REPL for interpreter using Flex/Bison

Ask Question

9

I've written an interpreter for a C-like language, using Flex and Bison for the scanner/parser. It's working fine when executing full program files.

Now I'm trying implement a REPL in the interpreter for interactive use. I want it to work like the command line interpreters in Ruby or ML:

3

1. Show a prompt
2. Accept one or more statements on the line
3. If the expression is incomplete
    1. display a continuation prompt
    2. allow the user to continue entering lines
4. When the line ends with a complete expression
    1. echo the result of evaluating the last expression
    2. show the main prompt

My grammar starts with a `top_level` production, which represents a single statement in the language. The lexer is configured for interactive mode on stdin. I am using the same scanner and grammar in both full-file and REPL modes, because there's no semantic difference in the two interfaces.

My main evaluation loop is structured like this.

```
while (!interpreter.done) {
    if (interpreter.repl)
        printf(prompt);
    int status = yyparse(interpreter);
    if (status) {
        if (interpreter.error)
            report_error(interpreter);
    }
    else {
        if (interpreter.repl)
            puts(interpreter.result);
    }
}
```

This works fine except for the prompt and echo logic. If the user enters multiple statements on a line, this loop prints out superfluous prompts and expressions. And if the expression continues on multiple lines, this code doesn't print out continuation prompts. These problems occur because the granularity of the prompt/echo logic is a `top_level` statement in the grammar, but the line-reading logic is deep in the lexer.

What's the best way to restructure the evaluation loop to handle the REPL prompting and echoing? That is:

- how can I display one prompt per line
- how can I display the continuation prompt at the right time

- how can I tell when a complete expression is the last one on a line

(I'd rather not change the scanner language to pass newline tokens, since that will severely alter the grammar. Modifying `YY_INPUT` and adding a few actions to the Bison grammar would be fine. Also, I'm using the stock Flex 2.5.35 and Bison 2.3 that ship with Xcode.)

c bison lex read-eval-print-loop flex-lexer

share improve this question

edited May 6 '13 at 11:18

lesmana
14.7k 6 50 74

asked Jul 9 '11 at 18:56

Jay Lieske
1,753 2 17 32

add a comment

2 Answers

active oldest votes

5

After looking at how languages like Python and SML/NJ handle their REPLs, I got a nice one working in my interpreter. Instead of having the prompt/echo logic in the outermost parser driver loop, I put it in the innermost lexer input routine. Actions in the parser and lexer set flags that control the prompting by input routine.

I'm using a reentrant scanner, so `yyextra` contains the state passed between the layers of the interpreter. It looks roughly like this:

```
typedef struct Interpreter {
    char* ps1; // prompt to start statement
    char* ps2; // prompt to continue statement
    char* echo; // result of last statement to display
    BOOL eof; // set by the EOF action in the parser
    char* error; // set by the error action in the parser
    BOOL completeLine // managed by yyread
    BOOL atStart; // true before scanner sees printable chars on line
    // ... and various other fields needed by the interpreter
} Interpreter;
```

The lexer input routine:

```
size_t yyread(FILE* file, char* buf, size_t max, Interpreter* interpreter)
{
    // Interactive input is signaled by yyin==NULL.
    if (file == NULL) {
        if (interpreter->completeLine) {
            if (interpreter->atStart && interpreter->echo != NULL) {
                fputs(interpreter->echo, stdout);
                fputs("\n", stdout);
                free(interpreter->echo);
                interpreter->echo = NULL;
            }
            fputs(interpreter->atStart ? interpreter->ps1 : interpreter->ps2, stdout
            fflush(stdout);
        }

        char ibuf[max+1]; // fgets needs an extra byte for \0
        size_t len = 0;
        if (fgets(ibuf, max+1, stdin)) {
            len = strlen(ibuf);
            memcpy(buf, ibuf, len);
            // Show the prompt next time if we've read a full line.
            interpreter->completeLine = (ibuf[len-1] == '\n');
        }
        else if (ferror(stdin)) {
```

```
            // TODO: propagate error value
        }
        return len;
    }
    else { // not interactive
        size_t len = fread(buf, 1, max, file);
        if (len == 0 && ferror(file)) {
            // TODO: propagate error value
        }
        return len;
    }
}
```

The top level interpreter loop becomes:

```
while (!interpreter->eof) {
    interpreter->atStart = YES;
    int status = yyparse(interpreter);
    if (status) {
        if (interpreter->error)
            report_error(interpreter);
    }
    else {
        exec_statement(interpreter);
        if (interactive)
            interpreter->echo = result_string(interpreter);
    }
}
```

The Flex file gets these new definitions:

```
%option extra-type="Interpreter*"

#define YY_INPUT(buf, result, max_size) result = yyread(yyin, buf, max_size, yyextra

#define YY_USER_ACTION  if (!isspace(*yytext)) { yyextra->atStart = NO; }
```

The `YY_USER_ACTION` handles the tricky interplay between tokens in the language grammar and lines of input. My language is like C and ML in that a special character (';') is required to end a statement. In the input stream, that character can either be followed by a newline character to signal end-of-line, or it can be followed by characters that are part of a new statement. The input routine needs to show the main prompt if the only characters scanned since the last end-of-statement are newlines or other whitespace; otherwise it should show the continuation prompt.

share improve this answer

answered Aug 14 '11 at 18:20

Jay Lieske
1,753 2 17 32

add a comment

0

I too am working on such an interpreter, I haven't gotten to the point of making a REPL yet, so my discussion might be somewhat vague.

Is it acceptable if given a sequence of statements on a single line, only the result of the last expression is printed? Because you can re-factor your top level grammar rule like so:

top_level = top_level statement | statement ;

The output of your top_level then could be a linked list of statements, and interpreter.result would be the evaluation of the tail of this list.

share improve this answer

answered Jul 10 '11 at 22:36

wmercer
603 1 7 20

1 That's roughly what I had in my grammar before I started refactoring it for the REPL. The challenge of the recursive pattern is that it's greedy: the parser will keep requesting lookahead tokens (i.e. new lines of input from the scanner) until one of the branches fails or it reaches EOF. – Jay Lieske Jul 10 '11 at 23:35

how about right-recursion: top_level = statement top_level | statement – Foo Bah Aug 10 '11 at 22:10

add a comment