Language Processors Lab 8 (Week 9)
Extending SIMPLE: Adding the REPEAT statement

Last week we extended the SIMPLE Programming Language by adding the power operator. This week, we will continue with the exercise left at the end of last week's lab and add the REPEAT statement to the language.

The files and environment

Start a Unix shell window and move to your LanguageProcessors directory. Download the file lab8.tar.gz from Moodle or from http://www.soi.city.ac.uk/~sbbc287/lab8.tar.gz. Unzip and untar the file with the following commands:

```
gunzip   lab8.tar.gz    // this will genenerate lab6.tar
tar  -xvf   lab8.tar
```

The last command will generate the lab8 directory. It is possible that, while downloading, the file system automatically unzips, and even expands (untars), the file. If that's the case, you may have to skip one or both of the above commands, though make sure to copy the new directory.

Your new lab8 directory should contain one directory: tpl-interpret . Double check that this is the case.

Load java and javacc by executing the command:

```
module add java javacc
```

Extending the syntax

The syntax of the new repeat statement is:

RepeatStatement → "repeat" "(" Expression ")" StatementBlock

In order to add this new statement, we must first extend the syntax of the programming language by modifying the TPL.jjt file. By doing this, the new generated parser will be able to parse **repeat** statements and generate ASTRepeatStatement trees.

Fire up your text editor of choice and load the TPL.jjt file. Add the following token declaration:

```
TOKEN : { < KEYREPEAT: "repeat" > }
```

Look for the non-terminal Statement() and add line that includes RepeatStatement() as a new statement:

```
void Statement() #void :        /* `#void' means create no default nodes. */
{}
{
  SkipStatement()
|
  RepeatStatement()   // added by you
|
```

Finally, Create the RepeatStatement() non-terminal by adding:

```
void RepeatStatement() :            /* Added "repeat" statement. */
{}
{
 <KEYREPEAT>  "("  Expression()   ")"      StatementBlock()
}
```

Typechecking and identification

Execute jjtree TPL.jjt and javacc TPL.jj. JJTree will add the default implementation of ASTRepeatStatement.java. We must include, at least, the implementations of identification, typecheck and interpret.

Before we do this, recall the informal specification of repeat statement. For example, the following code extract:

```
x = 5;
y = 0;
repeat(x+2) {
    y = y + 1;
    write y;
}
```

should execute the body of the repeat 7 times. What's should the output be?

Open the ASTRepeatStatement file and add:

```
public void identification () {
  jjtGetChild(0).identification();
  jjtGetChild(1).identification();
 }
```

In this case, the identification method only makes sure that the nodes (two) of RepeatStatement are traversed. Typechecking requires a bit more work. First, the two nodes need to be visited i.e. typechecked, and the result of the first node (the number of times the statement block needs to be executed), needs to be int. Add the following:

```
 public void typecheck () {
   jjtGetChild(0).typecheck();   // number of times- Expression
   jjtGetChild(1).typecheck();   // body -StatementBlock


   if (jjtGetChild(0).GetNodeType() != TPLTypes.intType)
     System.out.println("TPL Typechecker: for statement condition non-int");

    NodeType = TPLTypes.stmType;     // the type of RepeatStatement is stmtype
 }
```

The Interpreter

The implementation of interpret for RepeatStatement should be clear by now:

- Evaluate expression to an integer value, sya, n.
- Execute StatementBlock n times.

The Java code corresponding to the specification above should be:

```
public void interpret () {
    int n,i;
     jjtGetChild(0).interpret();
    n = ((Integer) stack.pop()).intValue());  // number of repeat-times
    i = 0;
    while (i < n) {
        jjtGetChild(1).interpret();   // Execute statement block
        i = i + 1;
}
```

Add the Java code to the ASTRepeatStatement.java file.

Compiling and Testing

Compile the files by executing the command javac *.java. In order to test the new statement you need to create a SIMPLE program that uses the repeat statement. For example, create the file **repeattest.tpl** with the following contents:

```
int x;
int y;
x = 5;
y = 0;
repeat(x+2) {
    y = y + 1;
    write y;
}
```

Run the program with:

java TPL repeattest.tpl.

Convince yourself that the implementation works by testing new examples.