1 Switch Statement Code Generation

The biggest challenge here is remembering which label the case statements need to jump to when they finish execution. We do this by augmenting the tree with a parentLabelId field on each node.

```c
void genCode(tree t)
...
if(t.rule is "statement -> SWITCH LPAREN expr RPAREN LBRACE cases default RBRACE"){
    x = genLabelID()
    // Evaluate the expr and push it onto the stack
    genCode(t.children[2])
    push($3) // put expr onto to compare with each case
    // Pass the label ID we generated to the children
    c.children[5].parentLabelId = x
    // Generate all the case statements
    genCode(t.children[5])
    // Generate code for the default case
    genCode(t.children[6])
    endSwitch + x:
}
if(t.rule is "cases -> cases case"){
    // Generate the code for the case statements
    // Pass on the parentLabelId
    t.children[0].parentLabelId = t.parentLabelId
    genCode(t.children[0])
    t.children[1].parentLabelId = t.parentLabelId
    genCode(t.children[1])
}
if(t.rule is "cases -> "){
}
if(t.rule is "case -> CASE LPAREN expr RPAREN LBRACE statements RBRACE"){
    x = genLabelID()
    // Evaluate the expr
    genCode(t.children[2])
    // Pop the switch statement expression from the stack
    pop($5)
    // Compare the case statement and switch statement expressions
    bne $3, $5, endLabel + x
    genCode(t.children[5])
    jr endSwitch + t.parentLabelId
```
As an exercise, reduce the number of pushes and pops required to generate a switch statement. You will still want to push the expression’s result onto the stack at the beginning of the switch (why?).

2 Extending WLP4: Pre- and Post-Increment

1. If lvalue is an int, then so is factor. If it is an int*, then so is factor.

2. We need to modify the line which says add $5, $5, $11 to say:

   if (typeOf(lvalue) == int*) {
     add $5, $5, $4
   } else {
     add $5, $5, $11
   }

3 Extending WLP4: Pointers as Conditions

1. Recall that all of our conditions are under the test nonterminal, so we should add our new rule there:

   test → expr

2. We need to verify that expr has type int* for the new rule to be well-typed.

3. We want to put 1 in $3 if the condition is true, and 0 otherwise. Note that different sections may use different values for NULL. Here assume that it is 0, but it should be easy to swap out.

   genCode(test expr) {
     genCode(expr)
     add $5, $3, $0 ;Back up the value of expr
     lis $6
     .word 0 ;Our value for NULL
     add $3, $0, $0
     beq $5, $6, 1
     add $3, $11, $0
   }