PPL 4/8

Reagan Shirk

April 8, 2020

Program Execution in our Interpreter

- The whole point of control flow is to manipulate the program execution to create different behaviors
- I told myself I would try really hard to pay attention in lecture today and then he went backwards in slides so I started playing guitar oops
  - I stopped playing guitar and thought "I'll pay attention now" and started playing again within minutes
- Okay, I'm gonan really try this time

WHILE Codegen

```
while <- WHILE
    { $$ = next_instruction; }
    condition
    { $$ = gencode(JZ, $3, N/A, TBDA); }
    body
    {
        gencode(JMP, N/A, N/A, $2);
        itable[$4].a3 = next_instruction;
    }
```

- Using `last_instruction` instead of `next_instruction` in the last statement of the code above would create an infinite loop
- The target of the jump instruction is "forward", so we need to temporarily store the instruction that will be completed later
- You can have an unconditional jump to reevaluate the condition. . . ? Hopefully he says more about this

IF Codegen

```
if <- IF '(' expr ')' THEN
    { $$ = gencode(JZ, expr, N/A, TBDA); }
    body
    {
        $$ = gencode(JMP, N/A, N/A, TBDA);
        itable[$6].a3 = next_instruction;
    }
    elsebranch
    { itable[$8].a3 = next_instruction; }
  ;
elsebranch <- ELSE body
  | epsilon
  ;
```

- You will skip the true branch if the condition is false and vice versa if the condition is true

- The target of both jump instructions is “forward” so we need to temporarily store the instruction entry that will be completed later
- Why is it `$6` in the `itable` index?
  - `IF = $1`
  - `'(' = $2`
  - `expr = $3`
  - `')' = $4`
  - `THEN = $5`
  - What is in the first set of brackets is `$6`
- Parameters in the `gencode` function:
  - first = some operation code
    - comparison, math, etc
  - the rest are `a1, a2, a3` and depend on the particular operation, it’s in slide 7 of this slide set (not sure which slide set we’re on though sorry guys)

Structured and Unstructured Flow

- The target of a GOTO statement can be a line number
  - You have to define a label somewhere to use GOTO, not really sure what that means
  - Ohhh you have to put a label where you want to go and then your GOTO statement says goto LABEL
- The break command stops execution in a single loop construct in C/C++ (which I think we all knew)
- The continue command skips the remaining instructions in a loop and proceeds with evaluating the next condition/iteration
- Multi-level returns (MLR) are a construct that allows exiting of several function calls