libcdict: fast dictionaries in C

Robert G. Izzard [1]¶, **David D. Hendriks** [1], **and Daniel P. Nemergut** [1]

1 Department of Physics, School of Mathematics and Physics, University of Surrey, Guildford, GU2 7XH, Surrey, UK ¶ Corresponding author

Summary

A common requirement in science is to store and share large sets of simulation data in an efficient, nested, flexible and human-readable way. Such datasets contain number counts and distributions, i.e. histograms and maps, of arbitrary dimension and variable type, e.g. floating-point number, integer or character string. Modern high-level programming languages like Perl and Python have associated arrays, knowns as dictionaries or hashes, respectively, to fulfil this storage need. Low-level languages used more commonly for fast computational simulations, such as C and Fortran, lack this functionality. We present a libcdict, a C dictionary library, to solve this problem. Libcdict provides C and Fortran application programming interfaces (APIs) to native dictionaries, called cdicts, and functions for cdict to load and save these as JSON and hence for easy interpretation in other software and languages like Perl, Python and R.

Statement of need

Users of high-level languages such as Perl or Python have access to associated-array data structures through dictionaries and hashes, respectively. These allow arbitrary data types to be stored in array-like structures. These are in turn accessed through key-value pairs which allow the value to be a further, nested associated array, allowing arbitrary nesting of data. Compiled low-level languages, like C and Fortran, are more suited to high-speed and repeated calculations typical in science. These languages lack native associated-array functionality. While there are pure hash-table solutions out there, such as glib (Glib, 2022) and uthash (Hansen, 2022), these do not combine a simple API for setting and adding to nested structures, a small library footprint, fast input and output, and standardised JSON output to easily interface with other languages and tools. libcdict provides an API for such functionality which allows cdicts to be nested in cdicts, hence arbitrarily-nested dictionaries of variables in C just as in Perl or Python.

libcdict is written in C and provides an API through a set of C macros. Nested cdict structures have values in them set with a single line of code. libcdict has been used for the last year in the binary_c single- and binary-star population nucleosynthesis framework (Izzard et al., 2004, 2006, 2009, 2018). Recent works (Hendriks & Izzard, 2023b; Izzard & Jermyn, 2023; Mirouh et al., 2023; Yates et al., 2023) compute the evolution of millions of single- and binary-stellar systems in only a few hours using its binary_c-python Python frontend (Hendriks & Izzard, 2023a). We provide libcdict as open-source code on Gitlab subject to the GPL3. libcdict also has a comprehensive test suite run through its configuration program cdict-config.

Using libcdict

`libcdict` is flexible but pragmatic. Keys to `cdicts` can be any C scalar or pointer. Values can be scalars, pointers, arrays or other `cdicts`, but arrays must be of a single C type. Values can store metadata of arbitrary type. Pointer values are optionally garbage collected when a `cdict` is freed. A set of API macros provides simple nesting facilities so that placing a value in a nested location given a list of keys is a simple task for the C programmer. Issues such as C variable typing are automatically handled for the user.

Variables are internally hashed using uthash (Hansen, 2022). `libcdict` provides a custom JSON output function and inputs JSON using jsmn (Zaitsev, 2022). Floating-point input and output uses `fast double parser` (Lemire, 2021, 2022) and Ryū (Adams, 2018, 2019, 2022), respectively, both of which are considerably faster than equivalent C library functions. `libdict` allows customizable floating-point accuracy for output and when comparing floating-point numbers, e.g. during sorting of key or variable lists.

Installation uses meson (Pakkanen, 2022) and `ninja` (Martin, 2022). `libcdict` has been tested with the GCC (10.3.0) and Clang (12.0.0) compilers.

`libcdict` in stellar-population statistics calculations

`libcdict` was developed to solve the problem of storing statistics in stellar-population calculations in `binary_c`. When evolving a population of millions, sometimes billions, of stars, each for thousands of time steps, enormous amounts of data are computed. It is impractical to output these data every time step as these are typically $\sim 10^6 \times 10^4 = 10^{10}$ lines, each of which can easily be $\sim 1\,\mathrm{KB}$ long. The data from each star could be sent to a Perl or Python front-end which merges them into a dictionary of population statistics. This communication between programming languages involves significant overhead which compares similarly to the runtime of the stellar code itself thus greatly increases runtime and cost.

To overcome this problem, `binary_c` internally generates an associative-array `cdict` in native C. This `cdict`, and the stellar statistics it contains, is filled inside the `binary_c` simulation as each star is simulated. Generation of the stellar-population data in the `cdict` is efficient because it is only in C and communication with the frontend (Python) code is kept to a minimum. The `cdict`'s dataset is output *only once*, as human-readable JSON easily understood by Perl or Python, at the end of the simulation. Large simulations are often split across clusters of machines using `binary_c-python`. The data from each run are stored as JSON chunks then merged in Python when the final run completes. The overhead involved in this joining is small compared to the effort of simulating the stars: the goal of `libcdict` has thus been achieved.

