GASP, an assembly preprocessor

for GASP version 1

March 1994

Roland Pesch

Cygnus Support

1 What is GASP?

The primary purpose of the GNU assembler is to assemble the output of other programs—notably compilers. When you have to hand-code specialized routines in assembly, that means the GNU assembler is an unfriendly processor: it has no directives for macros, conditionals, or many other conveniences that you might expect.

In some cases you can simply use the C preprocessor, or a generalized preprocessor like M4; but this can be awkward, since none of these things are designed with assembly in mind.

GASP fills this need. It is expressly designed to provide the facilities you need with hand-coded assembly code. Implementing it as a preprocessor, rather than part of the assembler, allows the maximum flexibility: you can use it with hand-coded assembly, without paying a penalty of added complexity in the assembler you use for compiler output.

Here is a small example to give the flavor of GASP. This input to GASP

```
        .MACRO  saveregs from=8 to=14
count   .ASSIGNA \from
        ! save r\from..r\to
        .AWHILE  \&count LE \to
        mov     r\&count,@-sp
count   .ASSIGNA  \&count + 1
        .AENDW
        .ENDM

        saveregs from=12

bar:    mov     #H'dead+10,r0
foo     .SDATAC "hello"<10>
        .END
```

generates this assembly program:

```
        ! save r12..r14
        mov     r12,@-sp
        mov     r13,@-sp
        mov     r14,@-sp

bar:    mov     #57005+10,r0
foo:    .byte   6,104,101,108,108,111,10
```

2 Command Line Options

The simplest way to use GASP is to run it as a filter and assemble its output. In Unix and its ilk, you can do this, for example:

```
$ gasp prog.asm | as -o prog.o
```

Naturally, there are also a few command-line options to allow you to request variations on this basic theme. Here is the full set of possibilities for the GASP command line.

```
gasp  [ -a | --alternate ]
      [ -c char | --commentchar char ]
      [ -d | --debug ]  [ -h | --help ] [ -M | --mri ]
      [ -o outfile | --output outfile ]
      [ -p | --print ]  [ -s | --copysource ]
      [ -u | --unreasonable ]  [ -v | --version ]
      infile ...
```

infile ...
: The input file names. You must specify at least one input file; if you specify more, GASP preprocesses them all, concatenating the output in the order you list the *infile* arguments.

Mark the end of each input file with the preprocessor command `.END`. See Section 3.7 [Miscellaneous commands], page 12.

`-a`
`--alternate`
: Use alternative macro syntax. See Section 3.9 [Alternate macro syntax], page 15, for a discussion of how this syntax differs from the default GASP syntax.

`-c` '*char*'
`--commentchar` '*char*'
: Use *char* as the comment character. The default comment character is '`!`'. For example, to use a semicolon as the comment character, specify '`-c ';'`' on the GASP command line. Since assembler command characters often have special significance to command shells, it is a good idea to quote or escape *char* when you specify a comment character.

For the sake of simplicity, all examples in this manual use the default comment character '`!`'.

`-d`
`--debug`
: Show debugging statistics. In this version of GASP, this option produces statistics about the string buffers that GASP allocates internally. For each defined buffersize *s*, GASP shows the number of strings *n* that it allocated, with a line like this:

```
strings size s : n
```

GASP displays these statistics on the standard error stream, when done preprocessing.

`-h`
`--help`
: Display a summary of the GASP command line options.

`-M`
`--mri` Use MRI compatibility mode. Using this option causes GASP to accept the syntax and pseudo-ops used by the Microtec Research `ASM68K` assembler.

`-o` *outfile*
`--output` *outfile*
Write the output in a file called *outfile*. If you do not use the '`-o`' option, GASP writes its output on the standard output stream.

`-p`
`--print` Print line numbers. GASP obeys this option *only* if you also specify '`-s`' to copy source lines to its output. With '`-s -p`', GASP displays the line number of each source line copied (immediately after the comment character at the beginning of the line).

`-s`
`--copysource`
Copy the source lines to the output file. Use this option to see the effect of each preprocessor line on the GASP output. GASP places a comment character ('`!`' by default) at the beginning of each source line it copies, so that you can use this option and still assemble the result.

`-u`
`--unreasonable`
Bypass "unreasonable expansion" limit. Since you can define GASP macros inside other macro definitions, the preprocessor normally includes a sanity check. If your program requires more than 1,000 nested expansions, GASP normally exits with an error message. Use this option to turn off this check, allowing unlimited nested expansions.

