

# M0, version 0.4

---

A flexible macro processor  
Edition 0.4, 3 September 2026

by Marco de Beurs  
([m0macroprocessor@gmail.com](mailto:m0macroprocessor@gmail.com))

---

This manual (3 September 2026) is for M0 (version 0.4), a package containing an implementation of the `m0` macro processor.

Copyright © 2025, 2026 Marco de Beurs

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License.”

# Table of Contents

|          |                                            |           |
|----------|--------------------------------------------|-----------|
| <b>1</b> | <b>Introduction and preliminaries</b>      | <b>3</b>  |
| 1.1      | For the reader                             | 3         |
| 1.2      | Why another macro processor?               | 3         |
| 1.3      | The m0 name?                               | 3         |
| 1.4      | Performance?                               | 4         |
| <b>2</b> | <b>Invoking m0</b>                         | <b>5</b>  |
| 2.1      | Introduction                               | 5         |
| 2.2      | Command line options                       | 5         |
| 2.3      | Specifying input files on the command line | 6         |
| 2.4      | Specifying options for emulation           | 6         |
| <b>3</b> | <b>Macro processing</b>                    | <b>7</b>  |
| 3.1      | Introduction                               | 7         |
| 3.2      | How m0 copies input to output              | 7         |
| 3.3      | Macro names                                | 7         |
| 3.3.1    | Normal macros                              | 8         |
| 3.3.2    | Variable length macros                     | 8         |
| 3.4      | Macros with arguments                      | 9         |
| 3.5      | Symbols for argument substitution          | 9         |
| 3.6      | Recursion                                  | 10        |
| 3.7      | The complete working of macros in m0       | 11        |
| 3.7.1    | pre-size and post-size                     | 13        |
| 3.7.2    | Pattern for collecting arguments           | 13        |
| 3.7.3    | The stack                                  | 13        |
| 3.7.4    | Builtin functions                          | 14        |
| 3.7.5    | Default argument substitution              | 14        |
| 3.7.6    | Pattern argument substitution              | 15        |
| 3.8      | Sets of macros                             | 15        |
| 3.9      | At startup                                 | 15        |
| 3.10     | Priority                                   | 15        |
| <b>4</b> | <b>Patterns</b>                            | <b>17</b> |
| 4.1      | Introduction                               | 17        |
| 4.2      | Activation of patterns                     | 17        |
| 4.3      | Pattern parts                              | 17        |
| 4.3.1    | Characters and character sets              | 17        |
| 4.3.2    | Matching of a pattern part                 | 18        |
| 4.3.3    | Programs                                   | 18        |
| 4.4      | Pattern parts working together             | 18        |
| 4.4.1    | Example                                    | 19        |
| 4.4.2    | Priorities                                 | 20        |

|          |                                                         |           |
|----------|---------------------------------------------------------|-----------|
| <b>5</b> | <b>Macros and patterns consisting of multiple parts</b> | <b>21</b> |
| 5.1      | Introduction                                            | 21        |
| 5.2      | Technical background                                    | 21        |
| 5.3      | Defining a part                                         | 21        |
| 5.4      | Linking parts                                           | 22        |
| 5.5      | Macro with multiple parts expansion                     | 23        |
| 5.6      | Patterns with multiple parts                            | 23        |
| <b>6</b> | <b>Builtin functions</b>                                | <b>25</b> |
| 6.1      | Introduction                                            | 25        |
| 6.2      | Functions for defining macros                           | 25        |
| 6.3      | Functions for defining vlm parts                        | 30        |
| 6.4      | Macro sets                                              | 31        |
| 6.5      | Functions for defining patterns                         | 35        |
| 6.6      | Functions for setting symbols                           | 36        |
| 6.7      | String functions                                        | 38        |
| 6.8      | Functions for using variables and system information    | 39        |
| 6.9      | Functions for getting macro information                 | 40        |
| 6.10     | Functions for input and output                          | 41        |
| 6.11     | Functions for handling of errors                        | 42        |
| 6.12     | Some functions are missing?                             | 42        |
| <b>7</b> | <b>Programs for patterns and macros</b>                 | <b>45</b> |
| 7.1      | Introduction to programs                                | 45        |
| 7.2      | Stacks                                                  | 45        |
| 7.3      | Programs                                                | 46        |
| 7.3.1    | Data types                                              | 46        |
| 7.3.2    | Instructions                                            | 46        |
| 7.3.3    | Call to a program                                       | 47        |
| 7.4      | Selecting a stack                                       | 47        |
| 7.5      | Manipulation on the stack                               | 47        |
| 7.6      | Copy between stacks                                     | 48        |
| 7.7      | Collecting argument strings                             | 49        |
| 7.8      | Program flow                                            | 50        |
| 7.9      | Getting variables                                       | 51        |
| 7.10     | Comparing values                                        | 52        |
| 7.11     | Mathematical instructions                               | 53        |
| 7.12     | Logical bit instructions                                | 54        |
| 7.13     | Logical instructions                                    | 55        |
| 7.14     | String instructions                                     | 55        |
| 7.15     | Output manipulation instructions                        | 57        |
| 7.16     | Instructions for overruling macro settings              | 58        |
| 7.17     | Operator stack instructions                             | 59        |
| 7.18     | Getting macro information                               | 60        |
| 7.19     | Counter variables                                       | 62        |
| 7.20     | Calling a builtin function                              | 62        |

|          |                                      |           |
|----------|--------------------------------------|-----------|
| 7.21     | Manipulation of stacks .....         | 63        |
| <b>8</b> | <b>Emulation examples .....</b>      | <b>65</b> |
| 8.1      | Introduction .....                   | 65        |
| 8.2      | M4 emulation example .....           | 65        |
| 8.2.1    | Introduction .....                   | 65        |
| 8.2.2    | Emulation .....                      | 65        |
| 8.2.3    | Main design .....                    | 65        |
| 8.2.4    | At the start .....                   | 66        |
| 8.2.5    | Large argument numbers .....         | 67        |
| 8.2.6    | Argument with single number .....    | 67        |
| 8.2.7    | Quoting in default macro set .....   | 67        |
| 8.2.8    | Special characters .....             | 68        |
| 8.2.9    | Command line options .....           | 68        |
| 8.2.10   | Collecting arguments of macros ..... | 69        |
| 8.2.11   | define .....                         | 69        |
| 8.2.12   | pushdef .....                        | 70        |
| 8.2.13   | defn .....                           | 71        |
| 8.2.14   | popdef .....                         | 71        |
| 8.2.15   | undefine .....                       | 72        |
| 8.2.16   | Quoting in <code>m4</code> .....     | 72        |
| 8.2.17   | changequote .....                    | 72        |
| 8.2.18   | Comments .....                       | 73        |
| 8.2.19   | changecom .....                      | 73        |
| 8.2.20   | builtin .....                        | 73        |
| 8.2.21   | include .....                        | 74        |
| 8.2.22   | sinclude .....                       | 74        |
| 8.2.23   | place .....                          | 74        |
| 8.2.24   | splace .....                         | 74        |
| 8.2.25   | divert .....                         | 74        |
| 8.2.26   | undivert .....                       | 75        |
| 8.2.27   | divnum .....                         | 75        |
| 8.2.28   | <code>__line__</code> .....          | 75        |
| 8.2.29   | <code>__file__</code> .....          | 75        |
| 8.2.30   | <code>__program__</code> .....       | 75        |
| 8.2.31   | <code>__unix__</code> .....          | 75        |
| 8.2.32   | <code>unix</code> .....              | 76        |
| 8.2.33   | errprint .....                       | 76        |
| 8.2.34   | m4exit .....                         | 76        |
| 8.2.35   | index .....                          | 76        |
| 8.2.36   | substr .....                         | 76        |
| 8.2.37   | translit .....                       | 76        |
| 8.2.38   | m4wrap .....                         | 77        |
| 8.2.39   | esyscmd .....                        | 77        |
| 8.2.40   | syscmd .....                         | 77        |
| 8.2.41   | sysval .....                         | 77        |
| 8.2.42   | maketemp .....                       | 77        |
| 8.2.43   | mkstemp .....                        | 77        |

|        |                                          |    |
|--------|------------------------------------------|----|
| 8.2.44 | dnl                                      | 78 |
| 8.2.45 | incr                                     | 78 |
| 8.2.46 | decr                                     | 78 |
| 8.2.47 | shift                                    | 78 |
| 8.2.48 | indir                                    | 79 |
| 8.2.49 | ifdef                                    | 79 |
| 8.2.50 | len                                      | 79 |
| 8.2.51 | ifelse                                   | 79 |
| 8.2.52 | eval                                     | 80 |
| 8.2.53 | expr                                     | 81 |
| 8.2.54 | format                                   | 81 |
| 8.2.55 | dumpdef                                  | 82 |
| 8.2.56 | regexp                                   | 82 |
| 8.2.57 | patsubst                                 | 82 |
| 8.2.58 | tracoon                                  | 82 |
| 8.2.59 | traceoff                                 | 82 |
| 8.2.60 | debugmode                                | 82 |
| 8.2.61 | debugfile                                | 83 |
| 8.2.62 | end of emulation file, start of m4 files | 83 |
| 8.3    | M6 emulation example                     | 83 |
| 8.3.1  | Introduction                             | 83 |
| 8.3.2  | Emulation                                | 83 |
| 8.3.3  | Main design                              | 83 |
| 8.3.4  | At the start                             | 84 |
| 8.3.5  | Quoting in default macro set             | 84 |
| 8.3.6  | Quoting                                  | 84 |
| 8.3.7  | Collecting arguments of macros           | 85 |
| 8.3.8  | Main macro for executing all macros      | 86 |
| 8.3.9  | GO, GOBK                                 | 86 |
| 8.3.10 | DEF                                      | 86 |
| 8.3.11 | COPY                                     | 86 |
| 8.3.12 | SEQ                                      | 86 |
| 8.3.13 | SNE                                      | 86 |
| 8.3.14 | GT                                       | 87 |
| 8.3.15 | GE                                       | 87 |
| 8.3.16 | LT                                       | 87 |
| 8.3.17 | LE                                       | 87 |
| 8.3.18 | EQ                                       | 87 |
| 8.3.19 | NE                                       | 87 |
| 8.3.20 | IF                                       | 87 |
| 8.3.21 | SIZE                                     | 88 |
| 8.3.22 | SUBSTR                                   | 88 |
| 8.3.23 | ADD                                      | 88 |
| 8.3.24 | SUB                                      | 88 |
| 8.3.25 | MPY                                      | 88 |
| 8.3.26 | DIV                                      | 88 |
| 8.3.27 | EXP                                      | 88 |
| 8.3.28 | DNL                                      | 88 |

