Computer Science 301

Programming Language Translation

Practical 8: Solution

The sources of full solutions for these problems may be found on the course web page as the file PRAC8SOL.ZIP.

Task 4 Repeat discussions with your team mates until you all get the idea

They don't get much easier than this.

```
    RepeatStatement<StackFrame frame>  (. Label loopStart = new Label(known); .)
    =  "repeat" {
         Statement<frame>
       }
       WEAK "until"
       "(" Condition ")" WEAK ";"      (. CodeGen.BranchFalse(loopStart); .) .
```

Task 5 You had better do this one or else....

The problem, firstly, asked for the addition of an *else* option to the *IfStatement*. Adding an *else* option to the *IfStatement* efficiently is easy once you see the trick. Note the use of the "no else part" option associated with an action, even in the absence of any terminals or non-terminals. This is a very useful technique to remember.

```
    IfStatement<StackFrame frame>      (. Label falseLabel = new Label(!known);
                                          Label outLabel = new Label(!known); .)
    =  "if" "(" Condition ")"          (. CodeGen.BranchFalse(falseLabel); .)
 **    [ "then"                     (. SemError("then is not used in Parva"); .)
       ]   Statement<frame>
 **    (   "else"                           (. CodeGen.Branch(outLabel);
 **                                            falseLabel.Here(); .)
 **        Statement<frame>                 (. outLabel.Here(); .)
 **      | /* no else part */               (. falseLabel.Here(); .)
 **    ) .
```

In the past students attempting this problem have come up with the following sort of code instead. This can generate BRN instructions where none are needed.

```
    IfStatement<StackFrame frame>           (. Label falseLabel = new
Label(!known);
                                               Label outLabel = new
Label(!known); .)
    =  "if" "(" Condition ")"               (. CodeGen.BranchFalse(falseLabel);
.)
       [ "then"                             (. SemError("then is not used in
Parva"); .)
       ]  Statement<frame>                  (. CodeGen.Branch(outLabel);
                                               falseLabel.Here(); .)
       [ "else"  Statement<frame>  ]        (. outLabel.Here(); .) .
```

Using this strategy, source code like

```
if (i == 12) k = 56;
```

would lead to object code like

```
        12    LDA  0
        14    LDV
        15    LDC  12
        17    CEQ
        18    BZE  27
        20    LDA  5
        22    LDC  56
        24    STO
        25    BRN  27        // unnecessary
        27    ....
```

Although not asked for, we can handle *elsif* clauses with the same idea. Note that. after defining `falseLabel.Here()`, the label is "re-used" by assigning it another instance of an "unknown" label. If you don't do this you will get all sorts of bad code or funny messages from the label handler!

```
    IfStatement<StackFrame frame)       (. Label falseLabel = new Label(!known);
                                           Label outLabel = new Label(!known);
.)
    =  "if" "(" Condition ")"           (. CodeGen.BranchFalse(falseLabel); .)
       [ "then"                         (. SemError("then is not used in
Parva"); .)
       ]  Statement<frame)
 **     {                               (. CodeGen.Branch(outLabel);
 **                                        falseLabel.Here();
 **                                        falseLabel = new Label(!known); .)
 **       "elsif" "(" Condition ")"     (. CodeGen.BranchFalse(falseLabel); .)
 **       [ "then"                      (. SemError("then is not used in
Parva"); .)
 **       ] Statement<frame>
 **     }
       (  "else"                        (. CodeGen.Branch(outLabel);
                                           falseLabel.Here(); .)
           Statement<frame
         | /* no else part */           (. falseLabel.Here(); .)
       )                                (. outLabel.Here(); .) .
```

Task 7 This has gone on long enough - time for a break (and then continue)

Although I asked only for the *break* statement, this solution illustrates the less common *continue* statement as well, for those who like reading interesting, clear, code in a compiler...

The syntax of the *BreakStatement* and *ContinueStatement* is, of course, trivial. The catch is that one has to allow these statements only in the context of loops. Trying to find a context-free grammar with this restriction is not worth the effort.