`-v`
`--version`
Display the GASP version number.

3 Preprocessor Commands

GASP commands have a straightforward syntax that fits in well with assembly conventions. In general, a command extends for a line, and may have up to three fields: an optional label, the command itself, and optional arguments to the command. You can write commands in upper or lower case, though this manual shows them in upper case. See Section 3.8 [Details of the GASP syntax], page 13, for more information.

3.1 Conditional assembly

The conditional-assembly directives allow you to include or exclude portions of an assembly depending on how a pair of expressions, or a pair of strings, compare.

The overall structure of conditionals is familiar from many other contexts. `.AIF` marks the start of a conditional, and precedes assembly for the case when the condition is true. An optional `.AELSE` precedes assembly for the converse case, and an `.AENDI` marks the end of the condition.

You may nest conditionals up to a depth of 100; GASP rejects nesting beyond that, because it may indicate a bug in your macro structure.

Conditionals are primarily useful inside macro definitions, where you often need different effects depending on argument values. See Section 3.4 [Defining your own directives], page 8, for details about defining macros.

`.AIF` *expra cmp exprb*
`.AIF "`*stra*`"` *cmp* `"`*strb*`"`

The governing condition goes on the same line as the `.AIF` preprocessor command. You may compare either two strings, or two expressions.

When you compare strings, only two conditional *cmp* comparison operators are available: '`EQ`' (true if *stra* and *strb* are identical), and '`NE`' (the opposite).

When you compare two expressions, *both expressions must be absolute* (see Section 3.8.4 [Arithmetic expressions in GASP], page 14). You can use these *cmp* comparison operators with expressions:

`EQ` Are *expra* and *exprb* equal? (For strings, are *stra* and *strb* identical?)

`NE` Are *expra* and *exprb* different? (For strings, are *stra* and *strb* different?

`LT` Is *expra* less than *exprb*? (Not allowed for strings.)

`LE` Is *expra* less than or equal to *exprb*? (Not allowed for strings.)

`GT` Is *expra* greater than *exprb*? (Not allowed for strings.)

`GE` Is *expra* greater than or equal to *exprb*? (Not allowed for strings.)

`.AELSE` Marks the start of assembly code to be included if the condition fails. Optional, and only allowed within a conditional (between `.AIF` and `.AENDI`).

`.AENDI` Marks the end of a conditional assembly.

3.2 Repetitive sections of assembly

Two preprocessor directives allow you to repeatedly issue copies of the same block of assembly code.

`.AREPEAT` *aexp*
`.AENDR`
: If you simply need to repeat the same block of assembly over and over a fixed number of times, sandwich one instance of the repeated block between `.AREPEAT` and `.AENDR`. Specify the number of copies as *aexp* (which must be an absolute expression). For example, this repeats two assembly statements three times in succession:

```
        .AREPEAT        3
        rotcl   r2
        div1    r0,r1
        .AENDR
```

`.AWHILE` *expra cmp exprb*
`.AENDW`
`.AWHILE` *stra cmp strb*
`.AENDW`
: To repeat a block of assembly depending on a conditional test, rather than repeating it for a specific number of times, use `.AWHILE`. `.AENDW` marks the end of the repeated block. The conditional comparison works exactly the same way as for `.AIF`, with the same comparison operators (see Section 3.1 [Conditional assembly], page 5).

  Since the terms of the comparison must be absolute expression, `.AWHILE` is primarily useful within macros. See Section 3.4 [Defining your own directives], page 8.

You can use the `.EXITM` preprocessor directive to break out of loops early (as well as to break out of macros). See Section 3.4 [Defining your own directives], page 8.

3.3 Preprocessor variables

You can use variables in GASP to represent strings, registers, or the results of expressions.

You must distinguish two kinds of variables:

1. Variables defined with `.EQU` or `.ASSIGN`. To evaluate this kind of variable in your assembly output, simply mention its name. For example, these two lines define and use a variable '`eg`':

```
eg      .EQU    FLIP-64
        ...
        mov.l   eg,r0
```

   Do not use this kind of variable in conditional expressions or while loops; GASP only evaluates these variables when writing assembly output.

