The Journal of Open Source Software

GPP, the Generic Preprocessor

Tristan Miller[1] **and Denis Auroux**[2]

1 Austrian Research Institute for Artificial Intelligence **2** Department of Mathematics, Harvard University

DOI: 10.21105/joss.02400

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @Smattr
- @drj11

Submitted: 28 May 2020
Published: 28 July 2020

Summary

In computer science, a *preprocessor* (or *macro processor*) is a tool that programatically alters its input, typically on the basis of inline annotations, to produce data that serves as input for another program. Preprocessors are used in software development and document processing workflows to translate or extend programming or markup languages, as well as for conditional or pattern-based generation of source code and text. Early preprocessors were relatively simple string replacement tools that were tied to specific programming languages and application domains, and while these have since given rise to more powerful, general-purpose tools, these often require the user to learn and use complex macro languages with their own syntactic conventions. In this paper, we present GPP, an extensible, general-purpose preprocessor whose principal advantage is that its syntax and behaviour can be customized to suit any given preprocessing task. This makes GPP of particular benefit to research applications, where it can be easily adapted for use with novel markup, programming, and control languages.

Background

Preprocessors date back to the mid-1950s, when they were used to extend individual assembly languages with constructs that would later be found in high-level programming languages (Layzell, 1985). These languages, in turn, fostered the development of yet more special-purpose preprocessors aimed at providing even higher-level constructs, such as conditional loops and other control structures in FORTRAN (Meissner, 1975) and COBOL (Triance, 1980). The need for generalized, language-independent tools was eventually recognized (McIlroy, 1960), leading to the development of general-purpose preprocessors such as GPM (Strachey, 1965) and ML/I (Brown, 1967).

By the end of the 1960s, preprocessors had attracted a considerable amount of attention, by computing theorists and practitioners alike, and their use in software engineering had expanded beyond the augmentation and adaptation of programming languages. A survey paper by Brown (1969) identified four broad application areas: language extension, systematic searching and editing of source code, translation between programming languages, and code generation (i.e., simplifying the writing of highly repetitive code, parameterizing a program by substituting compile-time constants, or producing variants of a program by conditionally including certain statements or modules). While the first three of these application areas have largely been rendered obsolete by today's integrated development environments and expressive, feature-rich programming languages, implementing software variability with language-specific and general-purpose preprocessors remains commonplace (Apel, Batory, Kästner, & Saake, 2013; Kästner, Apel, Thüm, & Saake, 2012).

Text processing became another main application area for preprocessors, in particular to generate documents on the basis of user-specified conditions or patterns, and to convert between document markup languages (Walden, 2014). The earliest such uses were ad-hoc

repurposings of programming language–specific preprocessors to operate on human-readable texts (Keese, 1964; Stallman & Weinberg, 2020); these were soon supplanted by text-specific macro languages such as TRAC (Mooers & Deutsch, 1965), which were positioned as tools for stenographers and other writing professionals. More recently it has been common to use general-purpose preprocessors (Mailund, 2019; Pesch, 1992).

Statement of Need

Criticism of preprocessors commonly focuses on the idiosyncratic languages they employ for their own built-in directives and for users to define and invoke macros. The languages of early preprocessors were derided as "clumsy and restrictive" (Layzell, 1985) and "hard to read" (Brown, 1969), and even modern preprocessors are sometimes attacked for relying on "the clumsiness of a separate language of limited expressiveness" (Ernst, Badros, & Notkin, 2002) or, at the other extreme, for being overly complicated, quirky, opaque, or hard to learn, even for experienced programmers and markup users (Ernst et al., 2002; Paddon, 1993; Pesch, 1992).

Our general-purpose preprocessor, GPP, avoids these issues by providing a lightweight but flexible macro language whose syntax can be customized by the user. The tool's built-in presets allow its directives to be made to resemble those of many popular languages, including HTML and LaTeX. This greatly reduces the learning curve for GPP when it is used with these languages, eliminates the cognitive burden of repeatedly "mode switching" between source and preprocessor syntax when reading or composing, and allows existing syntax highlighters and other tools to process GPP directives with little or no further configuration. Furthermore, users are not limited to using these presets, but can fully define their own syntax for GPP directives and macros. This makes GPP particularly attractive for use in research and development, where its syntax can be readily adapted to match bespoke programming and markup languages.

