NAME

c2tc − A tiny compiler for the C2 language. Currently only provides simple code statistics.

SYNOPSIS

c2tc [**−heidofatA?**] [**−−help**] [**−−usage**] [**−−experiment**] [**−−ast0**] [**−−ast1**] [**−−test**] [**-f** [**−o** *NAME*]] [**−d** *DIR*] [**−−dir** *DIR*] [*TARGETS*]

DESCRIPTION

c2tc is a tiny compiler for the C2 language. It uses Daniel Holden's (orangeduck's) **micro parser combinators** library (see https://github.com/orangeduck/mpc for more info) for parsing. The version included in c2tc is modified to print index of each node relative to their parent node and to be very colorful. Since c2tc doesn't utilise an ad hoc parser but uses this library, the parsing efficiency seems to be bigger than **O(n)** but lower than **O(nˆ2)**. This is especially noticeable on files of great sizes (10k+ lines). Parsing currently takes the longest time of all the tasks the compiler performs. Speeding up parsing is on the imaginary TODO list.

For collections of dynamic sizes, c2tc mostly utilizes own implementation of vectors based on a tutorial written by Edd Mann (link: **http://eddmann.com/posts/implementing-a-dynamic-vector-array-in-c/**). The version of vectors provided in c2tc was originally implemented as a C2 paste-in library. A minimalistic implementation of a single-linked list is utilized by the included ooc library. The ooc library provides basic means for object-orientation, but remains mostly unused at the moment as the insides of functions are not being processed yet.

Error handling is facilitated by the **'throw'** library by **Joseph Werle <joseph.werle@gmail.com>**. In c2tc, **'throw'** is bundled inside the *error.c/h* files for the purpose of conservating spaces and avoiding littering the repository with a lot of tiny source files. The version of 'throw' included in c2tc has also been modified to use the *'log.h'* library (also created by Joseph Werle) to produce output to stderr.

Logging is handled by the afore-mentioned 'log.h' library by Joseph Werle. Currently, c2tc only logs into stdout, which is unlikely to change. Logging to a file can be easily done in the shell with pipes. c2tc tries to stay as small as possible and logging to files through a command-line option is a futile feature.

c2tc has a very small memory footprint. However, many pointers are intentionally not freed so there are memory leaks. Memory footprint of c2tc will most likely never reach sizes that would be threatening for common PCs. Because of the memory leaks, c2tc is **not** *recommended for embedded devices, but compiling and running it should be possible on any system that satisfies the dependencies.*

OPTIONS

−h, **−−help**
: Print help text.

−?, **−−usage**
: Print usage test.

−i, **−−info**
: Print info about each target found in the rustyfile.

−d, **−−dir** *path*
: Change directory before starting to look for recipe file.

−a, **−−ast0**
: Print the abstract syntax tree before right after parsing before any simplifications occur.

−A, **−−ast1**
: Print the abstract syntax tree after it has been simplified and cleaned up.

−o, **−−output** *name*
: Change name of the ad hoc target generated when in file-mode.

−e, **−−experiment**
: Enable experimental features. Most likely not noticeable by the user, but the function of this option changes often.

−f, **−−file**
: Switch to file-mode. File-mode means that input names are handled as filenames rather than as target names. c2tc generates an ad hoc target for these files, whose name can be changed by using the

-o option.

−t, **−−test**

Print test information. This also means handling files as tests rather than compilation units.

MODUS OPERANDI

1 c2tc reads command-line arguments. Unless the file-mode option is present, c2tc starts iteratively looking for recipe.txt through parent directories until it either reaches the root directory or finds it. The directory where recipe.txt was found becomes the current working directory of the program.

2 c2tc reads the recipe file, removes comments and empty lines. The rest is then parsed by c2tc's recipe reader, which, unlike C2 parser, is an ad hoc parser, and creates targets based on the data.

3 based on the information provided in the recipe file, c2tc locates source files of each targets and checks whether they are readable. When c2tc encounters an unreadable files an error is issued.

4 files are passed to the parsing part of the program. First comments are removed from the source (line numbers are kept - c2tc doesn't consider new−lines a part of the comment) then passed to C2 parser.

5 C2 parser then parses the file and returns a raw AST. The Abstract Syntax Tree is then passed to the clean-up functions.

6 First, unnecessary tokens are deleted from the AST. That means keywords, brackets etc. Secondly, AST node tags are simplified and semi-resolved - special characters from node's tag are removed, name is combined and built-in grammatical rules are combined with their respective parent rules (e.g. **'head|module|>'** -> **'module'**, **'ident|regex'** -> **'ident'**).

7 The trees are then passed to the analyser. The analyser and his friends are currently the only part of c2tc that isn't written in C, but in Rust (albeit with unsafe features). The analyser reads each tree and creates objects that contain data about several types of symbols in the language (currently import statements, user−defined types and functions).

8 The objects are read and colorful output is produced to stdout.