2. Variables for use during preprocessing. You can define these with `.ASSIGNC` or `.ASSIGNA`. To evaluate this kind of variable, write '`\&`' before the variable name; for example,

```
opcit  .ASSIGNA  47
       ...
       .AWHILE  \&opcit GT 0
       ...
       .AENDW
```

GASP treats macro arguments almost the same way, but to evaluate them you use the prefix '`\`' rather than '`\&`'. See Section 3.4 [Defining your own directives], page 8.

pvar `.EQU` *expr*
: Assign preprocessor variable *pvar* the value of the expression *expr*. There are no restrictions on redefinition; use '`.EQU`' with the same *pvar* as often as you find it convenient.

pvar `.ASSIGN` *expr*
: Almost the same as `.EQU`, save that you may not redefine *pvar* using `.ASSIGN` once it has a value.

pvar `.ASSIGNA` *aexpr*
: Define a variable with a numeric value, for use during preprocessing. *aexpr* must be an absolute expression. You can redefine variables with `.ASSIGNA` at any time.

pvar `.ASSIGNC "`*str*`"`
: Define a variable with a string value, for use during preprocessing. You can redefine variables with `.ASSIGNC` at any time.

pvar `.REG (`*register*`)`
: Use `.REG` to define a variable that represents a register. In particular, *register* is *not evaluated* as an expression. You may use `.REG` at will to redefine register variables.

All these directives accept the variable name in the "label" position, that is at the left margin. You may specify a colon after the variable name if you wish; the first example above could have started '`eg:`' with the same effect.

3.4 Defining your own directives

The commands `.MACRO` and `.ENDM` allow you to define macros that generate assembly output. You can use these macros with a syntax similar to built-in GASP or assembler directives. For example, this definition specifies a macro `SUM` that adds together a range of consecutive registers:

```
        .MACRO  SUM FROM=0, TO=9
        ! \FROM \TO
        mov     r\FROM,r10
COUNT   .ASSIGNA        \FROM+1
        .AWHILE \&COUNT LE \TO
        add     r\&COUNT,r10
COUNT   .ASSIGNA        \&COUNT+1
        .AENDW
        .ENDM
```

With that definition, '`SUM 0,5`' generates this assembly output:

```
        ! 0 5
        mov     r0,r10
        add     r1,r10
        add     r2,r10
        add     r3,r10
        add     r4,r10
        add     r5,r10
```

`.MACRO` *macname*
`.MACRO` *macname macargs ...*

Begin the definition of a macro called *macname*. If your macro definition requires arguments, specify their names after the macro name, separated by commas or spaces. You can supply a default value for any macro argument by following the name with '=*deflt*'. For example, these are all valid `.MACRO` statements:

`.MACRO COMM`

Begin the definition of a macro called `COMM`, which takes no arguments.

`.MACRO PLUS1 P, P1`
`.MACRO PLUS1 P P1`

Either statement begins the definition of a macro called `PLUS1`, which takes two arguments; within the macro definition, write '`\P`' or '`\P1`' to evaluate the arguments.

`.MACRO RESERVE_STR P1=0 P2`

Begin the definition of a macro called `RESERVE_STR`, with two arguments. The first argument has a default value, but not the second. After the definition is complete, you can call the macro either as

'RESERVE_STR *a*,*b*' (with '\P1' evaluating to *a* and '\P2' evaluating to *b*), or as 'RESERVE_STR ,*b*' (with '\P1' evaluating as the default, in this case '0', and '\P2' evaluating to *b*).

When you call a macro, you can specify the argument values either by position, or by keyword. For example, 'SUM 9,17' is equivalent to 'SUM TO=17, FROM=9'. Macro arguments are preprocessor variables similar to the variables you define with '.ASSIGNA' or '.ASSIGNC'; in particular, you can use them in conditionals or for loop control. (The only difference is the prefix you write to evaluate the variable: for a macro argument, write '*argname*', but for a preprocessor variable, write '\&*varname*'.)

name .MACRO
name .MACRO (*macargs* ...)

An alternative form of introducing a macro definition: specify the macro name in the label position, and the arguments (if any) between parentheses after the name. Defaulting rules and usage work the same way as for the other macro definition syntax.

.ENDM
Mark the end of a macro definition.

.EXITM
Exit early from the current macro definition, .AREPEAT loop, or .AWHILE loop.

\@
GASP maintains a counter of how many macros it has executed in this pseudo-variable; you can copy that number to your output with '\@', but *only within a macro definition*.

LOCAL *name* [, ...]

Warning: LOCAL is only available if you select "alternate macro syntax" with '-a' or '--alternate'. See Section 3.9 [Alternate macro syntax], page 15.

Generate a string replacement for each of the *name* arguments, and replace any instances of *name* in each macro expansion. The replacement string is unique in the assembly, and different for each separate macro expansion. LOCAL allows you to write macros that define symbols, without fear of conflict between separate macro expansions.