GPP's independence from any one programming or markup language makes it more versatile than the C Preprocessor, which was formerly "abused" as a general text processor and is still sometimes (inappropriately) used for non-C applications (Stallman & Weinberg, 2020). While GPP is less powerful than m4 (Seindal, Pinard, Vaughan, & Blake, 2016), it is arguably more flexible, and supports all the basic operations expected of a modern, high-level preprocessing system, including conditional tests, arithmetic evaluation, and POSIX-style wildcard matching ("globbing"). In addition to macros, GPP understands comments and strings, whose syntax and behaviour can also be widely customized to fit any particular purpose.

GPP in research

GPP has already been integrated into a number of third-party projects in basic and applied research. These include the following:

- The Waveform Definition Language (WDL) is Caltech Optical Observatories' C-like language for programming astronomical research cameras. WDL uses GPP to preprocess configuration files containing signals and parameters specific to the camera controllers, flags setting the devices' operating modes and image properties, and timing rules. According to the developers, GPP was chosen over the C Preprocessor "for added flexibility and to avoid some C-like limitations" (Kaye, Smith, Hale, & Mao, 2017).
- XSB is a research-oriented, commercial-grade logic programming system and Prolog compiler. The developers chose to make GPP XSB's default preprocessor because it "maintains a high degree of compatibility with the C preprocessor, but is more suitable for processing Prolog programs" (Swift et al., 2017).

- C-Control Pro is a family of electronic microcontrollers produced by Conrad Electronic; they are specifically designed for industrial and automotive applications. The official software development kit includes a modified version of GPP for use with the products' BASIC and Compact-C programming languages (Schirm & Sprenger, 2007).
- SUS is a tool that allows system administrators to exercise fine-grained control over how users can run commands with elevated privileges. It has a sophisticated control file syntax that is preprocessed with GPP (Gray, 2001).

Apart from these uses, GPP is occasionally cited as previous or related work in scholarly publications on metaprogramming or compile-time variability of software (Apel et al., 2013; Baxter & Mehlich, 2001; Behringer, 2017; Blendinger, 2010; Dreiling, 2010; Kästner et al., 2012; Lotoreychik & Shopyrin, 2006; Zmiry, 2016).

Acknowledgments

Tristan Miller is supported by the Austrian Science Fund (FWF) under project M 2625-N31. Denis Auroux is partially supported by NSF grant DMS-1937869 and by Simons Foundation grant #385573. The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry for Science, Research and Economy.

References

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013). Classic, tool-driven variability mechanisms. In *Feature-oriented software product lines*. Berlin/Heidelberg: Springer-Verlag. doi:10.1007/978-3-642-37521-7_5

Baxter, I. D., & Mehlich, M. (2001). Preprocessor conditional removal by simple partial evaluation. In *Proceedings of the 8th Working Conference on Reverse Engineering* (pp. 281–290). IEEE. doi:10.1109/WCRE.2001.957833

Behringer, B. (2017, July). *Projectional editing of software product lines – The PEoPL approach* (PhD thesis). Faculty of Sciences, Technology; Communication, Université de Luxembourg.

Blendinger, F. (2010, August). *A filesystem-based approach to support product line development with editable views* (Diploma Thesis). Department of Computer Sciences 4, Friedrich-Alexander University Erlangen-Nuremberg.

Brown, P. J. (1967). The ML/I macro processor. *Communications of the ACM*, *10*(10), 618–623. doi:10.1145/363717.363746

Brown, P. J. (1969). A survey of macro processors. *Annual Review in Automatic Programming*, *6*, 37–88. doi:10.1016/0066-4138(69)90001-9

Dreiling, A. (2010, July). *Feature Mining: Semiautomatische Transition von (Alt-)Systemen zu Software-Produktlinien* (Diploma thesis). Fakultät für Informatik, Institut für Technische und Betriebliche Informationssysteme, Otto-von-Guericke-Universität Magdeburg.

