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 2f.

1d ⟨*Create some parsers.* 1d⟩≡

```
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:
`Expr`, used in chunks 1f and 2a.
`Lispy`, used in chunks 1 and 2.
`Number`, used in chunks 1f and 2a.
`Operator`, used in chunks 1f and 2a.
Uses `mpc_parser_t` 3f.
This code is used in chunk 2f.

Define the Lispy grammar.

1e ⟨*parsing.c* 1e⟩≡

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

This definition is continued in chunk 2f.
Root chunk (not used in this document).

1f ⟨*Define the parsers with the Lispy grammar.* 1f⟩≡

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

Uses `Expr` 1d, `Lispy` 1d, `Number` 1d, and `Operator` 1d.
This code is used in chunk 2f.

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

1b ⟨`input` *is nonempty* 1b⟩≡

```
input && *input
```

This code is used in chunk 2f.

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

```
add_history(input);
```

Uses `add_history` 3e.
This code is used in chunk 2f.

2a ⟨*Undefine and delete our parsers.* 2a⟩≡
```
mpc_cleanup(4, Number, Operator, Expr, Lispy);
```
Uses `Expr` 1d, `Lispy` 1d, `Number` 1d, `Operator` 1d, and `mpc_cleanup` 3f.
This code is used in chunk 2f.

2b ⟨*The input can be parsed as Lispy code.* 2b⟩≡
```
mpc_parse("<stdin>", input, Lispy, &res)
```
Uses `Lispy` 1d and `mpc_parse` 3f.
This code is used in chunk 2e.

2e ⟨*Attempt to parse the user input.* 2e⟩≡
```
mpc_result_t res;
if (⟨The input can be parsed as Lispy code. 2b⟩) {
    ⟨Print and delete the AST. 2c⟩
} else {
    ⟨Print and delete the error. 2d⟩
}
```
Uses `mpc_result_t` 3f.
This code is used in chunk 2f.

2f ⟨*parsing.c* 1e⟩+≡
```
⟨Include the necessary headers. 3a⟩


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

    ⟨Define the parsers with the Lispy grammar. 1f⟩

    ⟨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. 2e⟩
        }

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

    ⟨Undefine and delete our parsers. 2a⟩

    return 0;
}
```
Uses `bool` 3b, `free` 3d, and `readline` 3e.

2c ⟨*Print and delete the AST.* 2c⟩≡
```
mpc_ast_print(res.output);
mpc_ast_delete(res.output);
```
Uses `mpc_ast_delete` 3f and `mpc_ast_print` 3f.
This code is used in chunk 2e.

2d ⟨*Print and delete the error.* 2d⟩≡
```
mpc_err_print(res.error);
mpc_err_delete(res.error);
```
This code is used in chunk 2e.

Headers

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

⟨*Include the line editing functions from libedit.* 3e⟩
⟨*Include the micro parser combinator definitions.* 3f⟩

This code is used in chunk 2f.

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

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

Defines:
`bool`, used in chunk 2f.
This code is used in chunk 3a.

3c ⟨*Include the standard I/O functions.* 3c⟩≡

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

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

3d ⟨*Include the standard library definitions.* 3d⟩≡

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

Defines:
`free`, used in chunk 2f.
This code is used in chunk 3a.

3e ⟨*Include the line editing functions from libedit.* 3e⟩≡

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

Defines:
`add_history`, used in chunk 1c.
`readline`, used in chunk 2f.
This code is used in chunk 3a.

3f ⟨*Include the micro parser combinator definitions.* 3f⟩≡

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

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

Chunks

Index

Todo list