Build Your Own Lisp

Eric Bailey

May 10, 2018 [1]

[1] Last updated May 12, 2018

Write an abstract

Contents

Prompt

1a ⟨*Print version and exit information.* 1a⟩≡

```
puts("Lispy v0.0.1");
puts("Press ctrl-c to exit\n");
```

Uses `Lispy` 1d.
This code is used in chunk 3a.

1d ⟨*Create some parsers.* 1d⟩≡

```
mpc_parser_t *Digit    = mpc_new("digit");
mpc_parser_t *Integer  = mpc_new("integer");
mpc_parser_t *Decimal  = mpc_new("decimal");
mpc_parser_t *Number   = mpc_new("number");
mpc_parser_t *Operator = mpc_new("operator");
mpc_parser_t *Expr     = mpc_new("expr");
mpc_parser_t *Lispy    = mpc_new("lispy");
```

Defines:
`Decimal`, used in chunk 2b.
`Digit`, used in chunk 2b.
`Expr`, used in chunk 2.
`Integer`, used in chunk 2b.
`Lispy`, used in chunks 1 and 2.
`Number`, used in chunk 2.
`Operator`, used in chunk 2.
Uses `mpc_parser_t` 4d.
This code is used in chunk 3a.

Here, `input` is functionally equivalent to `input` $\neq$ `NULL`, and `*input` is functionally equivalent to `input[0]` $\neq$ `'\0'`, i.e. `input` is non-null and nonempty, respectively.

1b ⟨`input` *is nonempty* 1b⟩≡

```
input && *input
```

This code is used in chunk 3a.

1c ⟨*Add* `input` *to the history table.* 1c⟩≡

```
add_history(input);
```

Uses `add_history` 4c.
This code is used in chunk 3a.

Define the Lispy grammar.

2a ⟨*parsing.c* 2a⟩≡

```
#define LISPY_GRAMMAR \
        " digit    : /[0-9]/ ;                                 " \
        " integer  : /-?/ <digit>+ ;                           " \
        " decimal  : /-?/ <digit>+ '.' <digit>+ ;              " \
        " number   : <decimal> | <integer> ;                   " \
        " operator : '+' | '-' | '*' | '/' ;                   " \
        " expr     : <number> | '(' <operator> <expr>+ ')' ;   " \
        " lispy    : /^/ <expr>+ /$/ ;                         "
```

This definition is continued in chunk 3a.
Root chunk (not used in this document).

2b ⟨*Define the parsers with the Lispy grammar.* 2b⟩≡

```
mpca_lang(MPCA_LANG_DEFAULT, LISPY_GRAMMAR,
          Digit, Integer, Decimal, Number,
          Operator, Expr, Lispy);
```

Uses Decimal 1d, Digit 1d, Expr 1d, Integer 1d, Lispy 1d, Number 1d, and Operator 1d.
This code is used in chunk 3a.

2c ⟨*Undefine and delete our parsers.* 2c⟩≡

```
mpc_cleanup(4, Number, Operator, Expr, Lispy);
```

Uses Expr 1d, Lispy 1d, Number 1d, Operator 1d, and mpc_cleanup 4d.
This code is used in chunk 3a.

2d ⟨*The input can be parsed as Lispy code.* 2d⟩≡

```
mpc_parse("<stdin>", input, Lispy, &res)
```

Uses Lispy 1d and mpc_parse 4d.
This code is used in chunk 2g.

2g ⟨*Attempt to parse the user input.* 2g⟩≡

```
mpc_result_t res;
if (⟨The input can be parsed as Lispy code. 2d⟩) {
    ⟨Print and delete the AST. 2e⟩
} else {
    ⟨Print and delete the error. 2f⟩
}
```

Uses mpc_result_t 4d.
This code is used in chunk 3a.

2e ⟨*Print and delete the AST.* 2e⟩≡

```
mpc_ast_print(res.output);
mpc_ast_delete(res.output);
```

Uses mpc_ast_delete 4d and mpc_ast_print 4d.
This code is used in chunk 2g.

2f ⟨*Print and delete the error.* 2f⟩≡

```
mpc_err_print(res.error);
mpc_err_delete(res.error);
```

This code is used in chunk 2g.

3a ⟨*parsing.c* 2a⟩+≡
⟨*Include the necessary headers.* 3b⟩

```
int main(int argc, char *argv[])
{
    ⟨Create some parsers. 1d⟩

    ⟨Define the parsers with the Lispy grammar. 2b⟩

    ⟨Print version and exit information. 1a⟩

    bool nonempty;
    do {
        char *input = readline("> ");
        if ((nonempty = (⟨input is nonempty 1b⟩))) {
            ⟨Add input to the history table. 1c⟩
            ⟨Attempt to parse the user input. 2g⟩
        }

        free(input); // N.B. This is a no-op when !input.
    } while (nonempty);

    ⟨Undefine and delete our parsers. 2c⟩

    return 0;
}
```

Uses bool 3c, free 4b, and readline 4c.

Headers

3b ⟨*Include the necessary headers.* 3b⟩≡
⟨*Include the boolean type and values.* 3c⟩
⟨*Include the standard I/O functions.* 4a⟩
⟨*Include the standard library definitions.* 4b⟩

⟨*Include the line editing functions from libedit.* 4c⟩
⟨*Include the micro parser combinator definitions.* 4d⟩

This code is used in chunk 3a.

3c ⟨*Include the boolean type and values.* 3c⟩≡

```
#include <stdbool.h>
```

Defines:
bool, used in chunk 3a.
This code is used in chunk 3b.

4a ⟨*Include the standard I/O functions.* 4a⟩≡

```
#include <stdio.h>
```

Defines:
`printf`, never used.
This code is used in chunk 3b.

4b ⟨*Include the standard library definitions.* 4b⟩≡

```
#include <stdlib.h>
```

Defines:
`free`, used in chunk 3a.
This code is used in chunk 3b.

4c ⟨*Include the line editing functions from libedit.* 4c⟩≡

```
#include <editline/readline.h>
```

Defines:
`add_history`, used in chunk 1c.
`readline`, used in chunk 3a.
This code is used in chunk 3b.

4d ⟨*Include the micro parser combinator definitions.* 4d⟩≡

```
#include <mpc.h>
```

Defines:
`mpc_ast_delete`, used in chunk 2e.
`mpc_ast_print`, used in chunk 2e.
`mpc_cleanup`, used in chunk 2c.
`mpc_error_delete`, never used.
`mpc_error_print`, never used.
`mpc_parse`, used in chunk 2d.
`mpc_parser_t`, used in chunk 1d.
`mpc_result_t`, used in chunk 2g.
This code is used in chunk 3b.

Chunks

Index

Todo list