3.5 Data output

In assembly code, you often need to specify working areas of memory; depending on the application, you may want to initialize such memory or not. GASP provides preprocessor directives to help you avoid repetitive coding for both purposes.

You can use labels as usual to mark the data areas.

3.5.1 Initialized data

These are the GASP directives for initialized data, and the standard GNU assembler directives they expand to:

`.DATA` *expr*`,` *expr*`, ...`
`.DATA.B` *expr*`,` *expr*`, ...`
`.DATA.W` *expr*`,` *expr*`, ...`
`.DATA.L` *expr*`,` *expr*`, ...`

Evaluate arithmetic expressions *expr*, and emit the corresponding `as` directive (labelled with *lab*). The unqualified `.DATA` emits '`.long`'; `.DATA.B` emits '`.byte`'; `.DATA.W` emits '`.short`'; and `.DATA.L` emits '`.long`'.

For example, '`foo .DATA 1,2,3`' emits '`foo: .long 1,2,3`'.

`.DATAB` *repeat*`,` *expr*
`.DATAB.B` *repeat*`,` *expr*
`.DATAB.W` *repeat*`,` *expr*
`.DATAB.L` *repeat*`,` *expr*

Make `as` emit *repeat* copies of the value of the expression *expr* (using the `as` directive `.fill`). '`.DATAB.B`' repeats one-byte values; '`.DATAB.W`' repeats two-byte values; and '`.DATAB.L`' repeats four-byte values. '`.DATAB`' without a suffix repeats four-byte values, just like '`.DATAB.L`'.

repeat must be an absolute expression with a positive value.

`.SDATA "`*str*`" ...`

String data. Emits a concatenation of bytes, precisely as you specify them (in particular, *nothing is added to mark the end* of the string). See Section 3.8.2 [String and numeric constants], page 14, for details about how to write strings. `.SDATA` concatenates multiple arguments, making it easy to switch between string representations. You can use commas to separate the individual arguments for clarity, if you choose.

`.SDATAB` *repeat*`, "`*str*`" ...`

Repeated string data. The first argument specifies how many copies of the string to emit; the remaining arguments specify the string, in the same way as the arguments to `.SDATA`.

`.SDATAZ "`*str*`" ...`

Zero-terminated string data. Just like `.SDATA`, except that `.SDATAZ` writes a zero byte at the end of the string.

`.SDATAC "`*str*`" ...`

Count-prefixed string data. Just like `.SDATA`, except that GASP precedes the string with a leading one-byte count. For example, '`.SDATAC "HI"`' generates '`.byte 2,72,73`'. Since the count field is only one byte, you can only use `.SDATAC` for strings less than 256 bytes in length.

3.5.2 Uninitialized data

Use the `.RES`, `.SRES`, `.SRESC`, and `.SRESZ` directives to reserve memory and leave it uninitialized. GASP resolves these directives to appropriate calls of the GNU `as` `.space` directive.

`.RES` *count*
`.RES.B` *count*
`.RES.W` *count*
`.RES.L` *count*

Reserve room for *count* uninitialized elements of data. The suffix specifies the size of each element: `.RES.B` reserves *count* bytes, `.RES.W` reserves *count* pairs of bytes, and `.RES.L` reserves *count* quartets. `.RES` without a suffix is equivalent to `.RES.L`.

`.SRES` *count*
`.SRES.B` *count*
`.SRES.W` *count*
`.SRES.L` *count*

`.SRES` is a synonym for '`.RES`'.

`.SRESC` *count*
`.SRESC.B` *count*
`.SRESC.W` *count*
`.SRESC.L` *count*

Like `.SRES`, but reserves space for *count*`+1` elements.

`.SRESZ` *count*
`.SRESZ.B` *count*
`.SRESZ.W` *count*
`.SRESZ.L` *count*

Like `.SRES`, but reserves space for *count*`+1` elements.

3.6 Assembly listing control

The GASP listing-control directives correspond to related GNU `as` directives.

`.PRINT LIST`
`.PRINT NOLIST`

Print control. This directive emits the GNU `as` directive `.list` or `.nolist`, according to its argument. See section "`.list`" in *Using as*, for details on how these directives interact.

`.FORM LIN=`*ln*
`.FORM COL=`*cols*
`.FORM LIN=`*ln* `COL=`*cols*

Specify the page size for assembly listings: *ln* represents the number of lines, and *cols* the number of columns. You may specify either page dimension independently, or both together. If you do not specify the number of lines, GASP assumes 60 lines; if you do not specify the number of columns, GASP assumes 132 columns. (Any values you may have specified in previous instances of `.FORM` do *not* carry over as defaults.) Emits the `.psize` assembler directive.

