Compilation Methods SS 2013 - Assignment 1

Kastens, Pfahler, 17.04.2013

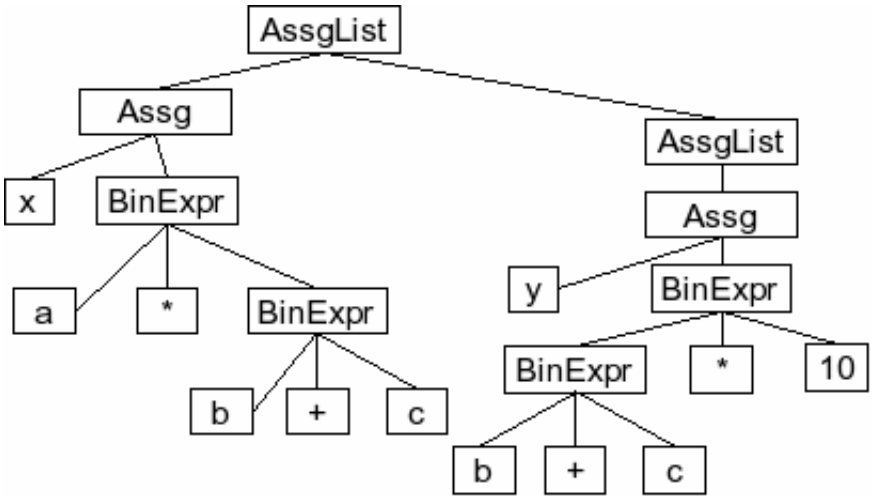
Exercise 1 (Different forms of intermediate code)

The following intermediate code fragments are given in 2-address-form (Team 1), 0-address-form/stack-form (Team 2), and tree representation (Team 3).

Note: Both instruction sequences initialize the variables a, b, and c. Those assignments are omitted in the tree representation.

```
mov local-4, 10 # a
mov local-8, 20 # b,
mov local-12, -30 # c,
mov %eax, local-12 # c, c
add %eax, local-8 # tmp59, b
imul %eax, local-4 # tmp61, a
mov local-16, %eax # x, tmp61
mov %eax, local-12 # c, c
mov %edx, local-8 # tmp62, b
add %edx, %eax # tmp62, c
mov %eax, %edx # tmp64, tmp62
sal %eax, 2  # tmp64,
add %eax, %edx # tmp64, tmp62
sal %eax  # tmp65
mov local-20, %eax # y, tmp65
```

```
0: bipush 10
2: istore_0
3: bipush 20
5: istore_1
6: bipush -30
8: istore_2
9: iload_0
10: iload_1
11: iload_2
12: iadd
13: imul
14: istore_3
15: iload_1
16: iload_2
17: iadd
18: bipush 10
20: imul
21: istore_4
```



Reconstruct a sequence of Java (or C) assignments that could be compiled to this intermediate code and prepare to explain the relationship to the other teams. Do you find optimization opportunities (Slide 202) in your intermediate code representation?

Exercise 2 (Translating statements to intermediate code)

Convert the following assignment statement to intermediate code in 0-address-form, 2-address-form, and an abstract syntax tree-representation. Assume that all variables are declared with type `int`.

```
c = (a + b) * (a + b) - 1;
```

The subexpression `a+b` appears twice. How could you avoid duplicate computation of the sum in each of the three forms of intermediate code?

Exercise 3 (Optimizations of Java Bytecode)

Which optimizations of Slide 202 are applied by the Java compiler? Which optimizations could have been applied additionally?

```
public class Optimization {

    public static int deadVariables() {
        int a = 50;
        int b = 60;

        int x = a + b;
        x = 5;

        return x;
    }

    public static int algebraicSimplification() {
        int p = 50;
        double i = 2 * 3.14;
        int j = p + 0;
        int k = p * 2;
        return j + k;
    }

    public static boolean bool;
    public static int constantPropagation() {
        int x = 2;
        if (bool) {
           int z = 42;
        }
        int y = x * 5;

        return y;
    }

    public static int copyPropagation() {
        int p = 40;

        int x = p;
        int z = x;

        return z;
    }
}
```

```
public static int deadVariables();
   0:       bipush  50
   2:       istore_0
   3:       bipush  60
   5:       istore_1
   6:       iload_0
   7:       iload_1
   8:       iadd
   9:       istore_2
   10:      iconst_5
   11:      istore_2
   12:      iload_2
   13:      ireturn

public static int algebraicSimplification();
   0:       bipush  50
   2:       istore_0
   3:       ldc2_w  #2; //double 6.28d
   6:       dstore_1
   7:       iload_0
   8:       iconst_0
   9:       iadd
   10:      istore_3
   11:      iload_0
   12:      iconst_2
   13:      imul
   14:      istore  4
   16:      iload_3
   17:      iload   4
   19:      iadd
   20:      ireturn

public static int constantPropagation();
   0:       iconst_2
   1:       istore_0
   2:       getstatic       #4; //bool
   5:       ifeq    11
   8:       bipush  42
   10:      istore_1
   11:      iload_0
   12:      iconst_5
   13:      imul
   14:      istore_1
   15:      iload_1
   16:      ireturn

public static int copyPropagation();
   0:       bipush  40
   2:       istore_0
   3:       iload_0
   4:       istore_1
   5:       iload_1
   6:       istore_2
   7:       iload_2
   8:       ireturn
```

Exercise 4 (HOMEWORK: Manually modifying Java bytecode)

The Java classfile `CountDown.class` contains an important Java program that has been developed for upcoming NASA Mars missions. Unfortunately the source code has been lost. All that is left is a bytecode listing of class `CountDown` in file `CountDown.j`. This file has been generated from the the classfile.

- Use the Java interpreter to execute the supplied classfile. What is wrong with the program (from the NASA's point of view)?
- Modify the assembler source code in file `CountDown.j` so that the countdown works as expected.

  Use the command `~compiler/bin/Jasmin CountDown.j` to assemble a new classfile, when you have fixed the assembler source code. Invoke the resulting classfile with the bytecode verifier enabled:
  `java -verify CountDown`

Hints: You can find an overview on Java Bytecode instructions at
`http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings`.