|        |                                                 |    |
|--------|-------------------------------------------------|----|
| 8.3.29 | Source, End, Trace .....                        | 89 |
| 8.3.30 | end of emulation file, start of m6 files .....  | 89 |
| 8.4    | GPM emulation example .....                     | 89 |
| 8.4.1  | Introduction .....                              | 89 |
| 8.4.2  | Emulation .....                                 | 89 |
| 8.4.3  | Main design .....                               | 89 |
| 8.4.4  | At the start .....                              | 89 |
| 8.4.5  | Quoting in default macro set .....              | 90 |
| 8.4.6  | Quoting .....                                   | 90 |
| 8.4.7  | Collecting arguments of macros .....            | 90 |
| 8.4.8  | Substitution of arguments .....                 | 91 |
| 8.4.9  | Main macro for executing all macros .....       | 91 |
| 8.4.10 | DEF .....                                       | 92 |
| 8.4.11 | UPDATE .....                                    | 92 |
| 8.4.12 | BIN .....                                       | 92 |
| 8.4.13 | DEC .....                                       | 92 |
| 8.4.14 | VAL .....                                       | 92 |
| 8.4.15 | BAR .....                                       | 93 |
| 8.4.16 | end of emulation file, start of gpm files ..... | 93 |

## **Appendix A Index for m0 builtins and stack instructions..... 95**

|     |                         |    |
|-----|-------------------------|----|
| A.1 | Builtin functions ..... | 95 |
| A.2 | Instructions .....      | 95 |

## **Appendix B How to make copies of the overall M0 package..... 99**

|     |                                                        |     |
|-----|--------------------------------------------------------|-----|
| B.1 | License for copying the M0 package .....               | 99  |
| B.2 | License of the SDS library used in the M0 package..... | 110 |

## **Appendix C How to make copies of this manual .. 111**

|     |                                       |     |
|-----|---------------------------------------|-----|
| C.1 | License for copying this manual ..... | 111 |
|-----|---------------------------------------|-----|



m0 is an implementation of a flexible macro processor.

This is release 0.4. Almost everything should work as described, but it is still considered experimental. This means that the functions, instructions and options and how they operate are not yet in a final state. Although effort is made to keep newer versions compatible with older macro code, this can not be guaranteed.



# 1 Introduction and preliminaries

## 1.1 For the reader

This document is not a manual for learning to work with macros. For learning macros in general the manuals of existing general macro processors such as GNU `m4` are advised.

This document explains the possibilities, use and flexibility of the `m0` macro processor. It will allow you to e.g. define your own macro language or to emulate an existing macro processor.

How the macros in `m0` work is described in Chapter 3 [Macro processing], page 7. Patterns are an important feature to provide flexibility of macros and are explained in Chapter 4 [Patterns], page 17.

The functions available for macros are listed in Chapter 6 [Builtin functions], page 25, the instructions available for patterns are listed in Chapter 7 [Programs for patterns and macros], page 45.

Emulation examples found in Chapter 8 [Emulation examples], page 65, demonstrate the use of `m0` to emulate existing macro processors. The `m4` emulation is demonstrated in Section 8.2 [M4 emulation example], page 65, the `m6` emulation in Section 8.3 [M6 emulation example], page 83, and the `gpm` emulation in Section 8.4 [GPM emulation example], page 89.

## 1.2 Why another macro processor?

Several good general macro processors such as `m4`, `ML/I` or `gpp` exist. These macro processors all have their own syntax and have no, some or more flexibility in the macro naming and use. They all have macros defined by words or "atoms" and are hereby linked to the known syntax of programming languages.

The purpose of `m0` is to have almost no restrictions on syntax and with that maximum flexibility. With this flexibility it should even be possible to emulate many of the existing macro processors.

This flexibility goes so far, that at startup only 1 macro is available to name new macros. You have to name the builtin functions and define the syntax first before a complete macro processor is available!

## 1.3 The `m0` name?

The `m4` macro processor has been inspirational for the development of `m0`. The much used `m4` macro processor has its predecessors named `m3` and `m6`. An extended macro processor is called `m5`.

For a mechanical engineer these names look like metric screw thread. Besides this, `m0` has as a goal to be more flexible and have more functionality than `m4`. It is however also more basic, in the sense that not much is predefined. This together with the trend of zero sugar drinks seems to make the `m0` name appropriate.

## 1.4 Performance?

Intuitively a very flexible general macro processor should be a lot slower in execution than a normal general macro processor. Flexibility normally means more and slower code.

The algorithm used in `m0` for detecting macros is a bitap algorithm. This algorithm is fast and flexible at the same time. A preliminary rough estimate (on a relatively modern 64 bit computer) is that `m0` with `m4` emulation is twice as fast as `m4` processing a big file without macros. When processing a big file with macros, `m0` has a similar or a little bit better performance than `m4`.

## 2 Invoking m0

### 2.1 Introduction

The format for calling m0 is following the "default" GNU and posix rules:

```
m0 [option...] [file...]
```

The ordering of options is not relevant unless the `-0 / --ordered-options` is set. The files are processed in order of placement on the command line.

### 2.2 Command line options

`-?`

`--help` Print a help summary on standard output, then immediately exit m0 without reading any input files or performing any other actions.

`--usage` Print a short usage on standard output, then immediately exit m0 without reading any input files or performing any other actions.

`-D name[=value]`

`--define=name[=value]`

This enters *name* into the macro name table. If `'=value'` is missing, the value is taken to be the empty string. The *value* can be any string, but the macro can not be defined to take arguments. The order with respect to file names is not significant in contrast to some other macro processors.

`-E`

`--fatal-warnings`

Controls the effect of warnings.

By default no warnings are printed and execution continues when errors are detected.

If specified:

- Once; warnings are printed, exit codes are not set and execution continues.
- Twice; warnings are printed, exit codes are set, but execution continues.
- Three times; warnings are printed, exit codes are set, execution stops.

`-o file`

`--output=file`

Specifies the file to use for output. The default output, when no output file is specified, is standard output.

`-0`

`--ordered-options`

After setting this option, the options on the command line are valid for succeeding files. This is valid for the options: `-D`, `-E`, `-o`, `-T` and `-t`.

The options set for the succeeding file are based on the defaults and options set before the `-0` option and the options set directly before the file.

The `-o` and `-t` options to specify an output or a trace file are persistent. This means that these files are only changed when specified and are not automatically

reset to the default. Specifying these files opens a new file. They will not append to an existing or previously specified file.

The `-D` option, which defines a macro, is also persistent and therefore this definition also exists in later files.

`-s`

`--statistics`

Print statistics of the use of macros, macro sets, patterns and internal memories at the end of running the program. This is output to standard output.

Can be used as a source for debugging of macros.

`-T`

`--traceon`

Set tracing of macros on. Information of the called macros is output to standard output or to a file.

`-t file`

`--tracefile=file`

Specifies the file to output trace information to.

`-V`

`--version`

Print the version number of the program on standard output, then immediately exit `m0` without reading any input files or performing any other actions.

## 2.3 Specifying input files on the command line

Multiple input files can be specified on the command line. The files are read in the order in which they are placed on the command line. If no file is specified the standard input is taken as input.

A `-` specifies standard input as input. This `-` can appear multiple times and will also open the standard input multiple times. The control-D normally ends the input from standard input in a console.

The position of options compared to the position of the files makes no difference, unless the `-O / --ordered-options` is set. Files placed after `--` are also used as input files and are not transferred to the argument string for use in the macro processor.

## 2.4 Specifying options for emulation

Options after a `--` are not processed as options to `m0` (files are however necessarily processed by the `m0` program).

These options are put in a string that can be output by a macro function. This string can be used to set options in an emulation package of another macro processor. Important options might be the definition of macros on the command line or syntax options.

It is however the emulation package that is responsible to process these options. In the `m0` program nothing is set depending on these options. It would also be difficult to let the `m0` program process these options, because these options should be known already before the emulation code runs.

## 3 Macro processing

### 3.1 Introduction

In this part the basics of macro expansion and processing in `m0` is explained.

This information is useful to understand the characteristics and flexibility of the processing of macros in `m0`. It is therefore advised to read this chapter before you start to define your own macros.

### 3.2 How `m0` copies input to output

In `m0` there is no concept of words, atoms or tokens to find macros. `M0` works with Bitap algorithms and these algorithms work on a character or byte as a basis.

The input bytes are copied to the output until a macro is recognised. When a macro is recognised the macro is replaced by a definition of the macro.

This is for a simple macro displayed as:

```

in:  | | | |m|a|c|r|o| |n|a|m|e| | | | | | | | | | |
      +-----+
      macro name
      |
      |
      v

out:  | | | |d|e|f|i|i|n|i|i|i|o|i|n| | | | | | | | | | |
      +-----+
      definition

```

The Bitap algorithm works sequentially on a stream of bytes. Therefore a macro is recognised when the last byte of the macro name is processed. The macro name is then removed from the output. After this, the definition is placed in the output. The copying of input to output bytes continues after the definition.

The definition can have any length (also 0) and can contain any character or byte.

### 3.3 Macro names

Internally `m0` has two types of macros:

- Normal macros.
- Variable length macros (vlm).

Although the user does not have to handle these different types differently when defining a macro, they have different possibilities that should be known. Further the use of a vlm reduces the performance because the vlm uses another additional algorithm next to the normal macros.

### 3.3.1 Normal macros

In most macro processors whitespace is used to recognise macros. The characters that can be used in macro names are limited to a set of allowable characters. For example in **m4** the allowable set of characters to recognise macros are alphanumeric characters and underscore.