`.HEADING` *string*

Specify *string* as the title of your assembly listings. Emits '`.title "`*string*`"`'.

`.PAGE` Force a new page in assembly listings. Emits '`.eject`'.

3.7 Miscellaneous commands

`.ALTERNATE`
: Use the alternate macro syntax henceforth in the assembly. See Section 3.9 [Alternate macro syntax], page 15.

`.ORG` This command is recognized, but not yet implemented. GASP generates an error message for programs that use `.ORG`.

`.RADIX` *s* GASP understands numbers in any of base two, eight, ten, or sixteen. You can encode the base explicitly in any numeric constant (see Section 3.8.2 [String and numeric constants], page 14). If you write numbers without an explicit indication of the base, the most recent '`.RADIX` *s*' command determines how they are interpreted. *s* is a single letter, one of the following:

`.RADIX B` Base 2.

`.RADIX Q` Base 8.

`.RADIX D` Base 10. This is the original default radix.

`.RADIX H` Base 16.

You may specify the argument *s* in lower case (any of '`bqdh`') with the same effects.

`.EXPORT` *name*
`.GLOBAL` *name*
: Declare *name* global (emits '`.global` *name*'). The two directives are synonymous.

`.PROGRAM` No effect: GASP accepts this directive, and silently ignores it.

`.END` Mark end of each preprocessor file. GASP issues a warning if it reaches end of file without seeing this command.

`.INCLUDE "`*str*`"`
: Preprocess the file named by *str*, as if its contents appeared where the `.INCLUDE` directive does. GASP imposes a maximum limit of 30 stacked include files, as a sanity check.

`.ALIGN` *size*
: Evaluate the absolute expression *size*, and emit the assembly instruction '`.align` *size*' using the result.

3.8 Details of the GASP syntax

Since GASP is meant to work with assembly code, its statement syntax has no surprises for the assembly programmer.

Whitespace (blanks or tabs; *not* newline) is partially significant, in that it delimits up to three fields in a line. The amount of whitespace does not matter; you may line up fields in separate lines if you wish, but GASP does not require that.

The *first field*, an optional *label*, must be flush left in a line (with no leading whitespace) if it appears at all. You may use a colon after the label if you wish; GASP neither requires the colon nor objects to it (but will not include it as part of the label name).

The *second field*, which must appear after some whitespace, contains a GASP or assembly *directive*.

Any *further fields* on a line are *arguments* to the directive; you can separate them from one another using either commas or whitespace.

3.8.1 Special syntactic markers

GASP recognizes a few special markers: to delimit comments, to continue a statement on the next line, to separate symbols from other characters, and to copy text to the output literally. (One other special marker, '`\@`', works only within macro definitions; see Section 3.4 [Defining your own directives], page 8.)

The trailing part of any GASP source line may be a *comment*. A comment begins with the first unquoted comment character ('`!`' by default), or an escaped or doubled comment character ('`\!`' or '`!!`' by default), and extends to the end of a line. You can specify what comment character to use with the '`-c`' option (see Chapter 2 [Command Line Options], page 3). The two kinds of comment markers lead to slightly different treatment:

`!` A single, un-escaped comment character generates an assembly comment in the GASP output. GASP evaluates any preprocessor variables (macro arguments, or variables defined with `.ASSIGNA` or `.ASSIGNC`) present. For example, a macro that begins like this

```
        .MACRO  SUM FROM=0, TO=9
        ! \FROM \TO
```

issues as the first line of output a comment that records the values you used to call the macro.

`\!`
`!!` Either an escaped comment character, or a double comment character, marks a GASP source comment. GASP does not copy such comments to the assembly output.

To *continue a statement* on the next line of the file, begin the second line with the character '`+`'.

Occasionally you may want to prevent GASP from preprocessing some particular bit of text. To *copy literally* from the GASP source to its output, place '`\(`' before the string to copy, and '`)`' at the end. For example, write '`\(\!)`' if you need the characters '`\!`' in your assembly output.

To *separate a preprocessor variable* from text to appear immediately after its value, write a single quote (`'`). For example, '`.SDATA "\P'1"`' writes a string built by concatenating the value of `P` and the digit '`1`'. (You cannot achieve this by writing just '`\P1`', since '`P1`' is itself a valid name for a preprocessor variable.)