A common approach to incorporating context-sensitive checking in conjunction with code generation, as hopefully you know, is based on passing information as parameters between subparsers. The pieces of information we need to pass here are `Label` objects. We change the parser for *Statement* and for *Block* as follows:

```
 **  Block<StackFrame frame, Label breakLabel, Label continueLabel>
     =                                      (. Table.openScope(); .)
 **     "{" { Statement<frame, breakLabel, continueLabel>
            }
         WEAK "}"                           (. Table.closeScope(); .) .

 **  Statement<StackFrame frame, Label breakLabel, Label continueLabel>
 **  =  SYNC (   Block<frame, breakLabel, continueLabel>
               | ConstDeclarations
               | VarDeclarations<frame>
               | AssignmentOrCall
               | IncDecStatement
 **            | IfStatement<frame, breakLabel, continueLabel>
               | WhileStatement<frame>
               | DoWhileStatement<frame>
               | RepeatStatement<frame>
 **            | BreakStatement<breakLabel>
 **            | ContinueStatement<continueLabel>
               | HaltStatement
               | ReturnStatement
               | ReadStatement
               | WriteStatement
               | ";"                        (. if (warnings) Warning("empty
statement"); .)
             ) .
```

The very first call to *Statement* passes `null` as the value for each of these labels:

```
     Body<StackFrame frame>               (. Label DSPLabel = new Label(known);
                                             int sizeMark = frame.size;
                                             CodeGen.OpenStackFrame(0); .)
 **  =  "{" { Statement<frame, null, null> }
        WEAK "}"                          (. CodeGen.FixDSP(DSPLabel.Address(),
                                               frame.size - sizeMark);
                                             CodeGen.LeaveVoidFunction(); .) .
```

The parsers for the statements that are concerned with looping, breaking, and making decisions become

```
    IfStatement<StackFrame frame, Label breakLabel, Label continueLabel>
                                            (. Label falseLabel =
                                                      new Label(!known);
                                               Label outLabel =
                                                      new Label(!known); .)
    =  "if" "(" Condition ")"               (. CodeGen.BranchFalse(falseLabel);
.)
 **       [ "then"                  (. SemError("then is not used in Parva"); .)
 **     ] Statement<frame, breakLabel, continueLabel>
       {                                    (. CodeGen.Branch(outLabel);
                                               falseLabel.Here();
```

```
                                                 falseLabel = new Label(!known);
.)
         "elsif" "(" Condition ")"            (. CodeGen.BranchFalse(falseLabel);
.)
 **       [ "then"                    (. SemError("then is not used in Parva"); .)
 **       ] Statement<frame, breakLabel, continueLabel>
      }
      (  "else"                               (. CodeGen.Branch(outLabel);
                                                 falseLabel.Here(); .)
 **        Statement<frame, breakLabel, continueLabel>
         | /* no else part */                 (. falseLabel.Here(); .)
      )                                       (. outLabel.Here(); .) .

   WhileStatement<StackFrame frame>           (. Label loopExit  = new
                                                        Label(!known);
 **                                              Label loopContinue = new
                                                        Label(known); .)
   =  "while" "(" Condition ")"               (. CodeGen.BranchFalse(loopExit); .)
 **   Statement<frame, loopExit, loopContinue>
                                              (. CodeGen.Branch(loopContinue);
 **                                              loopExit.Here(); .) .


** RepeatStatement<StackFrame frame>       (. Label loopExit = new Label(!known);
 **                                           Label loopContinue =
                                                             new Label(!known);
                                              Label loopStart = new Label(known);
.)
   =  "repeat" {
 **      Statement<frame, loopExit, loopContinue>
         }
      WEAK "until"                            (. loopContinue.Here(); .)
      "(" Condition ")" WEAK ";"              (. CodeGen.BranchFalse(loopStart);
 **                                              loopExit.Here(); .) .

   BreakStatement<Label breakLabel>
 ** =  "break"                              (. if (breakLabel == null)
 **                                        SemError("break is not allowed here");
 **                                          else CodeGen.Branch(breakLabel); .)
      WEAK ";" .

   ContinueStatement<Label continueLabel>
 ** =  "continue"                           (. if (continueLabel == null)
 **                                  SemError("continue is not allowed here");
 **                                          else CodeGen.Branch(continueLabel);
.)
      WEAK ";" .
```

There is at least one other way of solving the problem, which involves using local variables in the parsing methods to "stack" up the old labels, assigning new ones, and then restoring the old ones afterwards. But the method just presented seems the neatest.