Example Pseudocode to Quads/Symbol Table Translation
SPRING 2022

The following is an example of the conversion of a simple pseudocode/Java-like iterative factorial function into the project QuadTable and SymbolTable code which can be executed by the Interpreter class.

Pseudocode for a factorial function, with line numbers for easy reference later:

```
1    factorial(int n) {
2    //Iterative calculation, prints the result
3    int i, product;
4    product = 1;
5    for (i=1; i<= n; ) { //no increment in the loop criteria
6       product = product * i;
7       i = i + 1;        //increments here instead
8    }
9    print(product);
```

The first steps to translating this code is to scan it for each instance of a variable or a constant, and add them to the SymbolTable so that the operands for the Quad codes can be filled in as the Quad instructions are added.

Symbol Table

Index	Symbol name	Symbol kind	IntegerValue
0	n	v	10 (initialized like parameter)
1	i	v	0
2	product	v	0
3	1	c	1
4	$temp	v	0

A couple of explanations:

1) for this example, the variable ***n*** is initialized to 10 as if it were being sent in as a parameter, which saves a Quad MOV instruction when creating the Quads.

2) the variable ***$temp*** is shown above, but the need for it is not evident until the Quad for implementing the loop condition check is being added. It is shown above for convenience.

The next step in creating the translation is the creation of the set of Quad instructions which will implement each line and logic construct of the pseudocode. The Quads are shown, with the pseudocode line number they are representing:

Quad Table

Index	Instruction	Opcode	Op1	Op2	Op3	Pseudo Line #
0	MOV	5	3	0	2	(4) prod = 1
1	MOV	5	3	0	1	(5) i = 1
2	SUB	3	1	0	4	(5) loop compare, $temp = i - n
3	JP	10	4	0	7	if temp >= 0, branch to Quad line 7
4	MUL	2	2	1	2	(6) product = product * i
5	ADD	4	1	3	1	(7) i = i+1
6	JMP	8	0	0	2	(loop end) jump to Quad line 2
7	PRINT	6	2	0	0	(9) print product

Comments:

1) The loop condition check is done in a standardized way, at Quad line 2, by subtracting the right side of the relational operator from the left side, so **i <= n** leads to subtracting **n** from **i**. A temporary storage location is needed to hold the result of this operation, resulting in the creation of **$temp** in the Symbol Table at slot 4.

2) The check of the loop exit condition depends on the relational operator involved. In short, checking <= requires exit from the loop when the result of the subtraction results in a value > 0, which corresponds to the **JP (Branch on Positive)** instruction at Quad line 3. The value of Symbol Table item #4 is checked, and if it is positive, a branch to Quad line 7 will occur.

3) The unconditional branch, **JMP**, sends execution back to Quad line 2 to repeat the condition check code and exit the loop based on the condition result.