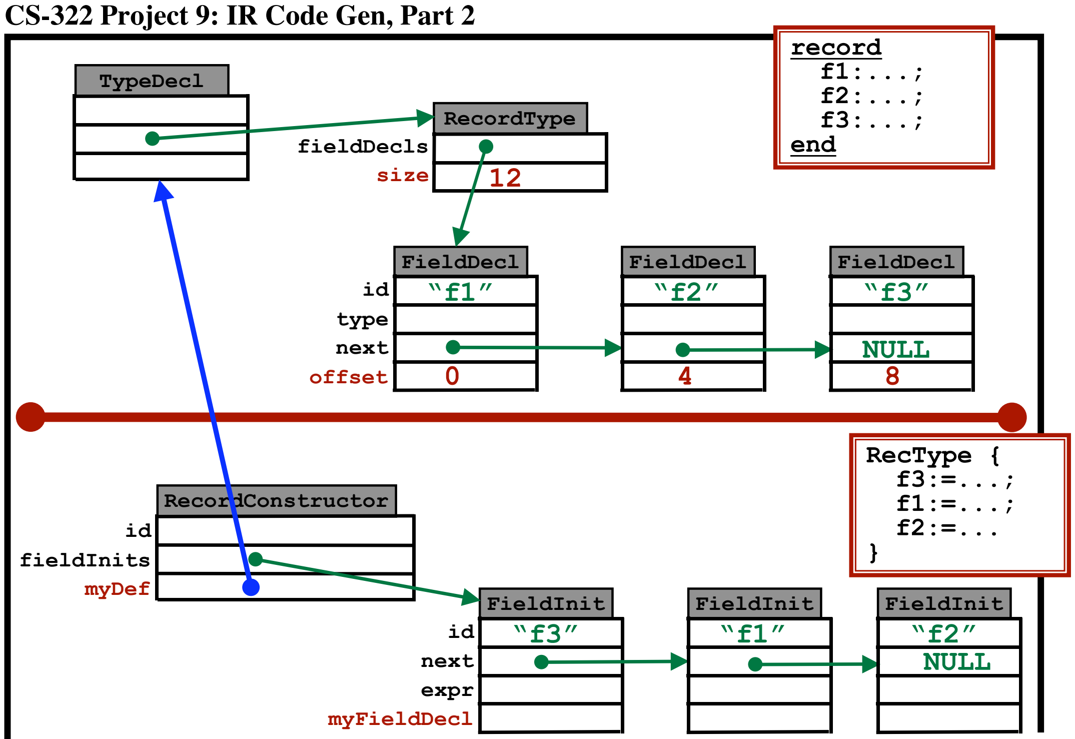


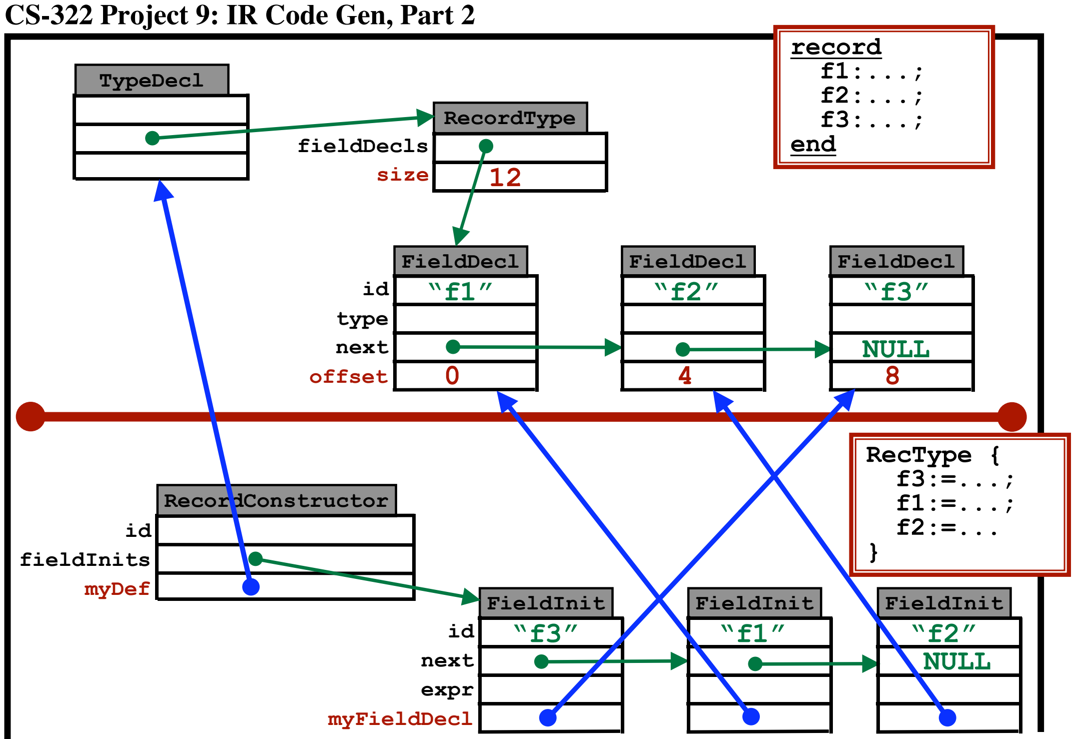
```
RecType {
  f3:=...;
  f1:=...;
  f2:=...
}
```



```
record
  f1:...;
  f2:...;
  f3:...;
end
```

