CS4905 **Introduction to Compiler Construction**
Lab 3 February 12, 2007
Purpose: Building an interpreter and a symbol table.

1. Log in to a Linux workstation in ITD415. By virtue of being enrolled in CS4905, you should receive a Computer Science Linux-lab login ID and password via your UNB E-mail.

2. Create a subdirectory in your UNIX directory space to contain the source code for this lab. For purposes of illustration, I will assume that you call this subdirectory "L3".

Part 1. Experiments with yacc to build an interpreter

Recall the architecture of yacc (yet another compiler compiler) as shown in Figure 1 below.

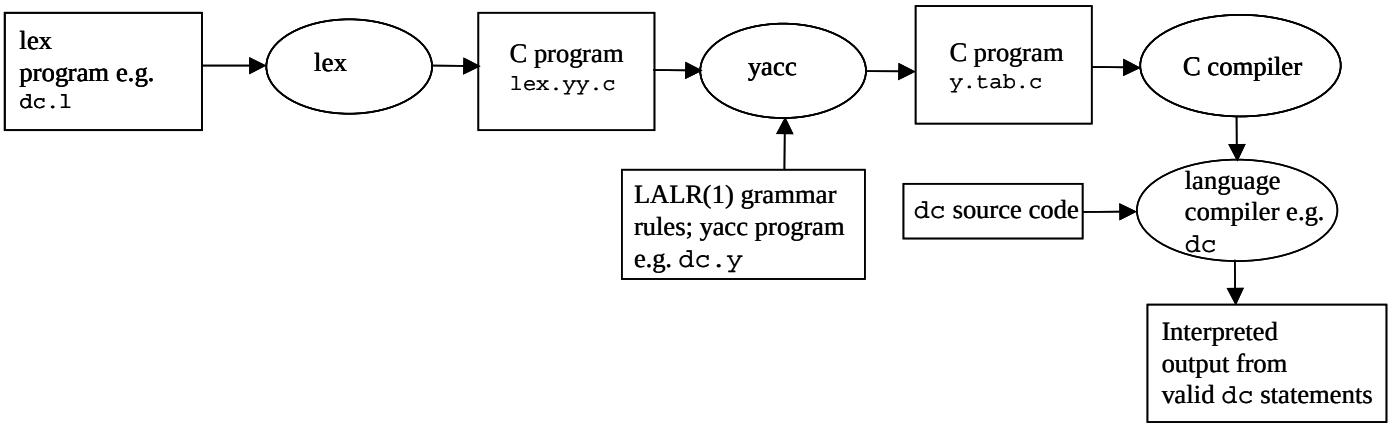

Figure 1. Using yacc to generate language interpreters.

3. Download the "`dc.l`" and "`dc.y`" files into your L3 subdirectory from the CS4905 web site http://www.cs.unb.ca/profs/nickerson/courses/cs4905/Examples/yacc/index.html (use e.g. Mozilla). The "dc" stands for "desktop calculator", and the initial "`dc.y`" is intended to generate a compiler to carry out calcuations from the command line. We already used this example in Lab 2. To avoid confusion, change the names of the files to e.g. "`bool.l`" and "`bool.y`".

4. Modify your desktop calculator to make it into an interpreter of logical expressions. The symbols of the grammar are

`t`	stands for "true"
`f`	stands for "false"
`!`	stands for "not" (i.e. logical negation)
`&&`	stands for "and"
`||`	stands for "or"

Assume that the priority of the operators is as given above; i.e. highest priority is for "not", 2nd highest priority is for "and" and the lowest priority is for "or". Note that logical negation ("not") in C is `!`, logical "and" in C is "`&&`", and logical "or" in C is "`||`". Use an integer `1` to represent "true", and `0` to represent "false". The stack type `YYSTYPE` should be changed from `double` to `int`. In addition, you will need to define tokens in yacc for return by lex that match the three boolean operators.

5. Test your boolean expression interpreter on the following logical expressions:

```
t && ! f
! ( t && ! t || t )
( f && t && ! f ) || ! ( f || t)
t && ( ! f || ( f && ! ( f || ! t && ! ( t && f ))))
( f || ! f ) && ! t || f
! f && ! t
```

If the result of the above expression is true, print "`true`"; otherwise print "`false`".

Part 2. Experiments with a symbol table

6. Download the "`L3.jj`", "`Main.java`" and "`HashT.java`" files into your L3 subdirectory (use e.g. Mozilla). File names ending in `.jj` are intended to contain JavaCC (Java compiler compiler) input.

7. Type

```
javacc-4.0 L3.jj
```

on the command line to compile the javacc into a Java program called MyParser. How can we get rid of the `Choice conflict in (...)* construct ...` warning messages (lines 34 and 37) generated by `javacc`? Note that version 4.0 of JavaCC is used to avoid the `enum` keyword conflict arising in version 2.1.

8. Type

```
javac Main.java
```

on the command line to compile the Java program into an executable (.class) program. This will automatically compile any dependent classes (e.g. the `HashT.java` hash table definition). You should see three errors arising from the fact that the hash table with external chaining (as shown on p.106 of the text) uses types

```
Object binding
```

and

```
Binding b
```

for specification of the object binding to be placed in the symbol table. Change the object binding type to `int` (for all instances of an `Object` or `Binding`), and assign the correct value for all integer bindings. Note that the hash function will need to be modified to return only postitive values for certain of the identifiers (hint: use the `Math.abs()` method). When you get the parser to compile, run it against the following test input data:

```
SizeOfOne = 2;
Offset = 5;
Base = 12;
Location1 = Offset * SizeOfOne - Base;
count = Location1 - 4 * 3;
print count;
CountFin = count * (count - Location1) / 2;
print CountFin
```

If you save the above test data in a file `L3.txt`, then file `L3.txt` is run through the interpreter using the command

```
java Main < L3.txt
```

This should result in a message stating

```
Interpreter successful
```

9. Modify the `L3.jj` and `HashT.java` programs to
   (a) print a result after each print statement, and
   (b) add a `print()` method to the `HashT` class that prints the entire contents of the symbol table after the parser has completed its translation of the entire "program".

   When run on the above `L3.txt` input file, you should see something like the following:

```
Print result = -18
Print result = 144
Key Base found at row 21
Key Location1 found at row 44
Key SizeOfOne found at row 56
Key count found at row 67
Key Offset found at row 83
Key CountFin found at row 98
Interpreter successful
```

Note that the reported `results` are incorrect. What would be required to change the program to get the correct `results`?