Ernst, M. D., Badros, G. J., & Notkin, D. (2002). An empirical analysis of C Preprocessor use. *IEEE Transactions on Software Engineering*, *28*(12), 1146–1170. doi:10.1109/TSE.2002.1158288

Gray, P. D. (2001). SUS – An object reference model for distributing UNIX super user privileges. In *Proceedings of the LISA 2001 15th Systems Administration Conference* (pp. 15–18). The USENIX Association.

Kaye, S., Smith, R., Hale, D., & Mao, P. (2017). *Waveform Definition Language*. Pasadena, CA: Caltech Optical Observatories.

Kästner, C., Apel, S., Thüm, T., & Saake, G. (2012). Type checking annotation-based product lines. *ACM Transactions on Software Engineering and Methodology*, *21*(3), 14:1–14:39. doi:10.1145/2211616.2211617

Keese, W. M., Jr. (1964). *A note on automatic generation of documentation by macro assemblers* (Technical memorandum No. TM-64-1031-1). Washington, DC: Bellcom, Inc.

Layzell, P. J. (1985). The history of macro processors in programming language extensibility. *The Computer Journal*, *28*(1), 29–33. doi:10.1093/comjnl/28.1.29

Lotoreychik, V. Y., & Shopyrin, D. G. (2006). Metaprogrammirovaniye na osnove tekstovogo preprotsessora [Text preprocessor–based metaprogramming]. *Nauchno-Tehnicheskii Vestnik Informatsionnykh Tekhnologii, Mekhaniki i Optiki [Scientific and Technical Journal of Information Technologies, Mechanics and Optics]*, *6*(2), 57–65.

Mailund, T. (2019). Preprocessing. In *Introducing Markdown and Pandoc: Using markup language and document converter*. Berkeley, CA: Apress. doi:10.1007/978-1-4842-5149-2_10

McIlroy, M. D. (1960). Macro instruction extensions of compiler languages. *Communications of the ACM*, *3*(4), 214–220. doi:10.1145/367177.367223

Meissner, L. P. (1975). On extending Fortran control structures to facilitate structured programming. *SIGPLAN Notices*, *10*(9), 19–30. doi:10.1145/987316.987320

Mooers, C. N., & Deutsch, L. P. (1965). TRAC, a text-handling language. In L. Winner (Ed.), *ACM '65: Proceedings of the 20th National Conference* (pp. 229–246). New York: Association for Computing Machinery. doi:10.1145/800197.806048

Paddon, M. (1993). Shake: A portable tool for generating Makefiles. In *AUUG '93 Conference proceedings* (pp. 145–156). Kensington, NSW, Australia: AUUG Inc.

Pesch, R. H. (1992). Configurable manuals. In *Conference record on Crossing Frontiers* (pp. 776–780). doi:10.1109/IPCC.1992.673146

Schirm, R., & Sprenger, P. (2007). Der Preprozessor. In *Messen, Steuern und Regeln mit C-Control Pro: Praxisanwendungen, Schaltungstechnik und Programmierung*. Poing, Germany: Franzis. ISBN: 978-3-7723-4097-0

Seindal, R., Pinard, F., Vaughan, G. V., & Blake, E. (2016). *GNU m4, version 1.4.18*. Free Software Foundation.

Stallman, R. M., & Weinberg, Z. (2020). Overview. In *The C Preprocessor* (GCC 10.1.0.). Free Software Foundation.

Strachey, C. (1965). A general purpose macrogenerator. *The Computer Journal*, *8*(3), 225–241. doi:10.1093/comjnl/8.3.225

Swift, T., Warren, D. S., Sagonas, K., Freire, J., Rao, P., Cui, B., Johnson, E., et al. (2017). *The XSB System, version 3.8.x, volume 1: Programmer's manual*.

Triance, J. M. (1980). Structured programming in COBOL—the current options. *The Computer Journal*, *23*(3), 194–200. doi:10.1093/comjnl/23.3.194

Walden, D. (2014). Macro memories, 1964–2013. *TUGboat: The Communications of the TeX Users Group*, *35*(1), 99–110.

Zmiry, I. E. (2016, April). *Lola 0.064: A programming language for augmenting programming languages* (Master's thesis). Technion – Israel Institute of Technology.