The macro names in **m0** can contain any byte value (0 – 255). There is no whitespace necessary in front or after the macro name. Because of this a macro can be recognised when included in a larger name.

The Bitap algorithm used in **m0** allows another useful possibility:

A single character position in the macro name can have a set of possible byte values.

For example:

**[Nn]ame**

means the word "name" starting with a capital or small n.

This possibility can be used in e.g. emulation. The macro names can be started and ended with a whitespace. A whitespace often includes a space, newline, tab and other bytes in the range of 0 – 0x20. With this whitespace at the start and end of a macro it is no longer recognised inside larger names.

The current implementation using the Bitap algorithm results in a limit for the length of macro names. The macro name can have a maximum of 64 positions. This should be sufficient for most uses; to quote supposedly Bill Gates: 640 Kbytes ought be be enough for anyone!

### 3.3.2 Variable length macros

A macro name can also be defined having a variable length. This means that the macro name recognised in the processed text can have parts that are variable in length. A variable length macro is thus a macro that has in addition to normal macros parts that have a variable length.

For example a macro with name:

**[Nn]\*ame**

would be recognised in all of the following examples:

**Name**

**ame**

**NNName**

**nnnnnnname**

**NnNname**

The **[Nn]\*** means zero or more characters being a capital or small n.

The variable length macro can be used for recognising patterns or regular expressions. The variable part is limited to a single character or set of characters, although it is possible to have multiple variable parts. The possibilities are equal to the pattern parts explained later in Chapter 4 [Patterns], page 17, except that a one time trigger is not supported. In Chapter 5 [Macros and patterns consisting of multiple parts], page 21, further possibilities to use multiple parts are described.

The vlm has extra options compared to a normal macro. The length match can be set. It is e.g. possible to set the macro to use the shortest match or the longest match in a text.

### 3.4 Macros with arguments

A macro whereby only text is replaced is often too limited. Therefore most macro processors allow the use of arguments to increase the usefulness of macros.

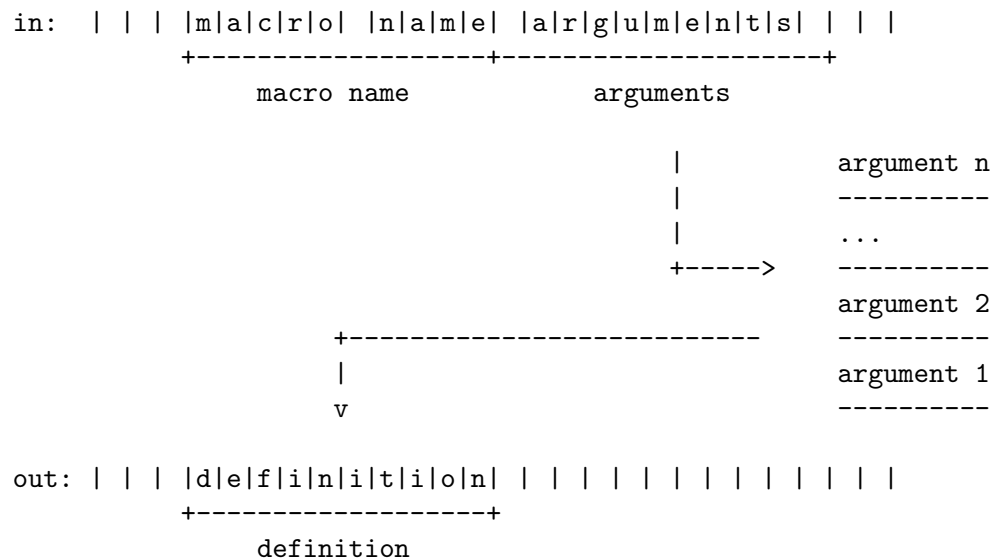
Arguments are added to the macro with a defined syntax to be able to distinguish the arguments from other text. A macro in e.g. `m4` looks like a function call as in programming languages. In `m0` the syntax for arguments is not predefined. The user can define the syntax by using patterns (see Chapter 4 [Patterns], page 17).

The arguments in `m0` are first placed on a stack. The first position on the stack (stack position 0) holds the name of the macro. The further positions (stack position 1 and up) hold the arguments, whereby the first argument is placed on the second position on the stack.

No limit exists for the number of arguments, except memory limitations of the computer.

In `m0` the stack can be manipulated by small programs (see Chapter 7 [Programs for patterns and macros], page 45) used in patterns and in a macro. This enables the flexibility of `m0`, whereby the user is able to define the syntax or even define a functionality of a macro that is not built in.

In the next section the use of arguments in the definition is explained. The following figure shows symbolically the working of a macro with arguments and stack:



### 3.5 Symbols for argument substitution

It is necessary to be able to substitute the arguments that are placed on the stack in the definition at specific positions in the definition. A default for doing this is defined in a similar way as `m4`. The position where an argument should be substituted is indicated by the symbol `$`. After this the number of the stack position is placed. For example `$1` is the first argument on the stack.

Also some special symbols like in `m4` are defined:

- `$*`        This symbol returns all arguments.
- `$@`        This symbol returns all arguments with quotes.
- `$#`        This symbol returns the number of arguments.

These symbols are defined by default at startup. They can be changed by the user by using the builtin function `chararg`. See for the use of this function Section 6.6 [Functions for setting symbols], page 36.

### 3.6 Recursion

Recursion is running the output of a macro through the macro processor so that possibly more macros are detected from the output of the macro. Thus if the definition of a macro contains another macro, this macro will also be expanded.

In `m0` this recursion can be activated per macro. This flexibility is needed depending on the purpose of a macro.

Another type of recursion is the detection of macros while collecting arguments of a macro. With this recursion it is possible to construct arguments using other macros. Also this recursion type can be activated per macro in `m0`.

These recursions can introduce problems when macros that are not wanted are used in the arguments or definition. In e.g. `m4` this is controlled by using quoting. However in `m0` there is no quoting built into the macro processor. If quoting is needed in `m0` a macro can be defined that functions like quoting. The recursion in this quote macro is set to not active. In the M4 emulation example (see Section 8.2 [M4 emulation example], page 65) this quoting macro example can be seen.

The recursion in `m0` goes even further than what is possible in other macro processors, because the macro processor uses a bitap algorithm that works on characters or bytes. It is possible to construct a macro that will be detected and expanded by using two other macros that will together expand to the constructed macro.

The following example that is also used in the description of `define` can be used to demonstrate this. First two macros are defined called `aa` and `bb`. Macro `aa` will expand to `te` and macro `bb` will expand to `st`. A third macro `test` is defined that will expand to `Hello World!`.

finally a string `aabb` that contains the first two macros is written:

```
Define:aa;te
Define:bb;st
Define:test;Hello World!

aabb
```

This example will result in:

```
Hello World!
```

What happened is:

- first `aa` is replaced with `te`.
- secondly `bb` is replaced with `st`.
- now the word `test` has appeared. This is also a macro and thus the `test` is replaced with `Hello World!`.

Recursion is one of the difficult and confusing aspects of macro processors. It is therefore important to set the possible recursions properly when defining a macro. Other macro processors use quoting to overcome issues of recursion. This is also possible in `m0`. Another way to stop the recursion as in the example above is the use of a virtual character in the macro. This is further explained in the description of the `define` function.

The settings of recursion are made when defining a macro. See the function `define` in Section 6.2 [Functions for defining macros], page 25.

### 3.7 The complete working of macros in `m0`

In the previous sections the main characteristics and some flexibility of macros in `m0` have been described. The macros in `m0` are however more complex and flexible. This section tries to explain the complete working of macros.

An important part of macros that gives a big flexibility is the use of patterns and small programs in macros. The use of patterns is even more complex than the macros and therefore is described in a separate chapter, see Chapter 4 [Patterns], page 17. The instructions for the small programs can be found in Chapter 7 [Programs for patterns and macros], page 45.

The following figure is a symbolic overview of the working of macros in `m0`:

```

in: | | | | m|a|c|r|o| | n|a|m|e|( | a|r|g|u|m|e|n|t|s|) | | |
    +-----+-----+
        macro name      : :      arguments
    +-+                +-+
pre-size              post-size  |
                        : :      v
                        : :
                        | | | | | | | | | | | | |
    +-----+-----+-----+
                        pattern for collecting arguments +
                        small programs

                                |
                                |
                                +----->      argument n
                                -----
                                ...
optional program for stack manipulation --> -----
                                                argument 3
                                                -----
                                                +-----or-----
                                                |         |
macro definition: | | $|1| | $|2| | |         |         |         argument 2
                  +-----+         |         |         ----- <--+
                  |         |         |         |         argument 1 ---|---+
                  |         |         |         |         -----
                  |         |         |         |         |
                  |         |         |         +----> builtin function --+ |
                  |         |         |         |         |
                  |         +----- or ---+         +-----+         |
                  |         |         |         |         |
                  |         |         |         |         |
    +-----or -----> or <--argument 1-----+
    |         |         |         |         |
    v         v         v         v         |
default substitution      pattern substitution +
                        small programs
    |
    |
    |         +-----+
    |         v
    +-----> or <-----+
    |
    v

out: | | | | | d|e|f|i|i|n|i|i|t|i|i|o|n| | | | | | | | | | |
    +-----+
    +-+ replacement of macro
pre-size

```

At the top of the figure a pre-size and post-size of the macro name can be seen. These are options set during the definition of a macro.

The post-size is the part of a macro name that reappears in the collection of arguments. Normally this is the character to start argument collection, so that a macro can be defined which requires arguments. It is also used to function as a word boundary like the pre-size.

The possibility to abort the collecting of arguments has as a consequence that macros will not be called during the processing of the post-size. If this was not the case, macros could be called before the abort, whereas they should be called after the abort. The post-size is however still used to detect a (part of a) macro.

After the macro name is detected, the collecting of arguments is the first thing executed. This collecting is however only executed if a pattern for the collecting is defined in the macro.

Small forth like programs are used to copy the input to the stack. This use of programs allows for flexibility in the syntax of the argument collection and to enable extra functionality not available in the builtin macros.

