Compilation

TP 3.1 : The Boost.Spirit parser framework

S. Filip

Boost.Spirit is an object oriented recursive descent parser generator framework which is part of the Boost C++ libraries. Parser objects are composed through C++ operator overloading and the result is a backtracking LL(∞) parser that can be used to parse quite ambiguous grammars. Spirit can be used as a lexer and/or parser.

Although Flex and Bison are powerful tools for generating C-based lexical analyzers and grammars, they can be a bit heavy to use for small parsing projects. Spirit aims to streamline this, by using C++ template meta-programming to express grammar productions using an approximate syntax of Extended Backus Naur Form (EBNF) completely in C++. One of the downsides of using Spirit is exactly the element which gives it such expressive power, the use of templates. If one is trying to write a complex grammar, it is very easy to make an error and receive a downpour of cryptic template error messages. This can, in some cases, make the project hard to debug. This is in contrast to the fact that Flex and Bison provide much better error messages and high level warnings (like for example shift/reduce conflicts), a byproduct of them being external tools. Nevertheless, there are projects which are using Boost.Spirit with very good results[1].

The purpose of the next exercises is to provide a 'gentle' introduction to Boost.Spirit and how it can be used to write parsers for relatively simple tasks. You are encouraged to read through the code and the comments to gain an understanding of how to use the library. For more information about the library and its evolution over the years, you can look at the corresponding Boost documentation[2].

Exercise 1. *Writing a JSON parser in Boost.Spirit*

This exercise shows you how to use Boost.Spirit for writing a simple parser for handling JSON files. JSON is an open standard format for transmitting data objects as attribute-value pairs. A simple grammar for describing JSON objects is expressed in EBNF like so:

$$\begin{aligned}
Object &= ''\{''Member('',''Member)^{*}''\}'' \\
Member &= String'' :''Value \\
String &=' ''' Character *' ''' \\
Value &= String|Number|Object|Array|''true''|''false''|''null'' \\
Number &= Integer|Real \\
Array &= ''[''Value('',''Value)^{*}'']''
\end{aligned}$$

To do:

- Download and unzip `spirit_json_parser0.tar.gz`. Analyze the code and understand what it does. Run it on different JSON inputs to verify that it is indeed a valid parser.
- We want to also add actions to our grammar. Download and unzip `spirit_json_parser1.tar.gz`. Study the code and understand how actions can be added for our grammar through C++ operator overloading. Run it on different input strings.

Exercise 2. *Arithmetic expression evaluation*

Download and unzip `spirit_addsub.tar.gz` and look at how we use Boost.Spirit to parse and evaluate input consisting of additions and subtractions of real values.
To do:

- Enhance the arithmetic expression evaluation grammar so that it can also handle multiplications, divisions and parentheses inside the input strings to be parsed. Test your implementation on appropriate input expressions.

[1] see for example `http://boost-spirit.com/home/info/who-is-using-spirit/`
[2] a recent version can be found at `http://www.boost.org/doc/libs/1_56_0/libs/spirit/doc/html/index.html`