Lecture 26:

- Code Generation

Announcements:

- HW-5 due Friday
- Proj Part 2 due Mon (examples, test cases, initial design/plan)

IR Code Generation

(6) While statements

- grab starting index of first instruction (to jump back to)
- generate while condition expression (via `accept()`)
- add a `JMPF` instruction with temp operand (don't know end yet)
- visit all of the statements (with new environment)
- add a `JMP` instruction (to starting index)
- update the `JMPF` instruction (to instruction index *after* `JMP`)

```
void f(x: int) {
  var i = 0
  while i < 10 {
    i = i + x
  }
}
```

```
...
 3: LOAD(1)   // load var i
 4: PUSH(10)
 5: CMPLT()
 6: JMPF(12)  // while loop
 7: LOAD(1)   // load var i
 8: LOAD(0)   // load var x
 9: ADD()
10: STORE(1)  // store i
11: JMP(3)
...
```

IR Code Generation

(7) For Loops

- similar to while, but ...
- push environment first for loop var (and do initialization)
- update loop var before `JMP`

```
void main() {
  var x = 4
  for i from 0 to 3 {
    x = x + i
  }
}
```

```
...
 2: PUSH(0)
 3: STORE(1)   // store i
 4: LOAD(1)    // load var i
 5: PUSH(3)
 6: CMPLE()    // i <= 3
 7: JMPF(17)   // for loop
 8: LOAD(0)    // load var x
 9: LOAD(1)    // load var i
10: ADD()      // x + i
11: STORE(0)   // store x
12: LOAD(1)    // load var i
13: PUSH(1)
14: ADD()      // i + 1
15: STORE(1)   // store i
16: JMP(4)
...
```


IR Code Generation

(8) Object creation – structs

- add an `ALLOCS` instruction
- initialize the corresponding fields:
- ... get the field information from the `StructDef`
- ... use `SETF` to set the object fields

```
struct T {
  x: int,
  y: bool
}

void main() {
  var t = new T(3, true)
}
```

```
0: ALLOCS()  // new T
1: DUP()
2: PUSH(3)
3: SETF(x)
4: DUP()
5: PUSH(true)
6: SETF(y)
7: STORE(0)  // t
...
```

(*) `SETF`(A): pop value x, pop oid, set obj(oid).$A = x$

IR Code Generation

(9) Array object creation

- add an `ALLOCA` instruction
- requires a size (where array is filled with size `null`'s)

```
void main() {
  var xs = new int[10]
}
```

```
0: PUSH(10)
1: ALLOCA()
2: STORE(0)  // xs
...
```


IR Code Generation

(10) Assigning values to array elements

- setup and then add a `SETI` instruction

```
void main() {
  var xs = new int[10]
  xs[0] = 42
}
```

```
...
3: LOAD(0)      // xs (oid)
4: PUSH(0)      // the index
5: PUSH(42)     // the value
6: SETI()
...
```

(*) `SETI()`: pop value v, pop index i, pop oid, set obj(oid)[i] = v

IR Code Generation

(11) General rvalue path and lvalue expressions

- load the variable value (e.g., the `p` in `p.x.y.z`)
- repeatedly add a `GETF` instruction for remaining path (e.g., `x`, `y`, and `z`)
- for array access, generate index code and use `GETI`
- for assignment statements, last instruction is a `SETF` or `SETI`

(a) Simple example of non-array rvalue

```
struct Node {
  val: int,
  next: Node
}

void main() {
  var p = new Node(3, null)
  var x = p.val
}
```

```
 0: ALLOCS()     // new Node
 1: DUP()
 2: PUSH(3)
 3: SETF(val)
 4: DUP()
 5: PUSH(null)
 6: SETF(next)
 7: STORE(0)     // p
 8: LOAD(0)
 9: GETF(val)
10: STORE(1)    // x
...
```


IR Code Generation

(b) Simple example of lvalue ...

```
struct Node {
  var: int,
  next: Node
}

void main() {
  var p = new Node(3, null);
  // circular linked list!
  p.next = new Node(0, p)
  p.next.val = 4
}
```

```
 0: ALLOCS()    // new Node
 1: DUP()
 2: PUSH(3)
 3: SETF(val)
 4: DUP()
 5: PUSH(null)
 6: SETF(next)
 7: STORE(0)    // p
 8: LOAD(0)     // for p.next
 9: ALLOCS()    // new Node
10: DUP()
11: PUSH(0)
12: SETF(val)
13: DUP()
14: LOAD(0)
15: SETF(next)
16: SETF(next)  // sets p.next
17: LOAD(0)
18: GETF(next)  // for p.next.val
19: PUSH(4)
20: SETF(val)   // sets p.next.val
```

IR Code Generation

Left as an exercise:

- expressions (evaluated left to right; except for >= and >)
- if statements (similar to loops, but more jumps to keep track of)
- array access (similar to field access, but use `GETI`, `SETI`, gen index code)