3.8.2 String and numeric constants

There are two ways of writing *string constants* in GASP: as literal text, and by numeric byte value. Specify a string literal between double quotes (`"`*str*`"`). Specify an individual numeric byte value as an absolute expression between angle brackets (`<`*expr*`>`. Directives that output strings allow you to specify any number of either kind of value, in whatever order is convenient, and concatenate the result. (Alternate syntax mode introduces a number of alternative string notations; see Section 3.9 [Alternate macro syntax], page 15.)

You can write *numeric constants* either in a specific base, or in whatever base is currently selected (either 10, or selected by the most recent `.RADIX`).

To write a number in a *specific base*, use the pattern *s*`'`*ddd*: a base specifier character *s*, followed by a single quote followed by digits *ddd*. The base specifier character matches those you can specify with `.RADIX`: '`B`' for base 2, '`Q`' for base 8, '`D`' for base 10, and '`H`' for base 16. (You can write this character in lower case if you prefer.)

3.8.3 Symbols

GASP recognizes symbol names that start with any alphabetic character, '`_`', or '`$`', and continue with any of the same characters or with digits. Label names follow the same rules.

3.8.4 Arithmetic expressions in GASP

There are two kinds of expressions, depending on their result: *absolute* expressions, which resolve to a constant (that is, they do not involve any values unknown to GASP), and *relocatable* expressions, which must reduce to the form

addsym`+`*const*`-`*subsym*

where *addsym* and *subsym* are assembly symbols of unknown value, and *const* is a constant.

Arithmetic for GASP expressions follows very similar rules to C. You can use parentheses to change precedence; otherwise, arithmetic primitives have decreasing precedence in the order of the following list.

1. Single-argument `+` (identity), `-` (arithmetic opposite), or `~` (bitwise negation). *The argument must be an absolute expression.*
2. `*` (multiplication) and `/` (division). *Both arguments must be absolute expressions.*
3. `+` (addition) and `-` (subtraction). *At least one argument must be absolute.*
4. `&` (bitwise and). *Both arguments must be absolute.*
5. `|` (bitwise or) and `~` (bitwise exclusive or; `^` in C). *Both arguments must be absolute.*

3.8.5 String primitives

You can use these primitives to manipulate strings (in the argument field of GASP statements):

`.LEN("`*str*`")`
: Calculate the length of string `"`*str*`"`, as an absolute expression. For example, '`.RES.B .LEN("sample")`' reserves six bytes of memory.

`.INSTR("`*string*`", "`*seg*`", `*ix*`)`
: Search for the first occurrence of *seg* after position *ix* of *string*. For example, '`.INSTR("ABCDEFG", "CDE", 0)`' evaluates to the absolute result `2`.

  The result is `-1` if *seg* does not occur in *string* after position *ix*.

`.SUBSTR("`*string*`",`*start*`,`*len*`)`
: The substring of *string* beginning at byte number *start* and extending for *len* bytes.

3.9 Alternate macro syntax

If you specify '`-a`' or '`--alternate`' on the GASP command line, the preprocessor uses somewhat different syntax. This syntax is reminiscent of the syntax of Phar Lap macro assembler, but it is *not* meant to be a full emulation of Phar Lap or similar assemblers. In particular, GASP does not support directives such as `DB` and `IRP`, even in alternate syntax mode.

In particular, '`-a`' (or '`--alternate`') elicits these differences:

Preprocessor directives
: You can use GASP preprocessor directives without a leading '`.`' dot. For example, you can write '`SDATA`' with the same effect as '`.SDATA`'.

LOCAL
: One additional directive, `LOCAL`, is available. See Section 3.4 [Defining your own directives], page 8, for an explanation of how to use `LOCAL`.

String delimiters
: You can write strings delimited in these other ways besides `"`*string*`"`:

  `'`*string*`'`
  : You can delimit strings with single-quote charaters.

  <*string*>
  : You can delimit strings with matching angle brackets.

single-character string escape
: To include any single character literally in a string (even if the character would otherwise have some special meaning), you can prefix the character with '`!`' (an exclamation mark). For example, you can write '`<4.3 !> 5.4!!>`' to get the literal text '`4.3 > 5.4!`'.

Expression results as strings
: You can write '`%`*expr*' to evaluate the expression *expr* and use the result as a string.

4 GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled "Endorsements". Such a section may not be included in the

Modified Version. N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
```

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

A

B

C

D

E

F

G

I

L

M

N

P

R

S

T

W

Table of Contents