As an example of the flexibility an `if else` macro can be made using the pattern and the programs. See the `ifelse` macro of `m4` in Section 8.2 [M4 emulation example], page 65, as an example how this can be programmed. Another example is the `eval` macro of the `m4` emulation that uses a big pattern for all the mathematical and logical operators.

The stack is the data store for all the programs. The stack is also the data store for the arguments of a macro.

The first position on the first stack (position 0) holds the name of the macro. This is the name as in the input minus the pre-size and post-size. It can be used to get the actual name of the macro when multiple characters were used in the macro name. Further positions on the first stack are used to hold arguments. These arguments are placed on the stack by the small programs of patterns and can also be manipulated by these programs.

Also the optional program linked to a macro can be used to manipulate the stack. An example use is to set default values for a macro that are not set by the argument collection.

The life of the stack is limited. It is started when the macro is recognised and closed after the arguments in the definition are substituted. It is thus not possible to exchange information between macros through the stack.

### 3.7.4 Builtin functions

The builtin functions perform functions that are normally not possible using user defined macros. The most important functions being the defining functions for macros and patterns.

The builtin functions get their arguments from the stack to perform their function. The functions and the used arguments are described in Chapter 6 [Builtin functions], page 25.

Some functions will have output and some will also put output on the stack. Normally the output of these functions will appear in the output, but this can be suppressed by defining a definition string. The definition string can use the information on the stack to output the wanted information.

The use of a builtin function and the definition string at the same time is not excluded. The builtin function will be executed before the argument substitution.

If the builtin function is defined in the macro it will unconditionally be executed and if it is not defined in the macro it will obviously not be executed.

### 3.7.5 Default argument substitution

To have a syntax for argument substitution available a default argument substitution is defined similar to the substitution in `m4`. The default argument substitution will use the definition text to substitute default codes with the arguments. If no definition string is defined, then there will obviously be no argument substitution.

The default is used when no pattern substitution is defined. The default characters used for argument substitution are:

`$` followed by a number, example: `$1`

Is used for argument substitution in the definition. The number is the entry of the stack.

`$*` Is used to output all arguments.

`$@` Is used to output all arguments with quotes.

`$#` Is used to output the number of arguments.

`$` followed by an `a – f` and followed by a number, example: `$b2`

Is used for argument substitution of arguments from the other stacks in the definition. An `a – f` after this character in the definition selects the stack. The number after this is the entry of the stack.

These default characters can be changed using the builtin function `chararg`. This function is described in Section 6.6 [Functions for setting symbols], page 36.

### 3.7.6 Pattern argument substitution

Instead of the default argument substitution a pattern argument substitution will be used when defined in a macro. This can be used for argument substitution with a syntax which you defined yourself. A pattern to handle this should be made before this can be defined in the macro.

If no definition string is defined in the macro, the first argument on the stack is used as replacement text. This can be used to have a macro whereby the first argument is e.g. a formatting text that is used by the pattern to substitute the arguments.

## 3.8 Sets of macros

Macros can be grouped in macro sets.

Only one macro set can be active at any one time. By switching between macro sets it is possible to switch between groups of macros to be active. It is also possible to call macros in another set by using a special macro that calls these macros.

The macro sets thus e.g. allow to hide macros from unwanted being triggered and expanded or to switch between macros with a complete different syntax.

## 3.9 At startup

At startup of `m0` only the minimum to be able to start is predefined.

The macro `0_define` exists at startup to name builtins and define new macros. It uses the pattern `0` to collect arguments and is defined in macro set `0`. The pattern `0` uses the `":"` as a start character for argument collection, the `;"` as separator between the arguments and a newline (`\n`) as the stop character of the argument collection.

In the macro set `0` the characters for argument substitution and character sets are the default characters (see Section 6.6 [Functions for setting symbols], page 36).

## 3.10 Priority

It is possible to define a macro with a name whereby this name is part of the name of another macro.

The question for this situation is: will both macros be expanded or only one? If only one which one?

The answer is: it depends.

The triggering of a macro happens when the last character is matched. So if a macro is triggered a character in position in the text before another macro, this macro will expand. The expansion hereof will probably result in the text being replaced and thereby will result in a different name for the second macro which will thus not be triggered.

When two macros trigger at the same position in the text, the macro which has the longest name will be expanded and the other not. If also the length of the macro names is the same, then the first macro defined will be expanded and the other not.

Normal macros have priority over variable length macros (`vlm`). If two `vlm` macros are triggered at the same time, the one first defined has priority. The second one will not be triggered nor expanded.



## 4 Patterns

### 4.1 Introduction

In this part the use and working of patterns is explained.

A pattern is a collection of pattern parts with associated programs. The programs are used to perform certain data transactions after a certain pattern part is detected.

Patterns do not function on their own, but are linked to macros when defining a macro. Unlike macros, patterns have a global scope and are not restricted to a macro set.

Patterns are mostly used for collecting arguments of macros. However, because of their flexibility they can also be used for other purposes such as argument substitution in the definition string or perform parser functions for e.g. infix notations.

### 4.2 Activation of patterns

Patterns are not permanently active. They are active from a certain point and end at another:

- A pattern for collecting arguments is started when a macro has been detected and stops itself when the end of argument collection is detected.
- A pattern to substitute the arguments in the definition is started at the begin of the definition and ends automatically at the end of the definition string. The same is applicable when the first argument is used as input string.

As a consequence of the linking to macros, at any one time only one pattern can be active. This does of course not exclude the nested use of macros and their associated patterns.

### 4.3 Pattern parts

Pattern parts make up a pattern and are detected in the same way as macros. However they do not replace a piece of text, but start a small program to do the necessary work.

#### 4.3.1 Characters and character sets

Pattern parts are detected in the same way as macros, but have more possibilities. Whereas a macro name has only the possibility for character or byte sets for a position, pattern parts have also the possibilities to:

- Define a position with zero or one character or character set. The default character used for this is: `?`.
- Define a position with one or more characters or character set. The default character used for this is: `+`.
- Define a position with zero or more characters or character set. The default character used for this is: `*`.
- Define a position which will only be detected the first time. The default character used for this is: `~`.

A set of characters is defined like in macro names with default square brackets. An example of all letters in ASCII code for a single position:

`[a-zA-Z]`

The set notation also has to be used for the other mentioned possibilities. The special character to define one of the possibilities is placed directly behind the set. For example to have zero or one character of a set of all letters:

`[a-zA-Z]?`

As can be seen from the previous examples, the `-` is the default to indicate a range of characters in a set.

Instead of the direct indication of a character, also the ASCII number can be used to define a character. The default character `@` is used to indicate that the following hexadecimal number defines an ASCII character. For example:

`@41`

is the letter `A`.

### 4.3.2 Matching of a pattern part

A standard pattern part is triggered in the same way as a macro: all characters should be in the input to match.

An example: a pattern part defined as `[a-z]` will be triggered with an input of `Hello` at the `e`, `l`, `l` and `o`.

A pattern part can also be set to depend on the previous pattern part using an append pattern part function. The matching of the appended pattern part will only start if the previous pattern part triggered, just as if the two pattern parts are one. For example:

First pattern part

`e1`

Appended pattern part

`[a-z]`

With input

`Hello`

Will trigger the first pattern part at the end of `Hel` and trigger the appended pattern part at the following `l`.

### 4.3.3 Programs

Together with the pattern part a small forth like program is defined. This program will need to do the necessary work for the pattern part. It is executed when the pattern part is triggered by a match in the input.

All the available instructions for the programs can be found in Chapter 7 [Programs for patterns and macros], page 45.

## 4.4 Pattern parts working together

The pattern parts together form a pattern that can do something useful. The selection of the parts is therefore an important aspect of having a proper working pattern. To illustrate this a similar pattern to the default `0` pattern for argument collection is used as an example.

### 4.4.1 Example

The following example is a pattern for collecting arguments with a similar syntax as the argument collection of `m4`. It is however more simple, in that leading spaces in arguments are not excluded and that the argument characters "(" and ")" can not be nested.

The example uses a `0_pattern` macro linked to the pattern function to define a pattern that is similar to the default pattern 0, but uses the:

- "(" instead of ":" as the start character of argument collection.
- ",", instead of ";" as the separator character of the arguments.
- ")" instead of the newline as the stop character of argument collection.

This example is using the pattern 0 for argument collection. The new pattern will be named 1.

The complete pattern is defined by:

```
0_pattern:1;[@00-@27@29-@ff]~;argpos =0 ifthen abort
0_pattern:1;[,)];end
0_pattern:1;[(,];begin
0_pattern:1;);stop
```

The first line:

```
0_pattern:1;[@00-@27@29-@ff]~;argpos =0 ifthen abort
```

defines a pattern part consisting of a single character, whereby every character except the "(" (@28) will match. It will test if a macro has arguments by checking if the first character is a "(" or not. It is triggered only once (the "~" is used). Thus if the character in the input is not "(" the pattern part matches and the program is executed.

The program checks if it is executed at the first position: `argpos =0 ifthen`. If it is executed at the first position, then the first character in the input is not "(", no arguments follow and the collection of arguments is aborted by: `abort`.

If the first character is "(", then the program will be executed later than the first position and the `abort` will not be executed.

The second line:

```
0_pattern:1;[,)];end
```

defines a pattern part consisting of a single character, whereby this character is either a ",", or a ")". If the input matches, then this is the end of an argument. The argument is then put on the stack by: `end`.

The third line:

```
0_pattern:1;[(,];begin
```

defines a pattern part consisting of a single character, whereby this character is either "(" or a ",". If the input matches, then this is the begin of an argument. The argument start position is stored by: `begin`.

The last line:

```
0_pattern:1;);stop
```

defines a pattern part consisting of a single character, whereby this character is a ")". If the input matches, then this means the end the argument collection. The argument collection is stopped by: `stop`. After this the macro has collected all arguments and continues.

It can be seen that different pattern parts are triggered at the same time by the same input. The order of the definition of the pattern parts therefore plays an important role.

#### 4.4.2 Priorities

Every pattern part that matches will trigger the related program. Thus multiple pattern parts can trigger their programs at the same position. The order in which these programs execute can have a big influence on the result, therefore the order of execution is:

Pattern parts are triggered in order of definition.