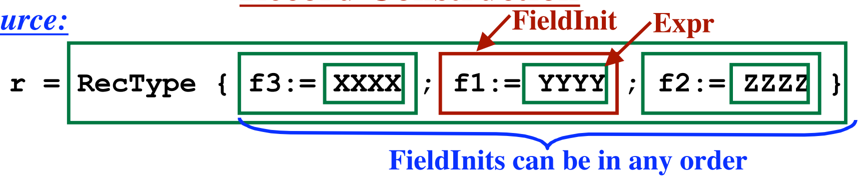
```
RecType {
  f3:=...;
  f1:=...;
  f2:=...
}
```


record
f1:...;
f2:...;
f3:...;
end
TypeDecl
RecordType
fieldDecls
size
12
FieldDecl
FieldDecl
FieldDecl
id
type
next
offset
"f1"
"f2"
"f3"
NULL
0
4
8
RecType {
f3:=...;
f1:=...;
f2:=...
}
RecordConstructor
id
fieldInits
myDef
FieldInit
FieldInit
FieldInit
id
next
expr
myFieldDecl
"f3"
"f1"
"f2"
NULL


record
f1:...;
f2:...;
f3:...;
end
TypeDecl
RecordType
fieldDecls
size
12
FieldDecl
FieldDecl
FieldDecl
id
type
next
offset
"f1"
"f2"
"f3"
NULL
0
4
8
RecType {
f3:=...;
f1:=...;
f2:=...
}
RecordConstructor
id
fieldInits
myDef
FieldInit
FieldInit
FieldInit
id
next
expr
myFieldDecl
"f3"
"f1"
"f2"
NULL

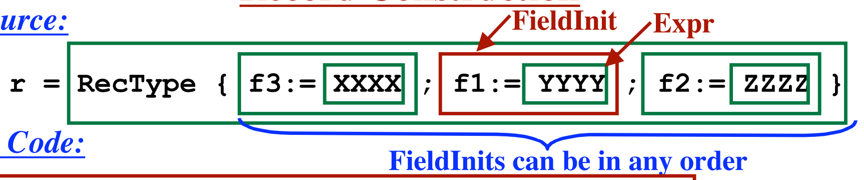
Record Construction

Source:

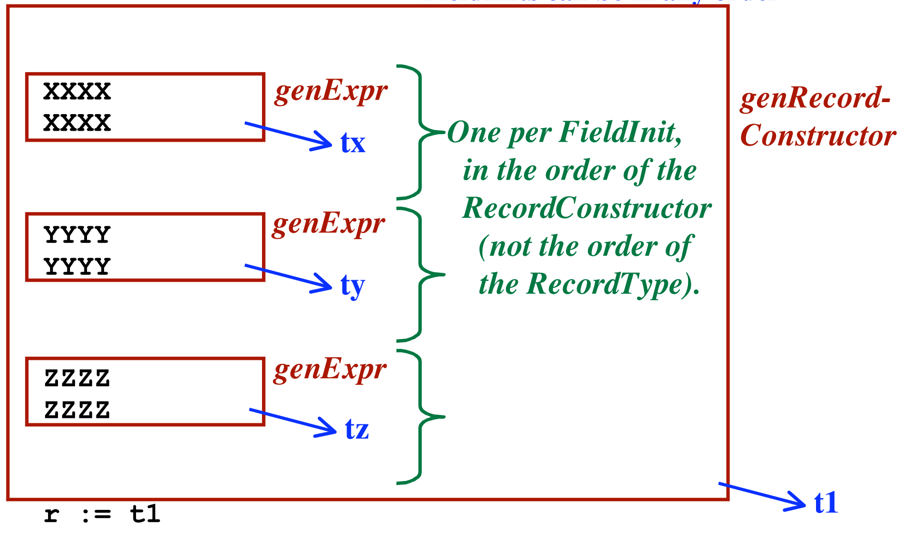

r = RecType { f3:= XXXX ; f1:= YYYY ; f2:= ZZZZ } ;
FieldInit
Expr
FieldInits can be in any order

Record Construction

Source:


r = RecType { f3:= XXXX ; f1:= YYYY ; f2:= ZZZZ } ;
FieldInit
Expr
FieldInits can be in any order

IR Code:


XXXX
XXXX
genExpr
tx
YYYY
YYYY
genExpr
ty
ZZZZ
ZZZZ
genExpr
tz
One per FieldInit, in the order of the RecordConstructor (not the order of the RecordType).
genRecord-Constructor
t1
r := t1

Record Construction

Source:

```
r = RecType { f3:= XXXX ; f1:= YYYY ; f2:= ZZZZ } ;
```

FieldInit

Expr

FieldInits can be in any order

IR Code:

```
t1  := alloc(12)
if t1 = 0 then goto runtimeError1 (int)
XXXX
XXXX
t2  := t1 + 8
*t2  := tx
YYYY
YYYY
t2  := t1 + 0
*t2  := ty
ZZZZ
ZZZZ
t2  := t1 + 4
*t2  := tz
r  := t1
```

genExpr → tx

genExpr → ty

genExpr → tz

→ t1

One per FieldInit, in the order of the RecordConstructor (not the order of the RecordType).

genRecord-Constructor

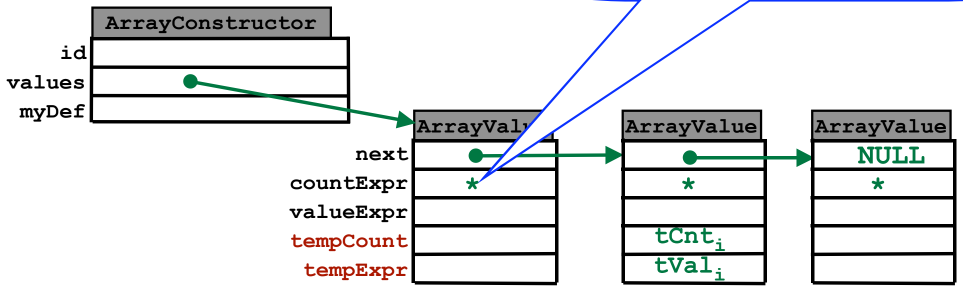
To find the correct offset, use fieldInit.myFieldDecl.offset

Array Constructors

```
type A is array of ...;

   A {{ count₁ of expr₁, ..., countK of exprK }}
```

$\underbrace{\texttt{count}_1 \texttt{ of}}_{\text{optional (default = 1)}}$

The CountExpr could be NULL



Examples:

```
a := MyArr {{ 1000 of -1 }};
b := MyArr {{ 101, 102, 103, 104 }};
c := MyArr {{ i*4 of foo(i), k of bar(x) }};
```

Expression; Could be negative!!!

genArrayConstructor

```
                  ...genExpr(countExpr_i)...


                  ...genExpr(valueExpr_i)...
```

