TDT 4205
Problem Set 2

The deadline for this problem set is friday, September 28th. Submissions will only be accepted through *It's learning*.

1 Top-down parsing and grammars

Consider the following grammar:

S → u B D z
B → B v | w
D → E F
E → y | ϵ
F → x | ϵ

1.1 10%

Compute the NULLABLE ($\rightarrow \epsilon$?) set of the grammar.

1.2 10%

Compute the FIRST set of the grammar.

1.3 10%

Compute the FOLLOW set of the grammar.

1.4 10%

Construct a parsing table, and show that the grammar is not LL(1).

Practical work

As with PS1, a code archive is made available on *It's learning* for you to start with. It contains two directories:

1. 'pencil', which contains a (modified) copy of last week's skeleton code
2. 'vsl', which contains a simple setup for building a small interpreter

The skeleton codes use the 'flex' and 'bison' generators for the scanner/parser specification languages 'Lex' and 'Yacc'. How to work with these tools will be the topic at our next recitation, and notes will be made available online.

2 Pencil language makeover

This task deals with the 'pencil' subdirectory. Everything is set up to compile just as for PS1; your implementation should go into the file `scanner.l`.

2.1 20%

Starting from the provided code, reimplement the scanner from PS1, using Lex instead of constructing the automaton by hand.

3 Interpreting simple expressions

This task deals with the 'vsl' subdirectory. This directory is further divided into 'src', 'work', 'obj' and 'bin', and contains a file for the 'make' tool, so the project can be built by issuing the `make` command in the 'vsl' directory.

- `src` contains all the source code you need to handle
- `work` is filled with generated C code as the program builds
- `obj` is filled with generated object code when the program builds
- `bin` receives the executable binary program 'vslc' when a build finishes

Thus, to start the program with some input with 'vsl' as your working directory, the command will look like
`echo 'this is some input' | bin/vslc`

3.1 40%

Starting from the provided code, implement a simple interpreter for the following grammar, using Lex and Yacc:

program → print_statement
print_statement → PRINT print_list
print_list → print_item | print_list , print_item
print_item → expression | text

expression → expression + expression
| expression - expression
| expression * expression
| expression / expression
| - expression
| (expression)
| integer

integer → INTEGER
text → TEXT

- INTEGER is a 32-bit signed integer in standard decimal notation
- TEXT is any string quoted in double quotation marks (“”)

This grammar is slightly peculiar for what we want to do: the first few productions are not really necessary in order to interpret expressions. The reason for this is that the grammar is a subset of VSL, and we will extend it to cover more of that language in the subsequent problem sets.

Tips

- The choice of tokens vs. nonterminals gives a suggested splitting of the responsibilities of scanner and parser.
- As we are not yet interested in building a syntax tree, output can be performed already at the print_item level, as the expressions/precedence have already been grouped properly at this point.
- Start with the scanner `src/scanner.l`, then proceed to the parser `src/parser.y`. In this manner, you can test the scanner separately by putting simple productions of your own design in the parser, until you are satisfied with how the two play together. There is an example in the supplied code.
- The supplied code is provided to make it easier to get started with Lex and Yacc, but not to be boilerplate code that just works. If something is unclear, read both the code and the comments. If it is still unclear, ask questions. The next problem set will assume that you are comfortable with how these two tools work together and with other code.