1. REPL. Uses editline to read lines and concatente them into a rope which `LossLess` parses and evaluates.

```
#include <histedit.h>
#include <stdio.h>
#include "lossless.h"
  ⟨ Function declarations 2 ⟩
  ⟨ Global variables 3 ⟩
```

2. ⟨ Function declarations 2 ⟩ ≡

```
  char *prompt(EditLine *);
```

This code is used in section 1.

3. Arbitrary history size limit can be made dynamic later.

```
#define HISTORY_SIZE 1000
⟨ Global variables 3 ⟩ ≡
  EditLine *E;
  History *H;
```

This code is used in section 1.

4. The main application — initialise editline & LossLess and process a line at a time.

```
#define INITIALISE "(do\n\n"
        "(define!␣(root-environment)␣list?␣((lambda␣()\n"
        "␣␣␣␣␣␣␣␣(define!␣(current-environment)␣-list?␣(lambda␣(OBJECT)\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(if␣(null?␣OBJECT)\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#t\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(if␣(pair?␣OBJECT)\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(-list?␣(cdr␣OBJECT))\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#f))))\n"
        "␣␣␣␣␣␣␣␣-list?)))\n"
        "\n\n"
        "(define!␣(root-environment)␣and␣((lambda␣()\n"
        "␣␣␣␣␣␣␣␣(define!␣(current-environment)␣-and␣(vov␣((ARGS␣vov\
            /args)␣(ENV␣vov/env))\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(arity!␣ARGS␣list?)\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(if␣(null?␣ARGS)\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#t\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(do␣␣␣␣␣(define!␣(current-environme\
            nt)␣ARG␣(eval␣(car␣ARGS)␣ENV))\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(if␣ARG␣(if␣(null?␣(cdr␣ARGS))\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ARG\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(eval␣(cons\
            ␣-and␣(cdr␣ARGS))␣ENV))\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#f)))))\n"
        "␣␣␣␣␣␣␣␣-and)))\n"
        "\n"
        "(define!␣(root-environment)␣or␣((lambda␣()\n"
        "␣␣␣␣␣␣␣␣(define!␣(current-environment)␣or␣(vov␣((ARGS␣vov/a\
            rgs)␣(ENV␣vov/env))\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(arity!␣ARGS␣list?)\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(if␣(null?␣ARGS)\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#f\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(do␣␣␣␣␣(define!␣(current-environme\
            nt)␣ARG␣(eval␣(car␣ARGS)␣ENV))\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(if␣ARG␣ARG␣(eval␣(cons␣or␣\
            (cdr␣ARGS))␣ENV))))))\n"
        "␣␣␣␣␣␣␣␣or)))\n"
        "\n"
        "(define!␣(root-environment)␣not␣(lambda␣(OBJECT)␣(if␣OBJECT␣#f␣#t)))\n"
        "\n"
        "(define!␣(root-environment)␣so␣(lambda␣(OBJECT)␣(if␣OBJECT␣#t␣#f)))\n"
        "\n"
        "(define!␣(root-environment)␣arity!␣(lambda␣ARGS\n"
        "␣␣␣␣␣␣␣␣#f))\n"
        "\n"
        "(define!␣(root-environment)␣maybe␣(lambda␣(PREDICATE)\n"
        "␣␣␣␣␣␣␣␣(lambda␣(OBJECT)\n"
        "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(or␣(null?␣OBJECT)␣(PREDICATE␣OBJECT)))))\n"
        "\n\n)"

  int main(int argc, char **argv)
  {
    HistEvent event;
    int length;
    const char *line;
    bool pending = false, valid;
    cell x;
    sigjmp_buf cleanup;
    Verror reason = LERR_NONE;
    ⟨ Initialise editline 5 ⟩
    mem_init( );    /* Initialise LossLess. */
    if (failure_p(reason = sigsetjmp(cleanup, 1))) goto die;
    lprint("Initialising...\n%s\n", INITIALISE);
    stack_push(NIL, &cleanup);
    SS(0, x = rope_new_buffer(false, false, INITIALISE, sizeof (INITIALISE) − 1, &cleanup));
    SS(0, x = lex_rope(x, &cleanup));
    valid = true;
    SS(0, x = parse(x, &valid, &cleanup));
    assert(valid);
    evaluate_program(SO(0), &cleanup);
    ACC = VOID;
    stack_push(NIL, &cleanup);
    do {⟨ Read and dispatch a line 6 ⟩} while (length);
    if (H ≠ Λ) history_end(H);
    el_end(E);
  }
```