Repeated K times

•••

Initialize count$_i$ *elements to* value$_i$

Repeated K times

•••

genArrayConstructor

```
   t1 := 0
   tCnt_i := ...genExpr(countExpr_i)...
   if tCnt_i <= 0 goto runtimeError5
   t1 := t1 + tCnt_i
   tVal_i := ...genExpr(valueExpr_i)...
   xxxxxxxx
          •••
   xxxxxxxx
   t2 := t1*4
   t2 := t2 + 4
   t2 := alloc(t2)
   if t2 = 0 goto runtimeError1
   t3 := t2
   *t2 := t1
   t4 := tCnt_i
Label_i:
   t2 := t2 + 4
   *t2 := tVal_i
   t4 := t4 - 1
   if t4 > 0 goto Label_i
   xxxxxxxx
          •••
   xxxxxxxx
```

Repeated K times

Repeated K times

genArrayConstructor

```
    t1  := 0                              t1 is the total number of elts
    tCnt_i  := ...genExpr(countExpr_i)...
    if tCnt_i <= 0 goto runtimeError5
    t1 := t1 + tCnt_i                     Repeated K times
    tVal_i  := ...genExpr(valueExpr_i)...
    xxxxxxxx
          •••
    xxxxxxxx
    t2  := t1*4                           At this point, t1 = N
    t2  := t2 + 4
    t2  := alloc(t2)                      t2 is a running pointer
    if t2 = 0 goto runtimeError1
    t3  := t2                             t3 is a pointer to the array
    *t2  := t1
    t4  := tCnt_i                         t4 is loop counter
 Label_i:
    t2  := t2 + 4
    *t2  := tVal_i                        Repeated K times
    t4  := t4 - 1
    if t4 > 0 goto Label_i
    xxxxxxxx
          •••
    xxxxxxxx
```

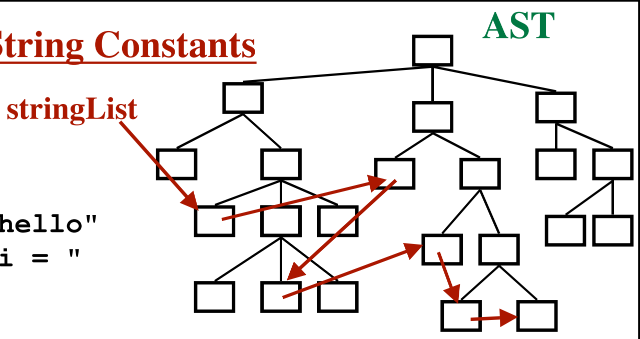
String Constants

Source:

```
   write ("hello");
   write ("i = ", i);
```

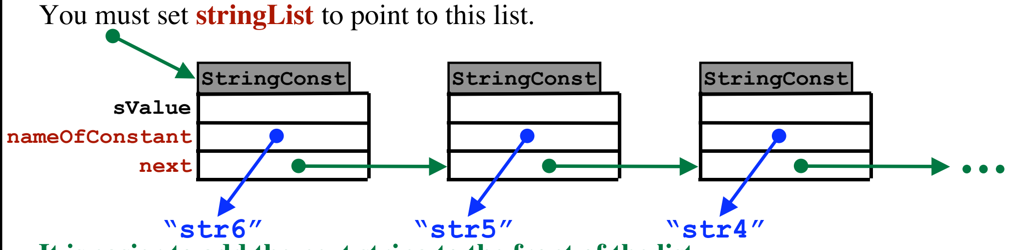
SPARC Target Code:

```
   str1:    .asciz    "hello"
   str2:    .asciz    "i = "
   ...
```



New fields: `StringConst.nameOfConstant` and `StringConst.next`

You must set **stringList** to point to this list.



It is easier to add the next string to the front of the list.

The strings will naturally print in reverse order.

```
str6:    .asciz    "i = "
str5:    .asciz    "hello"
str4:    .asciz    "abcd"
...
```

New Procedure Names

Source:

```
procedure foo1 (...) is
  procedure bar (...) is
procedure foo2 (...) is
  procedure bar (...) is
```

SPARC Target:

```
foo1:   save
        ...
bar:    save
        ...
foo2:   save
        ...
bar:    save
        ...
```

New Procedure Names

Source:

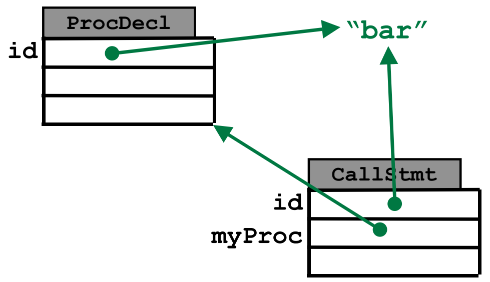
```
procedure foo1 (...) is
  procedure bar (...) is
procedure foo2 (...) is
  procedure bar (...) is
```

$$\begin{tikzcd}
\texttt{"bar"} \arrow[d] \\
\boxed{\texttt{newProcName()}} \arrow[d] \\
\texttt{"p17\_bar"}
\end{tikzcd}$$

SPARC Target:

```
foo1:   save
        ...
bar:    save
        ...
foo2:   save
        ...
bar:    save
        ...
```

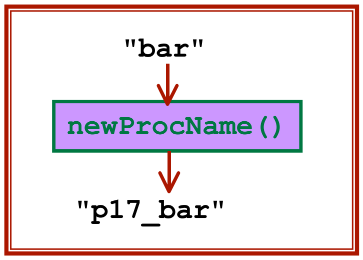
Just change "id":



New Procedure Names

Source:

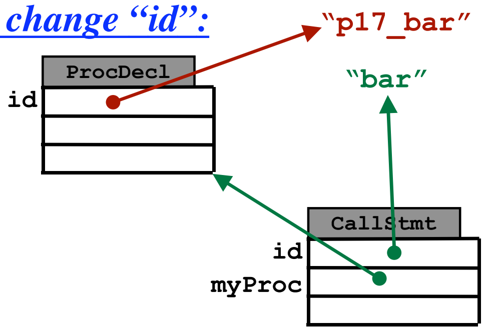
```
procedure foo1 (...) is
   procedure bar (...) is
procedure foo2 (...) is
   procedure bar (...) is
```



SPARC Target:

```
foo1:   save
        ...
bar:    save
        ...
foo2:   save
        ...
bar:    save
        ...
```

Just change "id":



New Procedure Names

SPARC Target:

```
p16_foo1:   save
            ...
p17_bar:    save
            ...
p18_foo2:   save
            ...
p19_bar:    save
            ...
```

For **IR.call**, we used a pointer to the **ProcDecl**

```
mov    %l3,%o1
call   p17_bar
nop
mov    %o0,%l4
```

Benefits:

User-defined names show through.
No conflicts with other assembly symbols:

```
main          float1
str1          float2
str2          ...
...           runtimeError1
Label_1       runtimeError2
Label_2       ...
...
p1_foo
p2_foo
...
```

Numerical Constants

Integers

IR Code: `x := x + 5`

SPARC:

```
ld    ...,%l4
add   %l4,5,%l4
st    %l4,...
```

```
ld    ...,%l4
set   5000,%l5
add   %l4,%l5,%l4
st    %l4,...
```

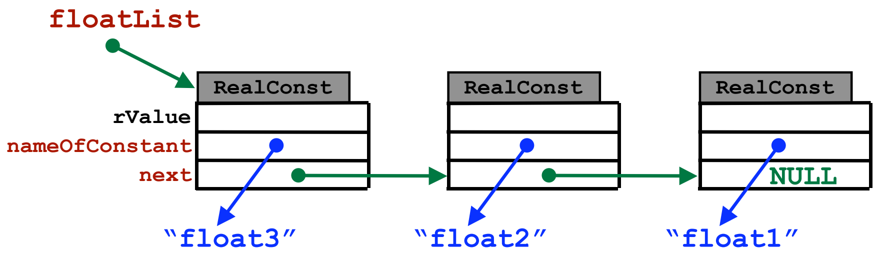
The value can be included as a literal in the instruction stream... No problem.