In the example above the `","` executes both the `end` and `begin`. This character is the separator between arguments. The previous argument should first be put on the stack by the `end` and the next argument can then be started by the `begin`. Thus the pattern part with the `end` is defined before the pattern part with `begin`. In a similar way the argument collection should put the last argument on the stack before it stops.

The issue of priority can also be overcome by a different pattern definition:

```
0_pattern:1;[@00-@27@29-@ff]~;argpos =0 ifthen abort
0_pattern:1;;;end begin
0_pattern:1;(begin
0_pattern:1);end stop
```

However for more complex patterns the priority still plays a role.

## 5 Macros and patterns consisting of multiple parts

### 5.1 Introduction

In the previous chapters the flexibility of macro names and patterns with multiple values on a single character position and the use of one or more characters has been explained. This flexibility is normally sufficient for most tasks.

In this part further possibilities are explained. These further possibilities described in this part give the names the same functionality as regular expressions known from some programming languages or software.

In this text the multiple parts that are linked together are called pattern expressions, because the name regular expression is mostly used for a notation to describe sets of character strings.

### 5.2 Technical background

Regular expressions are normally implemented with algorithms specially made for these. In `m0` the bitap algorithm is still used for macros and patterns consisting of multiple parts. The use of this has an influence on the way to construct a name from the multiple parts.

The multiple parts are constructed in the same way as the normal macros or pattern parts, but they are not started by the first character of the part. The parts are started when another part has been detected. By linking the parts together, the second part is activated when the first part has triggered.

Thus to define multiple parts you have to:

- Define macro or pattern parts.
- Link these parts.

### 5.3 Defining a part

The name of a part has all the possibilities mentioned under the variable length macros.

The difference is that a starting or intermediate part will not do a macro expansion. Normally only the last part is defined as a macro and will do the expansion. It will normally replace the whole piece of text of the different parts with the definition. However this can be defined differently.

If a part will start at the first character (starting part) or not (intermediate part), can be set during the definition of the part.

In conclusion three types of parts exist:

1. A starting part; the first part of the expression.
2. An intermediate part; optional parts between the start and end parts in the expression.
3. An end part; the part that executes the macro expansion.

There is no limit to the number of starting, intermediate or end parts. However the performance will be influenced if the total number of characters used by the parts become large. The parts are placed in the algorithm for vlm macros and every time a 64 character border is exceeded, an extra word / register is needed that will slow down the execution speed of the vlm algorithm.

## 5.4 Linking parts

The linking of parts make the parts work as a pattern expression.

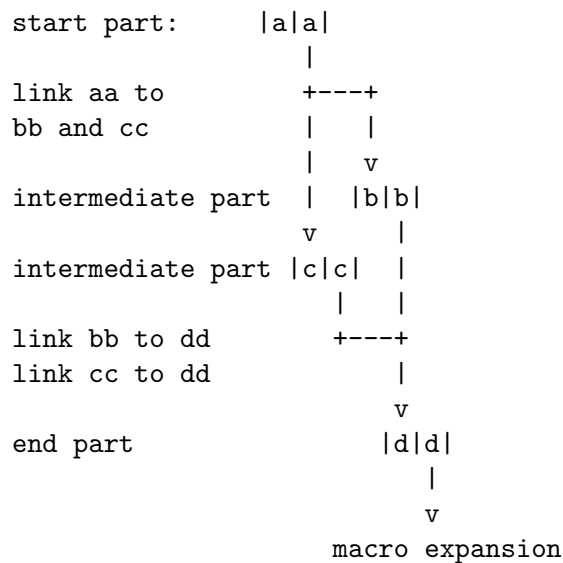
Linking defines which part will start or activate another or other parts.

The following example is a figure of the linking of a small pattern expression. This example has:

- 1 starting part: **aa**
- 2 intermediate parts: **bb** and **cc**, these are alternatives of each other.
- 1 end part: **dd** that will do the macro expansion.

A regular expression for this would be e.g.: **aa(bb|cc)dd**.

In the figure the arrows (**-->**) indicate the links:



The **aa** start part will start on the first character. If the **aa** is in the text, this part will trigger the start of intermediate parts **bb** and **cc** using the links **aa** to **bb** and **cc**. If the text contains an **aabb** or **aacc** the intermediate part **bb** or **cc** will trigger the start of end part **dd** using the link **bb** to **dd** or **cc** to **dd**. If the text further contains a **dd** the end part will trigger a macro expansion.

This example thus shows that using parts and links to create pattern expressions a same functionality as using regular expressions can be achieved.

The example shows only a very basic linking. The linking can be more elaborate to have e.g. zero or one time a part, or one or more times a part, or more complicated OR and AND combinations.

As an example a part of a pattern expression with a one or more part:

```

link from other part      >-----+
                           |
link from bbb to start of bbb  +-----+
                           v         |
intermediate part           |b|b|b|  |
                           |         |
link from bbb also to other part  +---+--->

```

It will have been triggered at least once before it triggers another part. When it triggers it also starts itself.

## 5.5 Macro with multiple parts expansion

The text that matches the pattern expression should be replaced by the definition of the macro. The algorithm in `m0` uses backtracking when a pattern expression for a macro is matched. The way the backtracking runs can be set during the definition of the parts.

There can sometimes seem to be ambiguities what the pattern expression exactly should match. Therefore the following points should bring clarity in the matching of the pattern expression:

- The macro expansion is triggered on the first position that matches the pattern expression. It is therefore not useful to end the name of the end part with e.g. a one or more character (`[]+`). The first moment a character matches this, the macro expansion is triggered and further possible matching characters are not included in the match. A way to overcome this is defining a different character (like e.g. a white-space) after the one or more character. This different character can be excluded from the macro expansion using the post-size of the macro.
- The length matching of a `vlm` part can be set in the options when defining the part. See Section 6.2 [Functions for defining macros], page 25, and Section 6.3 [Functions for defining `vlm` parts], page 30.
- The options of a part also determine if the length of a previous part should be included. Normally it is the case with pattern expressions that the length of the previous part should be included. The start part will obviously never include the length of a previous part.
- If multiple parts trigger a next part at the same position, then the part first defined will be used in the backtracking. The order of definition of parts has thus an influence in this case. Since the linking of parts is not dependent on the order of defining parts, there is complete freedom in deciding the order.

## 5.6 Patterns with multiple parts

Using the multiple parts in patterns (for argument collection or substitution) does not increase the functionality of patterns. Every part will start the related program irrespective of the type of part.

The use of multiple parts in patterns can however reduce repeated matching parts. With a large number of parts it will therefore reduce memory use and increase execution speed. The effort for coding the pattern will probably decrease.

## 6 Builtin functions

### 6.1 Introduction

The builtin functions are not immediately accessible after startup. At startup they must first be linked to a macro name using the `0_define` macro. This is the only macro available at startup.

The *arguments* should be placed on the first stack (stack a) by using a pattern. The pattern 0 is defined at startup. The syntax of this pattern is used in the examples below. The first argument is placed on position 1 on the stack. The position 0 on the stack has the name of the macro.

In the description below the arguments with square hooks ([...]) indicate optional arguments. The arguments are not placed in a defined syntax, because you can determine the syntax!

### 6.2 Functions for defining macros

This is probably the most complex macro to start with, but since it is the first macro to be used it is best to understand first.

```
define name [definition] [builtin function] macro-settings [builtin]
           [argument-pattern] [substitute-pattern] [program-instructions]
           [macro-set] [vlm-options]
```

This function defines a new macro. The `0_define` macro can be used for this at startup. For an overview how the options for the different parts function see Section 3.7 [The complete working of macros], page 11.

The *name* is the name of the new macro, must exist and have at least one character. The *name* can consist of any character or byte or a set of characters or bytes. Characters can be grouped with straight hooks (left hook [ and right hook ] are the default) to form a set at the specific position in the name. For example:

```
[Nn]ame
```

means the word "name" starting with a small or capital n. Also special characters defined with `specialc` (see Section 6.6 [Functions for setting symbols], page 36) can be used to define sets or strings.

The *name* can have a maximum of 64 positions. This is a limit invoked by the implementation and can not be overcome without substantial changes to the program.

The [*definition*] replaces the macro name. The [*definition*] is normally used in user macros and normally not when linking to builtin macros. However it can be used with builtin macros to output a text different from the output of the builtin. Special codes or words (e.g. \$1 as in `m4`) can be used to fill in the arguments of the macro.

The [*builtin function*] determines the builtin function to be called. This function will be called irrespective of the other options and settings. The pattern for collecting arguments and / or the program should have placed all the necessary arguments on the stack for the correct execution of the builtin function.

The *macro-settings* should be set, otherwise the default (**nn00**) will be used. This default is normally not very useful. The *macro-settings* is a string of a specific length with characters at a specific position having a specific meaning. The positions in the string have the meanings:

- 1            Is the definition recursively processed (**r**) or not (**n** or anything other than **r**) by the macro processor.
- 2            Is recursion for macro recognition during processing of the arguments active (**r**) or not (**n** or anything other than **r**).
- 3            Number (hexadecimal; 0 – f) defining the pre-size of the macro name.
- 4            Number (hexadecimal) defining the post-size of the macro name or the character "**S**".

The "**S**" sets the post-size to the length of the macro minus the pre-size. This can be used to detect the start of a pattern by a macro and include the macro in the argument processing, especially by using variable length macros. In this case, the macro name in argument 0 is as if the post-size is zero.

A warning:

Using the "**S**" makes the macro have a length of zero when processing the text. This has the consequence that aborting the argument processing will lead to an infinite loop of detecting the macro again. This could be overcome by setting a proper virtual character.

- 5            Optional virtual character. This character is output to the algorithm but not to the output. This can be used if a pre-size is set in macros so that these macros can be recognised after a preceding macro. Mostly for emulation of macros as words surrounded by white space. An example of the difference (see the difference at the "**rr00**"):
 

```
# A simple define.
0_define:Define;;;define;nr01;0;;stack_d "" putarg3 "rr00"
putarg4 "0" putarg5 "" putarg6 "" putarg7

Define:test;Hello World!
Define:aa;te
Define:bb;st

aabb

# A simple define with virtual char.
0_define:Define_;;;define;nr01;0;;stack_d "" putarg3 "rr00 "
putarg4 "0" putarg5 "" putarg6 "" putarg7

Define_:cc;te
Define_:dd;st
```

```
ccdd
```

This example will result in:

```
# A simple define.
```

```
Hello World!
```

```
# A simple define with virtual char.
```

```
test
```

An example of *macro-settings*:

```
nr01
```

The meaning of this is: no macro recursion, recursion during argument processing, no (0) pre-size, post-size of 1.