5. ⟨ Initialise editline 5 ⟩ ≡

```
  E = el_init(argv[0], stdin, stdout, stderr);
  el_set(E, EL_PROMPT, &prompt);
  el_set(E, EL_EDITOR, "emacs");    /* TODO: use environment. */
  H = history_init( );
  if ((H = history_init( )) ≡ Λ) fprintf(stderr, "WARNING:␣could␣not␣initialise␣history\n");
  else {
    history(H, &event, H_SETSIZE, HISTORY_SIZE);
    el_set(E, EL_HIST, history, H);
  }
```

This code is used in section 4.

6. ⟨ Read and dispatch a line 6 ⟩ ≡

```
  line = el_gets(E, &length);
  if (failure_p(reason = sigsetjmp(cleanup, 1))) {
    switch (reason) {
    case LERR_UNCLOSED_OPEN: pending = true;
      break;
    default: die: fprintf(stderr, "FATAL␣%u:␣%s.\n", reason, Ierror[reason].message);
      abort();
    }
  }
  if (length > 0) {
  if (H ≠ Λ) history(H, &event, H_ENTER, line);
  SS(0, x = rope_new_buffer(false, false, line, length, &cleanup)); if (pending) {
      serial(lapi_User_Register(UNDEFINED), SERIAL_DETAIL, 12, NIL, Λ, &cleanup);
  SS(0, x = cons(SO(0), NIL, &cleanup));
  SS(0, x = cons(lapi_User_Register(UNDEFINED), x, &cleanup));
  SS(0, x = cons(symbol_new_const("rope/append"), x, &cleanup));
  evaluate_program(SO(0), &cleanup);
  SS(0, x = ACC);
#if 0
  cons ("rope/append", USERREG, x
      ... SS(0, x = rope_append(lapi_User_Register(UNDEFINED), x, &cleanup));
#endif
      } lapi_User_Register(x);
      x = lex_rope(x, &cleanup);
      SS(0, x);
      valid = true;
      x = parse(SO(0), &valid, &cleanup);
      if (valid) {
        ACC = VOID;
        evaluate_program(x, &cleanup);
        lapi_User_Register(NIL);
        pending = false;
      }
      else if (pair_p(lcdr(x)) ∧ fix_value(lcar(lcar(lcdr(x)))) ≡ LERR_SYNTAX ∧
            pair_p(lcdr(lcdr(x))) ∧ fix_value(lcar(lcar(lcdr(lcdr(x))))) ≡
            LERR_UNCLOSED_OPEN ∧ null_p(lcdr(lcdr(lcdr(x))))) {
        pending = true;
      }
      else {
        SS(0, x = lcdr(x));
        while (pair_p(x)) {
          printf("␣␣%d␣%s␣==␣", fix_value(lcar(lcar(x))), Ierror[fix_value(lcar(lcar(x)))].message);
          serial(lcar(x), SERIAL_DETAIL, 12, NIL, Λ, &cleanup);
          printf("\n");
          SS(0, x = lcdr(x));
        }
        printf("\n");
        lapi_User_Register(NIL);
        pending = false;
      }
      printf("DONE␣");
      serial(Accumulator, SERIAL_DETAIL, 12, NIL, Λ, &cleanup);
      printf("\n"); }
      else printf("\n");
```

This code is used in section 4.

7.
```
char *prompt(EditLine *e_unused)
{
  return "OK␣";
}
```

8. Index.

REPL