EECS665

Compiler Construction

Drew Davidson
Ruturaj Vaidya

Lecture: LEEP2 G415
MWF 3:00-3:50

Lab: Eaton 1005B

ANOUNCEMENTS | LAB | SCHEDULE | MATERIALS | ASSIGNMENTS

Homework 7

Due on October 31st @ 3:00 PM (in class, to Drew, or at Engineering front office)

Not accepted late

ALL homework must be done individually

Question 1

We've covered Abstact-Syntax Trees (ASTs), Three-Address Code (3AC), and Control-Flow Graphs (CFGs), as intermediate representations of a source program. Why might a compiler use all three of these IRs in sequence? What is a benefit of each of these IRs?

Question 2

Consider the following function:

```
int main(){
```

```
                int b;
                int c;
                b = 4;
                c = 2;
                if (b > 2){
                        while (b > 0){
                                b = b - 1;
                        }
                }
                b = mystery (b + c + 1);
}
```

Part I

Draw the AST fragment corresponding to the above function definition. You may use the AST node types from Lil' C, as well as any other node types you might need. If you do need another node type, please explain what it does. I don't think you'll need any other node types, though.

Part II

Draw a 3AC representation of the above function definition. You may use the following 3AC instructions:

Assignment

- x = y op z
  (where op is a logical or mathematical operator)
- x = y

Jumps

- ifz (x) goto L
- goto L

Functions

- call p
- retrieve x
- return x
- enter p
- get_arg x, k
- set_arg x, k
- leave p

Labeling

- label L
  (You may annotate any instruction with a label followed by a colon)

If you feel that you need to use any other instructions, feel free to do so as long as you explain

what they do any they fit the 3AC constraints. Again, I don't think you'll actually need any other instructions, but go hog wild.

Part III

Draw a Control-Flow Graph of the above function definition. Your basic blocks should contain your 3AC code that you defined in Part II.