Computer Science 3 - 2015

Programming Language Translation

Informal Practical 4, Week beginning 21 September 2015

Hand in your solutions to this practical (and Practical 3) *before* lunch time on your next practical day, correctly packaged in a transparent folder with your cover sheets. **Unpackaged and late submissions will not be accepted - you have been warned.** Please do NOT come to a practical and spend the first hour printing or completing solutions from the previous week's exercises. Since the practical will have been done on a group basis, please hand in one copy of the cover sheet for each member of the group. These will be returned to you in due course, signed by the marker. **Please make it clear whose folder you have used for the electronic submission, for example g13A1234.** Lastly, please resist the temptation to carve up the practical, with each group member only doing one task. The group experience is best when you discuss each task together.

Objectives:

In this practical you are to

- extend the Parva grammar still further describe simple language features.
- develop a simple grammar for a friendlier assembler than the one you used last week!

You will need this prac sheet and your text book. As usual, copies of the prac sheet are also available at `http://www.cs.ru.ac.za/courses/CSc301/Translators/trans.htm` .

Outcomes:

When you have completed these practicals you should understand

- how to develop context-free grammars for describing the syntax of various languages and language features;
- the form of a Cocol description;
- how to check a grammar with Coco/R and how to compile simple parsers generated from a formal grammar description.

To hand in:

This week you are required to hand in, besides the cover sheet:

- Listings of your solutions to the grammar problems, produced on the laser printer by using the LPRINT utility Some of these listings will get quite "wide" so please set them out nicely.
- Electronic copies of your grammar files (ATG files).

I do NOT require listings of any C# code produced by Coco/R.

Keep the prac sheet and your solutions until the end of the semester. Check carefully that your mark has been entered into the Departmental Records.

You are referred to the rules for practical submission which are clearly stated in our Departmental Handbook. However, for this course pracs must be posted in the "hand-in" box outside the laboratory and not given to demonstrators.

A rule not stated there, but which should be obvious, is that you are not allowed to hand in another group's or student's work as your own. Attempts to do this will result in (at best) a mark of zero and (at worst) severe disciplinary action and the loss of your DP. You are allowed - even encouraged - to work and study with other students, but if you do this you are asked to acknowledge that you have done so. You are expected to be familiar with the University Policy on Plagiarism, which you can consult on the University web site.

Task 1 - creating a working directory and unpacking the prac kit

There are a few extra test files of silly Parva programs that you might find useful in the file `PRAC4.ZIP`

Task 2 - An even better calculator

You are probably sick of `CALC.ATG`. One last fling. Add in an exponentiation operator `↑` so that you can write expressions like

`4↑2 + 5↑(3+4)` that is "four squared plus five to the power seven" and `3↑4↑5` ????

Of course the trick here is to get the precedence and associativity correct…

Task 3 - A bigger and better Parva grammar

In the prac kit last week you found the grammar for the version of Parva used in the compiler for Prac 1, based on the one on page 170.

Now modify the grammar - which by now you should have extended somewhat - to add even more features. Specifically, add (and check that the additions work):

- Those useful assignment operators `+=` `-=` `*=` `/=` `%=` and so on.
- An alternative *for* statement suggested by the one in Python (look it up!).
- Optional *elsif* and *else* clauses for the *if* statement.
- A character type, and some way of converting from character values to integer values and *vice versa*.
- As a larger exercise, alter the grammar to recognize expressions whose operator precedence is the same as it is in C#, rather than the same as it is in Pascal (there is no need to add more *operators*, other than the `%` that you have already added). You may need to look up the operator precedence rules in C#.

Here are two silly examples of code that should give you some ideas:

```
void Main () {
// (not supposed to do anything useful!)
  int i = 0, j = 8, k = 4;
  char ch;
  for ch = 'z' downto 'a' {
    write(ch);;;;
    write(char(ch + 4), int(ch), i + j * ch);
  }
  for i in ( 3, 5, 7, k, 2 * k ) write(i);
  if (i > 4) ch = '6';
  elsif (i < 4) k = '7' + 45;
  elsif (i == 4) j *= 12;
  else ; // nothing
} // Main
```

These little programs and some others like them are in the kit, and you can easily write some more of your own. Start on something really simple, like:

```
void Main () {
  if (a)
    if (c) d();
    elsif (x) g();
    elsif (z) h();
    e f();
} // Main
```

and

```
void Main () {
  int a = 8;
  a += 5;
} // Main
```

Hint: All we require at this stage is the ability to *describe* these features. You do *not* have to try to give them any semantic meaning or write code to allow you to use them in any way. In later pracs we will try to do that, but please stick to what is asked for this time, and don't go being over ambitious.

Second Hint: Don't only test your system on correct programs. Give it a few that are (deliberately) incorrect and see what it does with them.

Task 4 How are things stacking up?

Develop a Cocol grammar that describes programs written in the PVM code that you struggled with in some previous exercises. That should be pretty easy, but be careful to describe the language as tightly as possible. Then, just to make the language more attractive for writing programs, suppose you had been able to use optional alphanumeric labels in your code as the destinations of branching instructions, as exemplified by (`Stk1.pvm`):

```
         DSP    2      ; one statement per line
LOOP     LDA    1      ; ignore comments starting ;
         INPI
         LDL    1
         BZE    EXIT   ; no need to count - fantastic
         BRN    LOOP   ; isn't this grand?
EXIT     LDL    1
         PRNI
         PRNS   "Time to say \"That's all\"!"
         HALT
```

Of course, your system won't actually be able to assemble and execute any of this code. Later we might extend it further to do that - for the moment simply describe programs written in the PVM assembler language.