The [*argument-pattern*] is the name of the pattern to be used for collecting arguments.

The [*substitute-pattern*] is the name of the pattern to be used for substitution of special characters with the arguments in the definition. If this option is not used the default argument substitution is used. If this option exists but the definition string is empty, then the first argument is used in place of the definition. This can be used for macros such as the **format** macro of **m4**.

The [*program-instructions*] is a program that is executed just after the arguments are collected and before the builtin macro is executed or argument substitution. This can be used to set arguments to fixed values.

The optional [*macro-set*] is the name of a set of macros that will be used to define the macro in. If this option is not set, the macro is defined in the current active set of macros. The default macro set name at start is 0. If the macro set does not yet exist, it will be initialised.

The optional [*vlm-options*] is a string of a specific length with characters at a specific position having a specific meaning. This option is only useful for variable length macros (vlm). The positions in the string have the meanings:

- |   |                                                                                                                                                                                                                   |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Sets the way in which the length of a vlm is determined. Possibilities are:                                                                                                                                       |
| L | The longest match of the vlm is used.                                                                                                                                                                             |
| S | The shortest match of the vlm is used.                                                                                                                                                                            |
| E | The shortest match of the vlm with an extension is used. This is similar to the S option, but will include a variable length part on the first position. This is the default if no [ <i>vlm-options</i> ] is set. |
| F | The fixed length of the <i>name</i> is used. Normally only used with macros that have a fixed length.                                                                                                             |

An example result with shortest match:

```
0_define:ab[ab]*c;d;;nn00;;;;;S
```

```
abaac
```

```
ababc
```

will result in:

```
d
```

```
abd
```

An example result with longest match:

```
0_define:ab[ab]*c;d;;nn00;;;;;L
```

```
abaac
```

```
ababc
```

will result in:

```
d
```

```
d
```

- 2 Option to set if this macro should be triggered using a link by another macro part.

**n** This is a normal macro. This is the default if no *[vlm-options]* is set.

**e** This is an end macro part that is only triggered by a link.

**l** This is an end macro part that is only triggered by a link. The length of the previous macro part is included in the length of the macro.

A warning:

Not all combinations make sense when using multi part macros. If the length of a previous part of a macro is included (option **l**), the short options (**S** and **E**) for determining the length of a macro are normally not useful.

Setting the *[vlm-options]* will define the macro as a vlm even if the *name* is not a vlm. If the macro was previously not a vlm, then setting the *[vlm-options]* will not change the macro to a vlm.

Only one macro is available at the start of **m0**. This is the macro **0\_define** and uses the pattern **0** for argument collection. It is defined as:

```
0_define:0_define;;define;nr01;0
```

This definition has

**0\_define:**

(second **0\_define:**) as the name of the macro. It is including the start (**:**) of the argument collection.

**define**      Is the builtin function.

**nr01**          The *macro-settings* meaning: no macro recursion (not necessary since no output), recursion during argument processing, no (0) pre-size, post-size of 1 (this is the : of the argument collection).

0              The pattern (named 0) for argument collection.

Output: none.

**push** *name* [*definition*] [*builtin function*] *macro-settings* [builtin]  
           [*argument-pattern*] [*substitute-pattern*] [*program-instructions*]  
           [*macro-set*] [*vlm-options*]

Similar to **define**, but saves the previous definition. The previous definition can be retrieved with **pop**. It redefines the user macro *name* using the same arguments as used in **define**.

Output: none.

**undefine** *name* [*macro-set*] [builtin]

Deletes the macro *name*. The macro does not exist anymore after this.

If the [*macro-set*] is used then the macro is deleted from this macro set. If [*macro-set*] is not defined, then the current macro set is used. If the macro set named in [*macro-set*] does not exist, an error is output.

Output: none.

**pop** *name* [*macro-set*] [builtin]

Retrieves the previous definition of the macro *name*.

If the [*macro-set*] is used then the macro is retrieved using this macro set. If [*macro-set*] is not defined, then the current macro set is used. If the macro set named in [*macro-set*] does not exist, an error is output.

Output: none.

**copy** *from-name to-name* [*from macro-set*] [*to macro-set*] [builtin]

Copies the macro or **mcall** with *from-name* to the macro with *to-name*.

If the [*from macro-set*] is used then the macro is retrieved from this macro set. If the [*to macro-set*] is used then the macro is copied to this macro set. If the [*from macro-set*] or the [*to macro-set*] is not defined, then the current macro set is used.

If the macro set named in [*from macro-set*] or [*to macro-set*] does not exist, an error is output.

Output: none.

**pushcopy** *from-name to-name* [*from macro-set*] [*to macro-set*] [builtin]

Similar to **copy**, but saves the previous definition of the *to-name* macro. The previous definition can be retrieved with **pop**.

Output: none.

**set\_info** *name info* [*macro-set*] [builtin]

With this function a special information text can be set in a macro. This information text can be retrieved with the function **info** by outputting the ninth argument.

The *name* is the name of the macro for which to set the information text.

The *info* is the information text.

If the [*macro-set*] is used then the macro from this macro set is used. If [*macro-set*] is not defined, then the current macro set is used. If the macro set named in [*macro-set*] does not exist, an error is output.

Output: none.

### 6.3 Functions for defining vlm parts

The functions described in this part are used specifically for variable length macros (vlm). They extend the macros with the same possibilities the patterns have.

**part\_m** *part-name* [*vlm-options*] [*macro-set*] [builtin]

This function defines a macro part with *part-name*. The *part-name* has the same possibilities as the *name* in macros.

The optional [*vlm-options*] is a string of a specific length with characters at a specific position having a specific meaning. It is similar to the [*vlm-options*] in the **define** function. The positions in the string have the meanings:

- 1            Sets the way in which the length of this (vlm) part is determined. Possibilities are:
  - L            The longest match of the vlm is used. This is the default if no [*vlm-options*] is set.
  - S            The shortest match of the vlm is used.
  - E            The shortest match of the vlm with an extension is used. This is similar to the S option, but will include a variable length part on the first position.
  - F            The fixed length of the *part-name* is used.
- 2            Option to set if this macro part should be triggered using a link by another macro part.
  - s            This macro part starts at the first character like normal macros.
  - i            This is an intermediate macro part that is only triggered by a link.
  - l            This is an intermediate macro part that is only triggered by a link. The length of the previous macro part is included in the length of the macro. This is the default if no [*vlm-options*] is set.

The optional [*macro-set*] is the name of a set of macros that will be used to define the macro part in. If this option is not set, the macro part is defined in the current active set of macros. If the macro set does not yet exist, it will be initialised.

Output: none.

**tag\_m** *tag-name* [*macro-set*] [builtin]

This function tags the current position of the vlm parts. A vlm part can be a part defined by the **part\_m** function or a vlm macro. It is similar to the **tag\_p** function in Section 6.5 [Functions for defining patterns], page 35.

The position can be referenced by *tag-name*. The *tag-name* has a maximum of 8 characters. The *tag-name* used in both this function and the **tag\_p** function should be unique, because the list with tags is used by both functions.

To link from a vlm part, the tag should be placed directly after the vlm part. To link to a vlm part, the tag should be placed directly in front of the vlm part.

The tags are used in the **link\_prt** function.

The optional [*macro-set*] is the name of a set of macros that will be used for the tag. If this option is not set, the tag is placed in the current active set of macros. If the macro set does not yet exist, it will be initialised.

Output: none.

**link\_prt** *from-tag to-tag* [*to-tag*] [builtin]

This function links a macro part or a pattern part (see Section 6.5 [Functions for defining patterns], page 35) to other parts. It links the part tagged with *from-tag* to the part tagged with *to-tag*. The part linked to will then start when the from part has triggered.

Multiple parts can be linked to in one function call.

Output: none.

## 6.4 Macro sets

Macros are defined in a set of macros. This allows to have macros active at different times when processing an input and not having to split an input and processing the parts separately.

Only one set of macros can be active at one time. It is thus not so flexible as to be able to activate and deactivate certain macros.

The number of macro sets has no impact on normal performance. There is however more memory used to store the macro sets.

**macroset** *macro-set* [builtin]

Switches to the macro set with name *macro-set*. This macro set must already exist, otherwise this is an error and the macro set is not changed.

Only macros available in the new macro set can be used after switching. The function **define** can for example be used to define macros in a macro set.

Output: none.

**defmcall** *name* [*called-macro*] [*called macro-set*] *mcall-settings* [builtin]  
*argument-pattern* [*program-instructions*] [*macro-set*] [*vlm-options*]

Defines a macro that will call a macro (**mcall**) in a macro set. The **mcall** enables the call to a macro (or an **mcall**) in another macro set without first switching to this macro set using the **macroset** function.

The `mcall` is similar to a macro, except that it calls another macro instead of a built in function or a definition. It calls the macro with the name in `[called-macro]` or when this is empty the macro name in first argument on the stack when the `mcall` is invoked. It will collect the arguments using the *argument-pattern* and run the `[program]` before the called macro is executed.

The called macro will not perform argument collection since this has already been done by the `mcall`. This has the implication that the syntax for argument collection as defined in the `mcall` (by the *argument-pattern*) is used and not the syntax as defined in the called macro.

Apart from the argument collection, the called macro will execute all the defined settings in the macro set of the called macro (the `[called macro-set]`). This means that if macros are used in the definition string of the called macro, these macros should exist in the called macro set if they should expand there and not expand in the current macro set.

The *name* is the name of the new macro call, must exist and has the same possibilities as a macro name.

The `[called-macro]` is the name of the called macro. The definition of the called macro at the time of the `mcall` is used. The `[called-macro]` should be written in a form that would normally be recognized in the processed text. If the `[called-macro]` is not defined, the first argument on the stack is used as the called macro name. If the called macro does not exist an error will result.

