Lecture 28:

- IR code generation (cont)

Announcements:

- HW-5 due
- HW-6 out

(5) Simple rvalues (i.e., literals)

- convert each lexeme to its associated Python type
- push the value onto the current frame's operand stack

```
    def visit_simple_rvalue(self, simple_rvalue):
        val = simple_rvalue.value.lexeme
        if simple_rvalue.value.token_type == TokenType.INT_VAL:
            self.add_instr(PUSH(int(val)))
        elif simple_rvalue.value.token_type == TokenType.DOUBLE_VAL:
            self.add_instr(PUSH(float(val)))
        elif simple_rvalue.value.token_type == TokenType.STRING_VAL:
            val = val.replace('\\n', '\n')
            val = val.replace('\\t', '\t')
            self.add_instr(PUSH(val))
        elif val == 'true':
            self.add_instr(PUSH(True))
        elif val == 'false':
            self.add_instr(PUSH(False))
        elif val == 'null':
            self.add_instr(PUSH(None))
```

(6) While statements

- grab the starting index of the first instruction (to jump back to)
- call the while condition visitor
- create and add a JMPF instruction with temp operand -1 (jump to end)
- push an environment in var table
- visit all of the statements
- pop an environment in var table
- add a JMP instruction (to jump to starting index)
- add a NOP instruction (for JMPF to refer to)
- update the JMPF instruction to refer to the NOP

Simple while example

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

```
Frame 'f'
   0: STORE(0)  // parameter x
   1: PUSH(0)
   2: STORE(1)  // store i
   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)
  12: NOP()
  13: PUSH(null)
  14: RET()
```

(7) For Loops

... similar to while loops except:

- push an environment first for var decl (and generate var decl code)
- rest similar as while (within another pushed and popped env)
- then add assignment statement code before final JMP
- pop the environment for var decl
- add the JMP, NOP, and update the JMPF

Simple while example

```
void main() {
  int x = 0;
  for (int i = 0; i < 5; i = i + 1) {
    x = x + i;
  }
}
```

```
Frame 'main'
   0: PUSH(0)
   1: STORE(0)   // store x
   2: PUSH(0)
   3: STORE(1)   // store i
   4: LOAD(1)    // load var i
   5: PUSH(10)
   6: CMPLT()
   7: JMPF(17)   // for loop
   8: LOAD(0)    // load var x
   9: LOAD(1)    // load var i
  10: ADD()
  11: STORE(0)   // store x
  12: LOAD(1)    // load var i
  13: PUSH(1)
  14: ADD()
  15: STORE(1)   // store i
  16: JMP(4)
  17: NOP()
  18: PUSH(null)
  19: RET()
```

(8) Object creation – structs

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

Simple example ...

```
struct T {
  int x;
  bool y;
}

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

```
Frame main
   0: ALLOCS()  // new T
   1: DUP()
   2: PUSH(3)
   3: SETF(x)
   4: DUP()
   5: PUSH(True)
   6: SETF(y)
   7: STORE(0)  // t
   8: PUSH(None)
   9: RET()
```

(9) Array object creation

- create and add an **ALLOCA** instruction
- which requires a size and initial value (which will be null)

Simple example ...

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

```
Frame main
   0: PUSH(10)
   1: ALLOCA()
   2: STORE(0)  // xs
   3: PUSH(None)
   4: RET()
```

Simple example of array value assignment (via heap object):

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

```
Frame main
   0: PUSH(10)
   1: ALLOCA()
   2: STORE(0)  // xs
   3: LOAD(0)
   4: PUSH(0)
   5: PUSH(42)
   6: SETI()
   7: PUSH(None)
   8: RET()
```