Floating Point Literals:

IR Code: `y := y + 5.67`

The value cannot be included as a literal... Must have a constant!

```
float4:  .single   0r5.67

         ...
         set       float4,%l5
         ldf       [%l5],%f0
         fadd      %f_,%f0,%f_
         ...
```



Must build this list.
(in reverse order)
Must give each RealConst a name.

```
String newFloatName () { ... }
```

Read and Write Statements

New IR instructions:

```
readInt x
readFloat x
writeInt y
writeFloat y
writeBoolean y
writeString s
writeNewLine
```

$\left.\texttt{readInt x},\ \texttt{readFloat x}\right\}$ Arg should contain an address

$\left.\texttt{writeInt y},\ \texttt{writeFloat y},\ \texttt{writeBoolean y}\right\}$ Arg should contain a value

`writeString s` ← Arg is nameOfConstant (e.g., "str5")

writeBoolean b

Will print either

`true`

or

`false`

depending on the value of "b"

Source:

$\texttt{read (}\underbrace{\texttt{i, j,}}_{\text{INT\_MODE}}\ \underbrace{\texttt{x, y}}_{\text{REAL\_MODE}}\texttt{);}$

These could be very complex L-Values, e.g., `r.a[i*foo(j)]`

IR Code:

genReadStmt

```
t3  := &i        genLValue → t3
readInt t3
t4  := &j        genLValue → t4
readInt t4
t5  := &x        genLValue → t5
readFloat t5
t6  := &y        genLValue → t6
readFloat t6
```

Source:

```
write ("Hello there");
write ("value=", i*j, ...);
```

INT_MODE

STRING_MODE

IR Code:

```
writeString str1
writeNewLine
```

genWriteStmt

```
writeString str2
t17 := i * j
writeInt t17
...
writeNewLine
```

genWriteStmt

genExpr

t17

Source:

```
write ("ans=", (xxx or xxx));
```

BOOLEAN_MODE

IR Code:

```
writeString str3
xxx
xxx
xxx
xxx
xxx
t18 := 1
writeBoolean t18
writeNewLine
```

genWriteStmt

genExpr

Short-Circuit Code

t18

Peephole Patterns

Pattern:

```
        goto  L
    L:
```

Action:

Delete the goto instruction (but keep the label)

Peephole Patterns

Pattern:

```
        goto  L
    L:
```

Action:

Delete the goto instruction (but keep the label)

Pattern:

```
        if ... goto  L1
        goto L2
    L1:
```

Action:

Replace with

```
        if not(...) goto  L2
    L1:
```

Negating Comparisons:

$= \rightarrow \neq$

$\neq \rightarrow =$

$< \rightarrow \geq$

$\leq \rightarrow >$

$> \rightarrow \leq$

$\geq \rightarrow <$

Peephole Patterns

Pattern:

```
        goto  L
    L:
```

Action:

Delete the goto instruction (but keep the label)

Pattern:

```
        if ... goto  L1
        goto L2
    L1:
```

Action:

Replace with

```
        if not(...) goto  L2
    L1:
```

Negating Comparisons:

$= \rightarrow \neq$

$\neq \rightarrow =$

$< \rightarrow \geq$

$\leq \rightarrow >$

$> \rightarrow \leq$

$\geq \rightarrow <$

Pattern:

```
        goto  L
        <anything except a label>
```

Action:

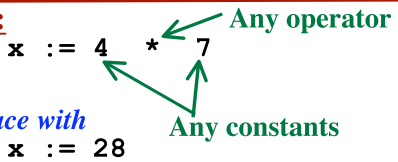
Delete the second instruction

Restart (without advancing)

to eliminate a series of instructions

Other Peephole Patterns

Pattern:



Action:

Replace with

```
        x := 28
```

Other Peephole Patterns

Pattern:

Any operator

```
x := 4   * 7
```

Any constants

Action:

Replace with

```
x := 28
```

Patterns:

```
x := z + 0
y := w * 1
a := b / 1.0
c := d - 0
```

Action:

Replace with

```
x := z
y := w
a := b
c := d
```

Other Peephole Patterns

Pattern:

Any operator

```
x := 4   * 7
```

Any constants

Action:

Replace with

```
x := 28
```

Patterns:

```
x := z + 0
y := w * 1
a := b / 1.0
c := d - 0
```

Action:

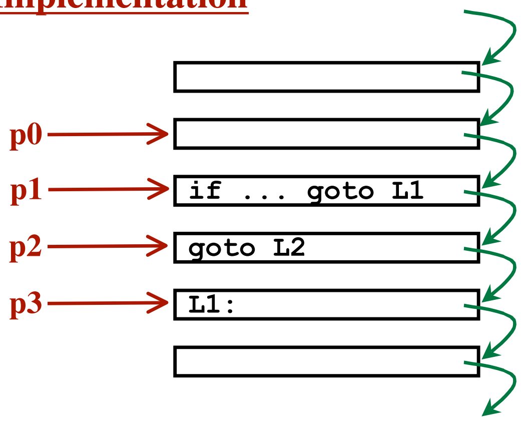
Replace with

```
x := z
y := w
a := b
c := d
```

Other Patterns:

```
x := 0 + z
y := 0 * w
y := w * 0
e := 0 - f
```

Ideas for Implementation



Look for pattern starting at p1.
If found...
    Modify list of instructions;
        Repeat without incrementing
If not found
    Increment all pointers.

