Lab8 - Introduction to bison

Started: Oct 25 at 12:20pm

Quiz Instructions

Question 1 — 40 pts

The **Cambridge-Polish Notation** (**CPN**) is a special prefix notation expressions (that we will call **s-expressions**), in which the operator name and the operands are enclosed in parentheses.

For example, `1+2` (or `add(1,2)`) is denoted as:

```
(add 1 2)
```

Any function can be used in that way, and more complex can be built using the following grammar:

```
s-expr ::= number | f_expr
f_expr ::= ( func s-expr ) | ( func s-expr s-expr )
number ::= [ + | - ] digit+ [ . digit+ ]
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

For example, the following s-expression:

```
(add (exp (sub 3.5 (sqrt 15))) (log (remainder (pow 3 (sqrt 20)) 5.2)))
```

is equivalent to a more familiar equivalent notation:

```
add(exp(sub(3.5, sqrt(15))), log(remainder(pow(3, sqrt(20)), 5.2)))
```

...or, to even more familiar traditional notation:

```
exp(3.5 - sqrt(15)) + log(remainder(pow(3, sqrt(20)), 5.2))
```

Please download **lab8_bison_t1.zip (https://cilearn.csuci.edu/files/511482/download?wrap=1)** file that contains a starting implementation of the s-expression calculator. Open the root directory in CLion, and then study the included yacc, lex, and C code recalling from the lecture, and the previous lab, how all the bits and pieces work together.

In this task, you are to complete the implementation of the calculator by completing the implementation of the `calc()` function that evaluates s-expressions.

Note that the following part of the bison code allows you to specify types of some tokens (that is part of semantic analysis that we will study in the coming weeks).

```
%union
{
 double dval;
 char *sval;
}

%token <sval> FUNC
%token <dval> NUMBER
%token LPAREN RPAREN EOL QUIT

%type <dval> s_expr f_expr
```

When you understand the code, move on to implement the missing functionality, so that the value of an s-expression is calculated properly.

When you run the code, you will enter a **Read-Evaluate-Print-Loop** (**REPL**) in which repetitively an expression is read from the standard input, evaluated, and the result is printed on the standard output. Please do not change the code of the `main()` function in the flex file, since the whole REPL logic depends on it.

The following is a sample session:

```
> (add 1 2)
3.000000
> (add (exp (sub 3.5 (sqrt 15))) (log (remainder (pow 3 (sqrt
20)) 5.2)))
0.626430
> quit
```

Please note that the value in the example assumes that the logarithm with the base of 10 is used as implemented by `log10()`. If you use the natural logarithm function (as implemented by `log()`), then the result will be as follows:

```
$ ./lab8_bison_t1
> (add (exp (sub 3.5 (sqrt 15))) (log (remainder (pow 3 (sqrt
20)) 5.2)))
0.545349
> quit
```

To use the functions from the Math library you need to include their definitions from `math.h`. You can explore the functions provided by the Math library by issuing the

following command:

```
$ man math
```

Please submit a zip archive of your complete CLion project directory.

Upload Choose a File

Question 2 60 pts

An **expert system** is a software system that attempts to mimic human intelligence by utilizing a **knowledge base**, a collection of facts, to provide answers to relevant queries just like a human expert would do. For example, a medical expert system may diagnose patients based on the observed symptoms.

A simple knowledge base based on the zero-order (propositional) logic is a collection of propositions. Following the principle of so-called **closed-world assumption**, if a proposition is in the knowledge base, then it is assumed to be true; otherwise, it is assumed to be false. New propositions might be added to the knowledge base (i.e., made true) through **assertions**. A process of **retraction** removes a proposition from the knowledge base (i.e., making it false).

Your task is to implement a **simple truth maintenance system** (**TMS**) that allows users to assert and retract propositions from the knowledge base, and to query the database using logical operators. The following transcript shows a sample user interactive session with the TMS using a special-purpose language, `ciProlog`, created by your instructors especially for this assignment:

```
?- assert('It rains')
=> TRUE

?- 'It rains'
=> TRUE

?- assert('It is cold')
=> TRUE
```

```
?- 'It is cold'
=> TRUE

?- 'It rains' AND 'It is cold'
=> TRUE

?- retract('It is sunny')
=> FALSE

?- 'It rains' OR 'It is sunny'
=> TRUE

?- NOT ('It rains' OR 'It is cold')
=> FALSE

?- 'It rains' AND 'It is sunny'
=> FALSE

?- NOT ('It rains' AND 'It is sunny')
=> TRUE

?- retract('It rains')
=> TRUE

?- 'It rains'
=> FALSE

?- 'It rains' OR 'It is cold'
=> TRUE

?- 'It rains' AND 'It is cold'
=> FALSE

?- halt
```

The following grammar generates `ciProlog` statements:

```
logexpr ::=
proposition
| assert ( proposition )
```

```
| retract ( proposition )
| ( logexpr )
| NOT logexpr
| logexpr AND logexpr
| logexpr OR logexpr

letter ::= [a-zA-Z]
proposition ::= '{letter}({letter}| )*'
```

The logical operators `NOT`, `AND`, and `OR`, have customary semantics used in constructing logical expressions.

The `assert` operation should respond `TRUE` if successful, and `FALSE` if not (i.e., when the proposition has not been added due to some problems; e.g, failed memory allocation). The `retract` operation should also respond `TRUE` if successful (i.e., the proposition has been found and successfully deleted from the knowledge base), and `FALSE` otherwise.

Download the archived CLion project **`lab8_bison_t2.zip`** that has some startup code. The code implements the TMS as a **REPL** (**read-evaluate-print-loop**) using lex/flex and yacc/bison for the task. You must complete the implementation by augmenting as necessary the included four files: `ciProlog.h`, `ciProlog.c`, `ciProlog.l`, and `ciProlog.y`. You must not change any code that is given to you; just expand it by adding the missing pieces.

Please submit a zip archive of your complete CLion project directory. You must include a transcript from your necessarily comprehensive testing sessions.

Upload Choose a File

Saving... Submit Quiz