Assignment 1: Making Pretty Printer for C programs (in a group of 2)
CMPT 470/816: Advanced Software Engineering
Full Marks: 100 (13% to the final)

What to do?

1. Look at the example grammars for C and Pascal at the "SupportMaterials" folder in Dropbox. In particular, look at how the Pascal grammar was gradually made. It followed kind of the same process as of the lectures in making the text, assignment and if-then-else grammars.

2. Now lets start making the C grammar for the example programs in folder "MakeCgrammar". You will see that there are 18 such programs. I also provided some Txl programs for the first few ones. For example, input 1abc_input.c will be run with txl programs 1a.Txl, 1b.Txl and 1c.Txl. Similarly, input 2_input.c will run with 2.Txl. Use the command, *txl 1a.Txl 1abc_input.c* and so on to compile. You do the rest. Because you are making a pretty-printer, make sure input programs of any format becomes the same as per your grammar. The bottom line idea would be the input programs should be displayed kind of the same as they are currently now. Also look at the Pascal pretty-printer provided in folder SupportMaterials/Pascal_PrettyPrinter. This Pascal grammar will give you lots of hints for making the C grammar. [45 marks]

3. Now for each of the example programs, make another similar example with names 1b_input.c, 2b_input.c, …, and 18b_input.c. Make sure that your corresponding Txl program runs on the new programs and pretty print them as per your grammar. [15 marks]

4. The grammar you made in step 2 above should run the programs in the folder "miniCgrammar". If so, you are at a very good stage. If not, refine the grammar, make it generalized enough and attempt to run the examples. They should pretty print any such programs. I provided an example as the starting point. Just run *txl StartingInput.c CPrettyPrinter.Txl* to see how it works. [30 marks]

5. Now make a couple of similar programs as of the examples (e.g., similar to input_4.c and input_5.c) in the folder "miniCGrammar" and run the grammar. If they pretty print well, you are done! [10 marks]

P.S. You might get lots of warning messages from TXL processor. You try to remove them. If not, no worries much.

How to start working on the grammar?

As noted above, start writing TXL grammar for the simple ones and then move building the complex ones. However, for each stage keep a copy of the old ones. For example, you make a grammar for a small syntactic construct in txl file named 1.Txl and it has the corresponding input file named 1_input. When you are done making the grammar for 1_input, you copy 1.Txl to make 2.Txl. You then further work on 2.Txl so that it will handle both input files of 1_input and 2_input where 2_input contains your next syntactic construct. In this way, you develop the complete grammar where the final grammar will support all the input files. This is kind of version control system. However, instead of using the version control, we are just keeping the old copies and making the new ones. When all the syntactic constructs are covered, you will need to change the grammar so that it will now support nested statements. For example, there might be a nested "if – then- else" statement inside of which there might be "for loop" and so on. You might want to try this first with a small example similar to the "if-then-else" statement I was showing in the lecture. You will need to submit all these small programs and their input files in Moodle as well. **If you follow a different method of developing the grammar that is also okay. However, keep the intermediate programs as an evidence of how you have come to the final grammar. Provide appropriate comments in your grammar as well.**

Expected output?

This assignment will help you learn of how to make a pretty-printer for the C programs and hence for any such languages thereafter. Any programs written in C in any formatting styles will produce the consistent formatting after you use this pretty-printer. This is kind of the tool *Artistic Style* (http://astyle.sourceforge.net/) which supports multiple languages. You will be a proud author of a pretty-printer for C language. Also note that you can now do lots of software analysis tasks with this. Just look at the TXL Cook book in the Readings (CourseMaterials/TXL/Readings) folder in the course in Dropbox. Covering full C language constructs would be difficult, and also the full C grammar is available in the TXL website. I am thus just expecting a mini C pretty-printer as of the examples provided in the assignment folder. Some highlights below:

a. The print and read statements.

b. The expression grammar for any expressions. You may have separate expression grammar for Boolean/conditional expressions as well. Feel free to copy from TXL website.

c. The assignment statement including complex assignment statements.

d. The method declarations and method calls. Note that some methods are built-in in the programming language of choice and some are user defined. You may want to

distinguish them or you may consider them as same type (i.e., consider the method name as [id] or so). They may accept any number of parameters of different types in the declarations or in the method calls.

e. The variable and constant declarations. You don't need to support all the different types. If your grammar can accept integers, I believe you can also make for other types as well.

f. The if-then and if-then-else statements including the complex conditional statements within their conditions such as the *if (i > 1)&& (i < 100)*. You might want to have a separate non-terminal for representing the conditional statements.

g. The case/switch statement

h. The for-loop statement

i. The while-loop statement

j. Make sure that you apply appropriate formatting clues to have consistent output.

How to submit?

1. Make a directory named as: Assgn4_ PatALastName_PatBLastName.

2. Copy the folders MakeCgrammar and MiniCgrammar including the both the TXL and input files in folder Assgn4_ PatALastName_PatBLastName.

3. Make a zip/tar file of Assgn4_ PatALastName_PatBLastName and submit to Moodle. In case you experience troubles in submitting in Moodle, just email me your submission.

4. Enjoy and please feel free to contact me if you have any questions. I am always available for my students, even at late night or during the weekend. I don't mind getting your emails anytime you send. You will also get a reply at the earliest time possible, possibly within an hour or so. If you are successful (even partially) in making the grammar, your hard efforts of coming to this lecture is well paid off, at least in my opinion. My email id is: chanchal.roy@usask.ca