If the first argument on the stack is used as the called macro name, the stack is adapted so that the called macro will see the arguments shifted by one. The first argument for the called macro is the second argument for the `mcall`. For the called macro, its name is on position zero of the stack.

The called macro is expanded in the `[called macro-set]`. If the `[called macro-set]` is not defined the current macro set is used as the macro set for the called macro. If the called macro set does not exist, it will be initialised.

The *mcall-settings* is the same as the *macro-settings* in `define` and should be set. The recursive processing of the definition can be overridden by the called macro using the instruction `recur_y` or `recur_n`.

The *[argument-pattern]* is the name of the pattern to be used for collecting arguments in the same way as for macros.

The *[program-instructions]* is a program that is executed just after the arguments are collected in the same way as for macros.

The optional *[macro-set]* is the name of a set of macros that will be used to define the macro call in. If this option is not set, the macro call is defined in the current active set of macros. If the macro set does not yet exist, it will be initialised.

The optional *[vlm-options]* is the same as the *[vlm-options]* in `define`.

The `undefine` function can be used to delete the `mcall`.

Output: none.

The following figure is a symbolic overview of the working of `mcall` when the `[called-macro]` holds the name of a macro:

```

in: | | | |m|a|c|r|o| |n|a|m|e|(|a|r|g|u|m|e|n|t|s|)| | |
    +-----+-----+
        macro name      : :      arguments
    +-+                +-+
pre-size              post-size    |
                        : :        v
                        : :
                        | | | | | | | | | | | |
                        +-----+-----+
                                pattern for collecting arguments +
                                    small programs

                                |
                                |
                                +-----> argument n
                                -----
                                ...
optional program for stack manipulation --> -----
                                           argument 3
                                           -----
                                           argument 2
                                           -----
                                           argument 1
                                           -----

                                |
                                v

out: | | | |m|a|c|r|o| |o|u|t|p|u|t| | | | | | | | |
    +-----+-----+
    +-+ replacement of macro
pre-size

```

The following figure is a symbolic overview of the working of `mcall` when the `[called-macro]` is empty:

```

in: | | | |m|a|c|r|o| |n|a|m|e|(|a|r|g|u|m|e|n|t|s|)| | |
    +-----+-----+-----+
        macro name      : :      arguments
    +-+                +-+
pre-size              post-size    |
                        : :        v
                        : :
                        | | | | | | | | | | | |
    +-----+-----+-----+
                                pattern for collecting arguments +
                                small programs
                                |
                                |
                                +----->      argument n
                                -----
                                ...
optional program for stack manipulation --> -----
                                -----
                                ...
                                -----
                                <---          ...
                                -----
new argument 2              <---          argument 3
                                -----
new argument 1              <---          argument 2
                                -----
new argument 0 <-+- name of macro <--- argument 1
                        |           |
                        v           |
called macro    <-+
                        |
                        v
out: | | | |m|a|c|r|o| |o|u|t|p|u|t| | | | | | | | | |
    +-----+-----+-----+
    +-+ replacement of macro
pre-size

```

`pshmcall` *name* [*called-macro*] [*called macro-set*] *mcall-settings* [builtin]  
*argument-pattern* [*program-instructions*] [*macro-set*] [*vlm-options*]

Push macro call is similar to `defmcall`, but saves the previous definition. The previous definition can be retrieved with `pop`. It redefines the macro call *name* using the same arguments as used in `defmcall`.

Output: none.

## 6.5 Functions for defining patterns

**pattern** *name pattern-part program-instructions* [*priority*] [builtin]

This function adds *pattern-part* to a pattern named *name*. A pattern with *name* is automatically initialised when not already used before.

See Chapter 4 [Patterns], page 17, for the possibilities of the *pattern-part*.

The program to be executed when the pattern part is triggered is defined in *program*. Normally every pattern part has a program, but it can be left out.

The [*priority*] is used in a special case. Normally a pattern will not be triggered if at the same position a macro is triggered. This is based on the logic that a macro will change the text and that at this position no pattern should also be triggered with the old input. The [*priority*] can overrule this behaviour by setting a priority of 1 or larger.

An example of a special case where this is used is the pattern for argument collection in the m4 emulation:

```
0_pattern:m4;[(,)[@00- ]*[@!-@ff];stack_a begin-1 ;1
```

This pattern part sets the begin of an argument after optional whitespace ([@00- ]\*) and has to include a first non whitespace character ([!-@ff]). It will set the begin of the argument directly before the non whitespace character (the **begin-1** instruction). The non whitespace can however also be a macro, like the quote macro. In this case both the pattern part and the macro should be triggered.

Output: none.

**append\_p** *name pattern-part program-instructions* [*priority*] [builtin]

This function appends *pattern-part* to the previous pattern part in the pattern named *name*. It has the same arguments as **pattern**.

The pattern-part matching will only start when the previously defined pattern-part has triggered.

Output: none.

**inter\_p** *name pattern-part program-instructions* [*priority*] [builtin]

This function defines an intermediate *pattern-part* in the pattern named *name*. It has the same arguments as **pattern**.

The intermediate pattern-part matching will only start when another pattern-part has triggered using a link to this intermediate pattern-part.

Output: none.

**tag\_p** *name tag-name* [builtin]

This function tags the current position (before and / or after a pattern-part) in the pattern named *name* with *tag-name*. The *tag-name* has a maximum of 8 characters.

To link from a pattern part, the tag should be placed directly after the pattern part. To link to a pattern part, the tag should be placed directly in front of the pattern part.

The tags are used in the **link\_prt** function.

Output: none.

**clr\_pat** *name* [builtin]

Clears an existing pattern named *name*.

The pattern still exists after this function call, but it is empty. This can be used to redefine a syntax.

Output: none.

**copy\_pat** *from to* [builtin]

Copies an existing pattern named *from* to *to*.

This can be used to reuse a pattern and extend it further in a new pattern.

Output: none.

**program** *name program-instructions* [builtin]

Defines a program that can be called from other programs.

The *name* should have a maximum of 8 characters. This is the name that is used to call this program using `@name`. The program should already be defined before the call instruction can be used, because the instructions are converted into internal codes when a program is defined. If the program is not yet known, then it is not possible to convert the call to an internal code.

Output: none.

## 6.6 Functions for setting symbols

**specialc** *special-character string* [builtin]

This function defines a special character (*special-character*) that expands to *string* in the name of macros or in pattern-parts.

This can be used as a short means of writing repeatedly longer strings. The special character is used by preceding it with `"\"` in macro names or pattern parts. This preceding character can be set using the **charpat** function.

The *special-character* is a single character and can have any byte value.

A redefinition of the special character has no influence on already use of it in macro names and pattern parts.

Output: none.

**charpat** *pattern-settings* [builtin]

This function defines the characters with a special meaning used in macro names and pattern parts.

The *pattern-settings* is a string of a specific length (9 characters) with characters at a specific position having a specific meaning. The positions in the string have the meanings:

- 1            Is the character used as opening brace for character sets. This is `"["` by default.
- 2            Is the character used as closing brace for character sets. This is `"]"` by default.

- 3        Is the character used to indicate range in a character set. This is "-" by default.
- 4        Is the character used to indicate a special character. This is "\" by default.
- 5        Is the character used to input a character by hexadecimal ASCII code. This is "@" by default.
- 6        Is the character used for the one or more characters in a character set. This is "+" by default.
- 7        Is the character used for the zero or more characters in a character set. This is "\*" by default.
- 8        Is the character used for the zero or one character in a character set. This is "?" by default.
- 9        Is the character used for a one time trigger of a character set. This is "~" by default.

The default *pattern-settings* is thus: []-\@+\*?~

Output: none.

**chararg** *argument-settings variable-quote-start variable-quote-end* [builtin]  
*variable-separator* [*macro-set*]

This function is used to define the characters for the argument substitution when using the the default argument substitution. Only a single character can be selected. This is similar to **m4**.

The *argument-settings* is a string of a specific length (5 or 6 characters) with characters at a specific position having a specific meaning. The positions in the string have the meanings:

- 1        This is a hexadecimal number for the amount of digits after an argument character. A number from 1 – 5 indicates the maximum number of digits is 1 – 5 with a minimum of 1 digit. A number from a – d sets the number of digits to exactly 2 – 5. The remaining numbers 0, 6 – 9, e and f are reserved.
- 2        Is the character used for arguments in the definition. This is \$ by default. The number after this character in the definition is the entry of the stack.
- 3        Is the character used to output all arguments. This character is put after the first character in a definition. This is \* by default.
- 4        Is the character used to output all arguments with quotes. This character is put after the first character in a definition. This is @ by default.
- 5        Is the character used to output the number of arguments. This character is put after the first character in a definition. This is # by default.
- 6        Is the optional character used for arguments from the other stacks in the definition. This is \$ by default. An a – f after this character in the definition selects the stack. The number after this is the entry of the

stack. If this is not set, the arguments from the other stacks can not be selected.

The *variable-quote-start*, *variable-quote-end* and *variable-separator* are the numbers of the variables where the characters or strings of the start of a quote, the end of a quote and the separator of arguments are stored. See also the function `set_var` in Section 6.8 [Functions for using variables and system information], page 39, for defining these characters or strings.

The optional *[macro-set]* is the name of a set of macros that will be used to set the argument characters for. If this option is not set, the argument characters are set in the current active set of macros. The default macro set name at start is 0. If the macro set does not yet exist, it will be initialised.

The default is defined as:

```
0_define:0_chars_args;;;chararg;nr01;0
```

```
0_chars_args:1$*@$;1;2;3
```

Output: none.

## 6.7 String functions

**strindex** *string substring* [builtin]

This function returns the position of *substring* in the *string*.

The first position has number 0.

If the *substring* is not found the number -1 is returned.

Output: position of the *substring* in *string* or -1.

The output is also put in argument 3.

**substr** *string from [length]* [builtin]

Expands to the substring of *string*, which starts at *from*, and extends for *[length]* characters, or to the end of *string*, if *[length]* is omitted. The starting index of a string is 0.

Output: the substring

The output is also put in argument 4.