We provide an interactive example made with `binary_c` and `binary_c-python` using `libcdict` in its examples directory (Izzard, 2022). The `libcdict` JSON output of a Hertzsprung-Russell diagram, the most important diagnostic plot in stellar astrophysics, is plotted using Bokeh (Bokeh Development Team, 2014; Bokeh GitHub, 2022) to provide immediate access to nested data sets.

Acknowledgements

References

Adams, U. (2018). Ryū: Fast float-to-string conversion. *SIGPLAN Not.*, *53*(4), 270–282. https://doi.org/10.1145/3296979.3192369

Adams, U. (2019). Ryū revisited: Printf floating point conversion. In *Proceedings of the ACM on Programming Languages* (OOPSLA; Vol. 3, pp. 1–23). Association for Computing Machinery (ACM). https://doi.org/10.1145/3360595

Adams, U. (2022). Ryū & Ryū Printf. In *GitHub repository*. https://github.com/ulfjack/ryu

Bokeh Development Team. (2014). *Bokeh: Python library for interactive visualization*. http://www.bokeh.pydata.org

Bokeh GitHub. (2022). Bokeh. In *GitHub repository*. GitHub. https://github.com/bokeh/

Glib. (2022). GLIB hash tables. In GLIB *official documentation*. Gitlab. https://docs.gtk.org/glib/index.html

Hansen, T. D. (2022). uthash: A hash table for C structures. In *GitHub repository*. https://troydhanson.github.io/uthash/

Hendriks, D. D., & Izzard, R. G. (2023a). Binary_c-python: A python-based stellar population synthesis tool and interface to binary_c. *Journal of Open Source Software*, *8*(85), 4642. https://doi.org/10.21105/joss.04642

Hendriks, D. D., & Izzard, R. G. (2023b). Mass-stream trajectories with non-synchronously rotating donors. *524*(3), 4315–4332. https://doi.org/10.1093/mnras/stad2077

Izzard, R. G. (2022). LIBCDICT examples. In *Gitlab.com repository*. Gitlab.com. https://doi.org/10.5281/zenodo.10276619

Izzard, R. G., Dray, L. M., Karakas, A. I., Lugaro, M., & Tout, C. A. (2006). Population nucleosynthesis in single and binary stars. I. Model. *Astronomy and Astrophysics*, *460*, 565–572. https://doi.org/10.1051/0004-6361:20066129

Izzard, R. G., Glebbeek, E., Stancliffe, R. J., & Pols, O. R. (2009). Population synthesis of binary carbon-enhanced metal-poor stars. *Astronomy and Astrophysics*, *508*, 1359–1374. https://doi.org/10.1051/0004-6361/200912827

Izzard, R. G., & Jermyn, A. S. (2023). Circumbinary discs for stellar population models. *521*(1), 35–50. https://doi.org/10.1093/mnras/stac2899

Izzard, R. G., Preece, H., Jofre, P., Halabi, G. M., Masseron, T., & Tout, C. A. (2018). Binary stars in the Galactic thick disc. *Monthly Notices of the Royal Astronomical Society*, *473*, 2984–2999. https://doi.org/10.1093/mnras/stx2355

Izzard, R. G., Tout, C. A., Karakas, A. I., & Pols, O. R. (2004). A new synthetic model for asymptotic giant branch stars. *Monthly Notices of the Royal Astronomical Society*, *350*, 407–426. https://doi.org/10.1111/j.1365-2966.2004.07446.x

Lemire, D. (2021). Number Parsing at a Gigabyte per Second. *arXiv e-Prints*, arXiv:2101.11408. https://doi.org/10.1002/spe.2984

Lemire, D. (2022). fast_double_parser: $4\times$ faster than strtod. In *GitHub repository*. https://github.com/lemire/fast_double_parser

Martin, E. (2022). Ninja, a small build system with a focus on speed. In *GitHub repository*. GitHub. https://github.com/ninja-build/ninja

Mirouh, G. M., Hendriks, D. D., Dykes, S., Moe, M., & Izzard, R. G. (2023). Detailed equilibrium and dynamical tides: impact on circularization and synchronization in open clusters. *524*(3), 3978–3999. https://doi.org/10.1093/mnras/stad2048

Pakkanen, J. (2022). The Meson build system. In *GitHub repository*. GitHub. https://github.com/mesonbuild/meson

Yates, R. M., Hendriks, D., Vijayan, A. P., Izzard, R. G., Thomas, P. A., & Das, P. (2023). The impact of binary stars on the dust and metal evolution of galaxies. https://doi.org/10.1093/mnras/stad3419

Zaitsev, S. (2022). jsmn, a minimalistic JSON parser in C. In *GitHub repository*. https://github.com/zserge/jsmn