

Homework 2: Parsing

Out Date:

09/27/2006

Due Date:

10/04/2006

Objectives

The purpose of this homework is to write the core of the parser for C-. You will be using the CUP parser generator to produce a Java class that implements your parser. By the end of this homework you will actually recognize whether a string belongs to the language C-.

Handout

To obtain your assignment, go to the class website in the homework section and download `hw2-packet.zip`. Similar to Hw1, this archive is a template project that you can import into Eclipse. It contains a little bit of support code and basic structure. It is up to you to “fill in” the blanks (mostly in `parser.cup`). In the zip file you will also find two files `test.cmm` and `test2.cmm` that written in C-. Make sure that your parser recognizes the first file without error while it reports syntax errors for the second. You should correctly report the location of at least one syntax error.

1 Grammar

Your first task is to derive a grammar that is LALR(1) compliant (i.e., that JavaCUP can process it with no trouble). You are starting from an EBNF grammar a special grammar notation explained below. The challenge is to find and apply the necessary rewrites so that the grammar becomes LALR(1) and still recognize the same language (C-). Figure 1 shows the complete grammar for C-.

The symbols in italics are the non-terminals. The symbols in black, normal font are the terminals and the red bold symbols are the EBNF annotations. Remember that EBNF is reminiscent of regular expressions. Specifically

- a fragment **[α]** means that the string α is optional.
- a fragment **{ $\alpha|\beta|\gamma$ }** is an alternative and means that we could see either α or β or γ .
- a fragment α^* is a Kleene closure of α , meaning that the string α can appear 0, 1, or more times.

together with this grammar, assume the traditional precedence for the arithmetic operators (e.g., $*$ binds more than $+$).

The main difficulty is to rewrite the grammar to eliminate the EBNF construction and produce an LALR(1) compliant grammar. Remember that CUP can use operator precedence to disambiguate some constructions (but not all, so rewrites will be necessary). The answer to this question should be in the separate `rtf` file that is part of the package.

2 CUP

Once you have a reasonable grammar, it is time to turn to CUP. Your task is to use CUP and your brand new LALR(1) grammar to implement a parser class. CUP takes as input a `parser.cup` file that contains the required specifications to tie it to a lexer. We are providing the lexer code (feel free to discard our code

```

classList ::= classDecl+
classDecl ::= class ID [ extends ID ] { decl* };
    decl ::= type ID;
           | type ID ([formals]) { statement* }
           | ID ([formals]) {statement* }
formals ::= formal[, formal]*
statement ::= expr ;
           | type ID ;
           | assignment ;
           | { statement* }
           | while ( expr ) statement
           | if (expr) statement [else statement]
           | for ( [assignment]; [expr]; [assignment]) statement
           | return expr ;
           | ;
assignment ::= lvalue{ = expr | ++ | -- }

expr ::= expr op expr
       | uop expr
       | NUMBER
       | ( expr )
       | lvalue
       | new ID ( [actuals] )
       | new type [ expr .. expr ]
op ::= &&||||=|!=|<=|>=|<|>|+|-|*|/
uop ::= ! | -
lvalue ::= ID [ ( [actuals] ) ]
         | lvalue [ expr ]
         | lvalue . ID [ ([actuals] ) ]
         | self
actuals ::= expr[, expr]*
formal ::= type ID
type ::= { int | bool | void | ID } [ [] ]

```

Figure 1: EBNF Grammar for C--

in favor of your own from hw1 if you want to). The parser.cup file is missing the grammar though. So you must specify your non-terminals and all your grammar productions. Once this is done, you can run CUP from the ant build file to create your parser. If the grammar was indeed LALR(1), you should end-up with a parser.java file and CUP should complete its task with **no conflicts whatsoever**. You can, of course, use/define operator precedences to eliminate conflicts as well as further grammatical rewrites. Remember that you must recognize any syntactically correct C--programs and nothing else.

Note that if you do revise your grammar, you should revise your answer to question 1. You will also need to include some error-productions to assist the the parser in recovering from syntax errors.

3 Driver

The driver class already exists. Once your parser is correct, you should be able to parse a test program. At the moment, the objective for the parser is to determine whether the input belongs to the C--language and do some rudimentary syntax error reporting.

4 Hand-in

Your solution should include the following:

1. The `rtf` file that contains the grammar in LALR(1) form and any explanations you feel necessary for explaining your solution.
2. The file `parser.cup` that contains the grammar implemented in JavaCUP.

You submit your homework by midnight on the due date as a zip archive exported from eclipse. The e-mail address to submit is `c244fa07@cse.uconn.edu`.