ebrew

what big PLT does not want you to know

who are we?

who are we?

ebrew is designed by Ted.

I have done much work on ebrew.

who are we?

ebrew is designed by Ted.

I have done much work on ebrew.

perhaps you would like to join us on this journey? ;)

ebrew syntax

- optional parentheses

```
(f x) x

(main) f f 10
```

```
(f x) x

(main) f (f 10)
```

```
(f x) x

(main) (f f 10)
```

```
(f x) x

(main) (f (f 10))
```

ebrew syntax

- optional parens
- default operators
  - `and  lhs  rhs`
  - `or  lhs  rhs`
  - `do  lhs  rhs`
  - `if  cond  then  else`
  - `for  name  init  expr`
  - `let  name  init  expr`
- context sensitive parser

ebrew syntax

- optional parens
- default operators
  - `and  lhs  rhs`
  - `or  lhs  rhs`
  - `do  lhs  rhs`
  - `if  cond  then  else`
  - `for  name  init  expr`
  - `let  name  init  expr`
- context sensitive parser

ebrew syntax

say you have this code:

```
(func var) other-func func var
```

ebrew syntax

say you have this code:

```
(func-2 x) x

(func-1 x) x

(func-0 x) func-1 func-2 x
```

ebrew syntax

say you have this code:

```
(func-2 x) x

(func-1 x) x
```

this line parses like the following lisp:

```
(func-0 x) func-1 func-2 x
```

```
(define (func-0 x)

    (func-1 (func-2 x)))
```

ebrew syntax

however if you had this code:

```
(func-2 x) x
```

the meaning of the last line changes.

```
(func-1 (f y) x) f x
```

this line parses like the following lisp:

```
(func-0 x) func-1 func-2 x
```

```
(define (func-0 x)
    (func-1 func-2 x))
```

ebrew syntax

- optional parens
- default operators
  - `and  lhs  rhs`
  - `or  lhs  rhs`
  - `do  lhs  rhs`
  - `if  cond  then  else`
  - `for  name  init  expr`
  - `let  name  init  expr`
- context sensitive parser
- LL1 to parse

ebrew tooling

- reference compiler
  - self hosted
  - bootstrapped gradually from chibicc in about 6-8 months
  - less than 1000 lines of code

ebrew tooling

- reference compiler
  - self hosted
  - bootstrapped gradually from chibicc in about 6-8 months
  - less than 1000 lines of code
- js hosted compiler and runtime
  - about 98% complete
  - runtime emulation of used linux syscalls
  - enough to bootstrap reference compiler

ebrew tooling

- visual studio code extension
  - easy to modify

ebrew tooling

- visual studio code extension
  - easy to modify
  - almost full parser for ebrew
    - basis for ebrew.js compiler

ebrew tooling

- visual studio code extension
  - easy to modify
  - almost full parser for ebrew
    - basis for ebrew.js compiler
  - contextual highlighting
    - functions calls distinct from function references
    - error recovery on bad parse

ebrew tooling

- visual studio code extension
  - easy to modify
  - almost full parser for ebrew
    - basis for ebrew.js compiler
  - contextual highlighting
    - functions calls distinct from function references
    - error recovery on bad parse
  - hover functionality
    - see the type of a function or variable
    - full type or simplified type

ebrew tooling

- visual studio code extension
  - easy to modify
  - almost full parser for ebrew
    - basis for ebrew.js compiler
  - contextual highlighting
    - functions calls distinct from function references
    - error recovery on bad parse
  - hover functionality
    - see the type of a function or variable
    - full type or simplified type
  - symbols and breadcrumbs
    - see current function
    - overview of ebrew source file

ebrew links

reference compiler: https://github.com/kerdek/ebrew

js compiler: https://github.com/shawsumma/ebrew.js

dlang transpiler to c: https://github.com/shawsumma/dbrew

vscode extension: https://github.com/shawsumma/ebrew-vscode