What to hand in, if you do PEEPHOLE?
- Email Peephole.java
- Turn in a short write-up
  ...with an annotated output listing

Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
    ...Then must reduce the "count"

If a LABEL's count goes to zero...
    Delete it!
This may make some instructions
    unreachable.

Make repeated passes over the IR
    instruction list, until a pass is made in
        which no instructions are eliminated.
[This process must terminate.... Why?]

Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the "count"

If a LABEL's count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.
[This process must terminate.... Why?]

Example

Count = 0

Count = 1

```
            xxxx
            xxxx
            goto      L4
L3:
            yyyy
            yyyy
            yyyy
            goto      L5
L4:
            xxxx
            xxxx
```

Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the "count"

If a LABEL's count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.
[This process must terminate.... Why?]

Example

Count = 1

```
            xxxx
            xxxx
            goto      L4

            yyyy
            yyyy
            yyyy
            goto      L5
L4:
            xxxx
            xxxx
```

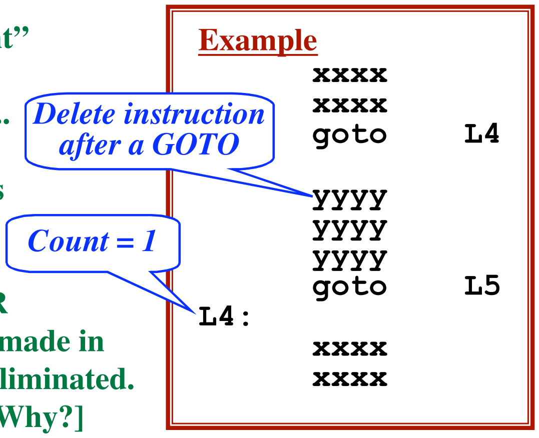
Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the "count"

If a LABEL's count goes to zero...
Delete it!
This may make some instructions
unreachable.

Make repeated passes over the IR
instruction list, until a pass is made in
which no instructions are eliminated.
[This process must terminate.... Why?]



```
        xxxx
        xxxx
        goto      L4

        yyyy
        yyyy
        yyyy
        goto      L5
L4:
        xxxx
        xxxx
```

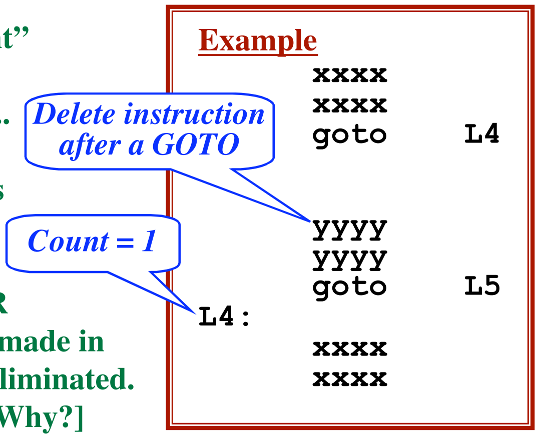
Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the "count"

If a LABEL's count goes to zero...
Delete it!
This may make some instructions
unreachable.

