CS 241 — Spring 2020 — Assignment 6

Assignments for CS 241

← Assignment 5 | Assignment 6 | Assignment 7 →

Monday, June 22nd at 5:00 pm **Friday, June 26th at 5:00 pm** ~~Friday, July 10th, at 5:00 pm~~ Sunday, July 12th, at 5:00 pm

P1 • P2 • P3 • P4 • P5 • P6 • P7 • P8

Marmoset Notes

For each question, Marmoset has public and release tests, and often has secret tests as well. The weights assigned to each question are given on the assignment.

When you release test a submission, Marmoset will only report on public and release tests. No information about secret tests will be visible until after the assignment deadline.

This means the number of marks you see on Marmoset (prior to the assignment deadline) will not necessarily match the number of marks shown on the assignment page.

Part I. Scanning

Problem 1 — 20 marks of 80 (filename: `wlp4scan.rkt` or `wlp4scan.cc`)

Write a scanner (lexical analyzer) for the WLP4 programming language. Your scanner should read an entire WLP4 program and identify the tokens, in the order they appear in the input. For each token, your scanner should compute the name of the token and the lexeme (the string of characters making up the token), and print it to standard output, one line per token.

For example, the correct output for this WLP4 program

```
//
// WLP4 program with two integer parameters, a and b
//   returns the sum of a and b
//
   int wain(int a, int b) {
     return a + b;   // unhelpful comment about summing a and b
   }
```

is:

```
INT int
WAIN wain
LPAREN (
INT int
ID a
COMMA ,
INT int
ID b
RPAREN )
LBRACE {
RETURN return
ID a
```

```
PLUS +
ID b
SEMI ;
RBRACE }
```

If the input cannot be scanned as a sequence of valid tokens (possibly separated by white space as detailed in the WLP4 language specification), your program should produce exactly one line of output to standard error containing the string `ERROR`, in upper case. We recommend that you try to make the error message informative.

A reference implementation of the scanner called wlp4scan is available after running `source /u/cs241/setup`. When testing your program, you can compare its output to wlp4scan's to make sure it is correct. Run it as follows:

```
wlp4scan < input.wlp4 > tokens.txt
```

You may wish to modify the starter code from Assignment 3 to solve this problem. Note that the scanner in these starters is based on a DFA that repeatedly recognizes the longest prefix of a string that is a token. You should be able to replace the DFA (which recognizes MIPS assembly language tokens) by one which recognizes WLP4 tokens.

Note that **all Marmoset tests for this problem are public**. Also, only the first 30 lines of each test case for this problem are visible, so if you fail a test, it might be due to something that happens after the first 30 lines.

Click here to go back to the top of the page.

Part II. Context-free Grammars

You are to create context-free grammars and derivations represented as CFG files each of which is a textual reprentation of a CFG, optionally followed by a textual representation of several derivations. The student.cs environment provides a tool `cs241.cfgcheck` which checks the validity of the CFG and derivations.

Problem 2 — 7 marks of 80 (filename: `a6p2.cfg`)

Create a file representing the CFG with the following production rules:

```
S → BOF A EOF
A → x
A → A x
```

Click here to go back to the top of the page.

Problem 3 — 7 marks of 80 (filename: `a6p3.cfg`)

Add derivations for the following strings to the CFG file created for Problem 2:

```
BOF x EOF
BOF x x EOF
BOF x x x EOF
```

Click here to go back to the top of the page.

Problem 4 — 7 marks of 80 (filename: `a6p4.cfg`)

The following production rules yield an ambiguous CFG:

```
S → BOF A EOF
A → x
A → A x A
```

Find a string for which the derivation is ambiguous. Compose a CFG file representing the CFG and two different derivations for your string.

Click here to go back to the top of the page.

Problem 5 — 7 marks of 80 (filename: `a6p5.cfg`)

Add a derivation to this context-free grammar for the following sequence of symbols:

```
BOF id EOF
```

Click here to go back to the top of the page.

Problem 6 — 7 marks of 80 (filename: `a6p6.cfg`)

Add a derivation to this context-free grammar for the following sequence of symbols:

```
BOF ( id - ( id - id ) ) - id EOF
```

Click here to go back to the top of the page.

Problem 7 — 7 marks of 80 (filename: `a6p7.cfg`)

Add a derivation to this augmented grammar for WLP4 for the following (augmented) sequence of WLP4 tokens:

```
BOF INT WAIN LPAREN INT ID COMMA INT ID RPAREN LBRACE RETURN ID PLUS ID STAR ID
SEMI RBRACE EOF
```

Click here to go back to the top of the page.

Problem 8 — 18 marks of 80 (filename: `galaxy.rkt` or `galaxy.cc`)

Write a Racket or C++ program that reads a CFG file that contains the same grammar as for Problems 5 and 6 with the example derivation replaced by another valid derivation for the same grammar.

Assume that each occurrence of the symbol `id` represents the value `42`, that each occurrence of `-` represents subtraction, that operations associate from left to right, except where overridden by parentheses. Output a single line giving the value of the derived expression.

The correct output for the sample grammar and derivation is a single line containing `-42`.

Notes:

- *The grammar is fixed, so you may, if you wish, simply skip over the grammar part of the .cfg file, and embed assumptions about the syntax directly into your program.*
- *The input will have exactly one derivation following the grammar.*

- *The derivation will be valid.*
- *It is not necessary to build a parse tree or any other data structure to solve this problem. If you wish to approach the problem recursively, it is possible to do so by just reading the input directly, rather than first building a tree.*
- *You may find these programs useful: derivationprinter.rkt derivationprinter.cc. Each reads a .cfg file, skips over the grammar, and prints the rules used in the derivation, with the indentation removed. You will want to replace the part that reads and prints the rules, and process the rules to evaluate the expression rather than just printing them out.*

Click here to go back to the top of the page.