WIKIPEDIA

Preprocessor

In computer science, a **preprocessor** is a program that processes its input data to produce output that is used as input to another program. The output is said to be a **preprocessed** form of the input data, which is often used by some subsequent programs like compilers. The amount and kind of processing done depends on the nature of the preprocessor; some preprocessors are only capable of performing relatively simple textual substitutions and macro expansions, while others have the power of full-fledged programming languages.

A common example from computer programming is the processing performed on source code before the next step of compilation. In some computer languages (e.g., C and PL/I) there is a phase of translation known as *preprocessing*. It can also include macro processing, file inclusion and language extensions.

Contents

Lexical preprocessors

Lexical preprocessors are the lowest-level of preprocessors as they only require lexical analysis, that is, they operate on the source text, prior to any parsing, by performing simple substitution of tokenized character sequences for other tokenized character sequences, according to user-defined rules. They typically perform macro substitution, textual inclusion of other files, and conditional compilation or inclusion.

C preprocessor

The most common example of this is the C preprocessor, which takes lines beginning with '#' as directives. Because it knows nothing about the underlying language, its use has been criticized and many of its features built directly into other languages. For example, macros replaced with aggressive inlining and templates, includes with compile-time imports (this requires the preservation of type information in the object code, making this feature impossible to retrofit into a language); conditional compilation is effectively accomplished with `if-then-else` and dead code elimination in some languages. However, a key point to remember is that all preprocessor directives should start on a new line.

Other lexical preprocessors

Other lexical preprocessors include the general-purpose m4, most commonly used in cross-platform build systems such as autoconf, and GEMA, an open source macro processor which operates on patterns of context.

Syntactic preprocessors

Syntactic preprocessors were introduced with the Lisp family of languages. Their role is to transform syntax trees according to a number of user-defined rules. For some programming languages, the rules are written in the same language as the program (compile-time reflection). This is the case with Lisp and OCaml. Some other languages rely on a fully external language to define the transformations, such as the XSLT preprocessor for XML, or its statically typed counterpart CDuce.

Syntactic preprocessors are typically used to customize the syntax of a language, extend a language by adding new primitives, or embed a domain-specific programming language (DSL) inside a general purpose language.

Customizing syntax

A good example of syntax customization is the existence of two different syntaxes in the Objective Caml programming language.[1] Programs may be written indifferently using the "normal syntax" or the "revised syntax", and may be pretty-printed with either syntax on demand.

Similarly, a number of programs written in OCaml customize the syntax of the language by the addition of new operators.

Extending a language

The best examples of language extension through macros are found in the Lisp family of languages. While the languages, by themselves, are simple dynamically typed functional cores, the standard distributions of Scheme or Common Lisp permit imperative or object-oriented programming, as well as static typing. Almost all of these features are implemented by syntactic preprocessing, although it bears noting that the "macro expansion" phase of compilation is handled by the compiler in Lisp. This can still be considered a form of preprocessing, since it takes place before other phases of compilation.

Specializing a language

One of the unusual features of the Lisp family of languages is the possibility of using macros to create an internal DSL. Typically, in a large Lisp-based project, a module may be written in a variety of such minilanguages, one perhaps using a

SQL-based dialect of Lisp, another written in a dialect specialized for GUIs or pretty-printing, etc. Common Lisp's standard library contains an example of this level of syntactic abstraction in the form of the LOOP macro, which implements an Algol-like minilanguage to describe complex iteration, while still enabling the use of standard Lisp operators.

The MetaOCaml preprocessor/language provides similar features for external DSLs. This preprocessor takes the description of the semantics of a language (i.e. an interpreter) and, by combining compile-time interpretation and code generation, turns that definition into a compiler to the OCaml programming language—and from that language, either to bytecode or to native code.

General purpose preprocessor

Most preprocessors are specific to a particular data processing task (e.g., compiling the C language). A preprocessor may be promoted as being *general purpose*, meaning that it is not aimed at a specific usage or programming language, and is intended to be used for a wide variety of text processing tasks.

M4 is probably the most well known example of such a general purpose preprocessor, although the C preprocessor is sometimes used in a non-C specific role. Examples:

- using C preprocessor for JavaScript preprocessing.[2]
- using C preprocessor for devicetree processing within the Linux kernel.[3]
- using M4 (see on-article example) or C preprocessor[4] as a template engine, to HTML generation.
- imake, a make interface using the C preprocessor, written for the X Window System but now deprecated in favour of automake.
- grompp, a preprocessor for simulation input files for GROMACS (a fast, free, open-source code for some problems in computational chemistry) which calls the system C preprocessor (or other preprocessor as determined by the simulation input file) to parse the topology, using mostly the #define and #include mechanisms to determine the effective topology at grompp run time.
- using GPP (http://en.nothingisreal.com/wiki/GPP) for preprocessing markdown files[5]

See also

- Directive (programming)
- Metaprogramming
- Macros
- Post processor
- Transpiler
- Sass (stylesheet language)
- Stylus (stylesheet language)
- Less (stylesheet language)
- Snippet management