Make repeated passes over the IR
instruction list, until a pass is made in
which no instructions are eliminated.
[This process must terminate.... Why?]



```
        xxxx
        xxxx
        goto      L4


        yyyy
        yyyy
        goto      L5
L4:
        xxxx
        xxxx
```

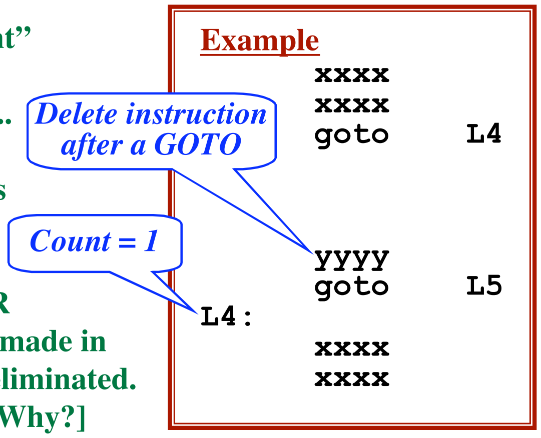
Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the "count"

If a LABEL's count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.
[This process must terminate.... Why?]



```
        xxxx
        xxxx
        goto    L4

        yyyy
        goto    L5
L4:
        xxxx
        xxxx
```

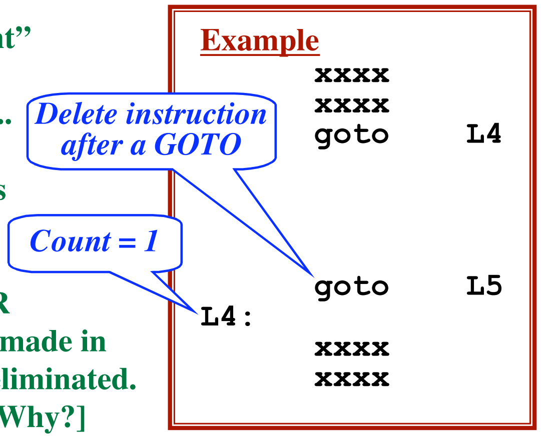
Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the "count"

If a LABEL's count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.
[This process must terminate.... Why?]



```
        xxxx
        xxxx
        goto    L4


        goto    L5
L4:
        xxxx
        xxxx
```

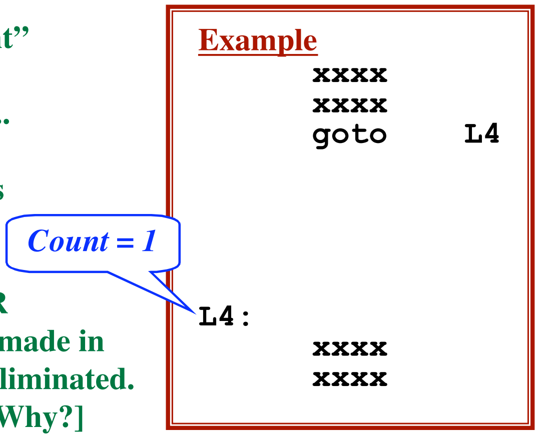
Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the "count"

If a LABEL's count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.
[This process must terminate.... Why?]


Example
```
          xxxx
          xxxx
          goto     L4



L4:
          xxxx
          xxxx
```

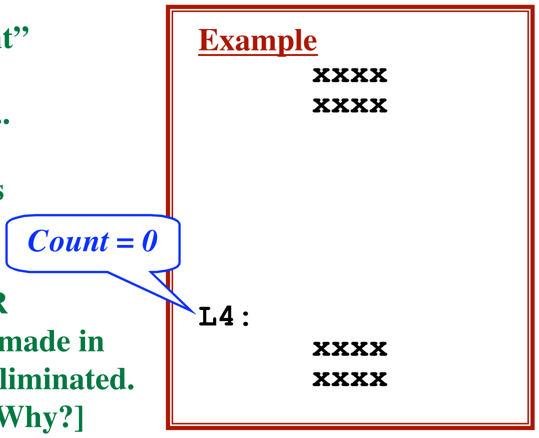
Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the "count"

If a LABEL's count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.
[This process must terminate.... Why?]


Example
```
          xxxx
          xxxx




L4:
          xxxx
          xxxx
```

Another Peephole Idea

Associate a "count" with each LABEL instruction.
Keep track of how many GOTOs branch to that label.

Sometimes we eliminate GOTOs
...Then must reduce the "count"

If a LABEL's count goes to zero...
Delete it!
This may make some instructions unreachable.

Make repeated passes over the IR instruction list, until a pass is made in which no instructions are eliminated.
[This process must terminate.... Why?]

Example

```
        xxxx
        xxxx



        xxxx
        xxxx
```