**strtrans** *string chars [replacement]* [builtin]

Translates the characters in *chars* in the *string* to corresponding characters in *[replacement]*. Corresponding characters have the same position in *chars* and *[replacement]*.

If a character in *chars* has no corresponding character in *[replacement]* it is deleted.

Also a range of characters can be given, e.g.: **a-z** or **z-a**.

Output: translated string

The output is also put in argument 4.

**num2chr** *number [number] ... [number]* [builtin]

This function returns the character with the ASCII of *number*. The number is therefore at least 0 and maximum 255. If the number is outside of this range a space is output.

Multiple `[number]` can be used and result in a string with the ASCII characters of the numbers.

Output: single character or string of characters

The output is also put in argument 2.

**num2str** *number* [*radix*] [*width*] [builtin]

Converts a *number* to a string.

By default the *number* is output as decimal. It can also be output in another base by setting [*radix*]. The [*radix*] can be 2 – 36. The radix prefix is not in the output.

The minimum width is set by [*width*]. The number is padded with zeros.

A custom output can be made by using a definition string and using outputs placed in arguments. The arguments that can be used for this are:

- 4           The radix prefix.
- 5           Positive binary output. The *number* is treated as unsigned.
- 6           A minus sign if the *number* is negative.
- 7           The leading zeros.
- 8           String of the *number* as in the output without leading zeros or minus sign.

Output: string of *number*

## 6.8 Functions for using variables and system information

A set of variables to store strings is available. They can be used to store information without using macros. They are also used to store the characters or strings of multiple arguments and quoting for the default argument substitution function.

These variables are referenced by a number from 1 to 255. Negative numbers are used to get certain system or process information.

**set\_var** *number string* [builtin]

Sets the variable with *number* to the value of *string*. The *number* can be from 1 to 255.

Output: none.

**get\_var** *number* [builtin]

Outputs a string from the variable with *number*.

The *number* can be:

- 1 - 255   Gets the string from the variable with *number*.
- 1       Gets the diversion number.
- 2       Gets the string of the operating system type e.g. **unix**.
- 3       Gets the current global line number. This counts all new lines starting from the first input file.

- 4 Gets the current local line number. This counts all new lines in the current input file.
- 5 Gets the current input file name.
- 6 Gets the program name.
- 7 Gets a string with all the options after the -- on the command line.
- 8 Gets a string with all the options on the command line after the -- and directly before the input file.
- 9 Gets the return value from the last command executed by the function `shell`.

The output is also put in argument 2.

Output: the selected string.

## 6.9 Functions for getting macro information

`ifmacro? name [yes] [no] [macro-set]` [builtin]

This function checks if the macro with *name* exists in the macro set *[macro-set]*. If the macro exists the string in *[yes]* is output otherwise the string in *[no]* is output.

If *[macro-set]* is not defined, then the current macro set is used. If the macro set named in *[macro-set]* does not exist, an error is output.

Output: the string in *[yes]* or in *[no]*.

`info name [macro-set]` [builtin]

This function outputs the definition string of the macro with *name* or the called macro in case of an `mcall` with *name* defined in the macro set *[macro-set]*.

If *[macro-set]* is not defined, then the current macro set is used. If the macro set named in *[macro-set]* does not exist, an error is output.

The other settings of the macro are put in arguments. They can be used for a customized output by using a definition. The arguments that can be used for this are:

- 2 Definition string or called macro.
- 3 The builtin function or macro set of the called macro.
- 4 *macro-settings*
- 5 Pattern for argument collection.
- 6 Pattern for argument substitution.
- 7 The optional program.
- 8 Macro set of the macro.
- 9 This will output the information text of the macro (see also the function `set-info`). Or if the information text is not set, depending on the type of macro this will be "macro" if the macro is a macro or "mcall" if the macro is a macro call.

Output: definition or called macro of *name*

## 6.10 Functions for input and output

Diversions are temporary memory buffers for storing the output. The diversions can be retrieved at a later time or are automatically retrieved at the end of the input files.

If a diversion is set, the output that would go to the normal output is stored in a buffer. Thus if a diversion is set multiple times inside a macro, only the last setting of the diversion has an influence on the output.

The memory buffers have a diversion number of one or larger. Diversion number 0 means the normal output. At the end of the input files the remaining diversions are automatically output in ascending order.

**divert** [*number*] [builtin]

Sets the diversion to [*number*]. If the [*number*] is not given, it will be by default 0, the normal output.

A negative [*number*] will discard all output.

Output: none

**undivert** [*number*] [*number*] [builtin]

The **undivert** will empty the diversion with [*number*] into the output. Multiple [*number*]'s can be given to undivert. If no number is given, all diversions are undiverted.

The contents of the diversion can still be recursed for macros if the *macro-settings* is properly set.

Output: diversion buffer

**include** *file* [builtin]

The **include** will read the *file* and output its contents.

The *macro-settings* of the definition will (like every macro) determine if the contents of the *file* is recursed for macro recognition or not. Using a definition or definition fill pattern does not seem useful.

If *file* does not exist an error is output and program execution is stopped.

Output: contents of *file*.

**sinclude** *file* [builtin]

Same as **include** but will fail silently if the *file* does not exist.

Output: contents of *file*.

**at\_last** *string* [builtin]

Puts the *string* in a buffer that will be append to the input after the the end of the last file. Similar to the **divert** function, but it will be automatically undiverted and used for macro recognition.

The function appends additional strings to the previous strings. It works thus like a FIFO buffer in the same way as the **divert** function.

Output: none.

**at\_first** *string* [builtin]

Puts the *string* in a buffer that will be prefixed to the input every time a new file from the command line is opened. Similar to the **at\_last** function, but the buffer will not be emptied after a prefix.

Can be used to e.g. set defaults or output a heading per file.

The function appends additional strings to the previous strings. It works thus like a FIFO buffer in the same way as the **divert** function.

Output: none.

**shell** *command* [builtin]

Executes the *command* in a shell. The *command* is executed in a shell with the stdin of **m0** as input and the stdout is redirected as input to **m0**.

Output: the output of stdout of the command.

**tempfile** *template* [builtin]

Creates a temporary file using a *template*. The *template* should have six X (e.g. **tmp/docXXXXXX**) at the end for placement of a random string. If these X are not in the *template*, the *template* will be expanded to hold the random string with six characters.

The file is created and left empty. The function will output the name of the temporary file for further use.

Output: name of the temporary file.

## 6.11 Functions for handling of errors

**errprint** *message* [*message*] [builtin]

This function will output *message* and following optionally multiple [*message*] to standard error.

Output: none

**exit** *errorcode* [builtin]

This will exit the **m0** program directly with an *errorcode*. The *errorcode* should be between 0 and 255.

Output: none

## 6.12 Some functions are missing?

Several functions that exist in e.g. **m4** are not available in **m0**. Examples are:

- Conditionals like **if else**.
- Arithmetic like **decr** to decrement a value or **eval** to evaluate a mathematical expression.
- Quoting and comments
- Shift macro
- String functions like **len** for the length of a string or **format** to output formatted strings.

These functions are not available because they can be implemented using macros, patterns and programs. Examples of these functions can be found in Section 8.2 [M4 emulation example], page 65.



## 7 Programs for patterns and macros

### 7.1 Introduction to programs

A part of the flexibility of `m0` is based on the possibility of using small programs executed when a pattern or macro is found.

The programming environment is forth like with stacks to be used for variables and forth like instructions.

Every macro starts its own stack environment for the collection of the arguments. It is thus normally not possible to transfer information between macros through the stack.

### 7.2 Stacks

The programming environment contains 8 stacks. The 8 stacks are 8 separate stacks for data and 8 separate stacks for instructions.

The instruction stacks can be used to implement parsers for e.g. infix notation. Special instructions are used to put instructions on the instruction stacks. Normal instructions execute directly on the active stack.

Instructions placed on the instruction stacks work on the corresponding data stacks.

An overview of the stacks:

```
|n| <- top
|. |
|. |
|2|
|1| | | | | | | | | | | |
|0| | |0| | |0| | |0| | |0| | |0| | |0| | <- bottom
-----
|d|i| |d|i| |d|i| |d|i| |d|i| |d|i| |d|i| |d|i|
  a      b      c      d      e      f      g      h

a - h = stacks
d = data stack
i = instruction stack

0 = bottom of stack, a[0] = name of macro
a[1] = first argument
a[n] = argument n
```

In the following descriptions of instructions the values are indicated as:

- top, or last value, or second value when two values are taken from the stack.
- top - 1, value below the value at the top of the stack, or first value when two values are taken from the stack.
- top - 2, value below the top - 1 value.

- bottom, value at position 0.

After the macro is started the stacks are empty, except for position 0 on stack a that holds the name of the macro.

The stacks are not cleared between the different possible patterns executed inside a macro.

At the end of the macro execution the stacks are deleted, except if a special instruction is used (see Section 7.21 [Manipulation of stacks], page 63).

## 7.3 Programs

Programs are a sequence of instructions, data or calls to other programs.

An instruction, a piece of data or a call are all single words separated by one or multiple spaces or newlines (`\n`). If a word is not one of these, an error is output.

### 7.3.1 Data types

There are 2 types of data that are placed on the stack:

#### Strings

Strings are formed by one or more bytes. A string can be put on the top of a stack by enclosing the string in `"`. An example:

```
"Hello world"
```

puts the the string `Hello world` on the stack.

#### Numbers

Numbers are signed integers with a length of 64 bits. A number can be put on the top of the stack by only using digits (0 - 9) with a possible `-` for negative numbers in the program. An example:

```
10 -100
```

puts the number 10 and the number -100 on the stack.

These types are automatically converted depending on the instructions used. A string can contain a decimal, binary (starts with `0b`), octal (starts with `0`), hexadecimal (starts with `0x`) or radix number (starts with `0r` a number between 2 - 36 and `:`). A number will be converted to a decimal number string.

### 7.3.2 Instructions

If a word is not data or a call, it is assumed to represent an instruction.

All possible instructions are described in the following parts.

It is not possible to define own instructions like in a real forth language. Instead a call to a program can be used.

### 7.3.3 Call to a program

A separate program can be defined by using the function **program** (see Section 6.5 [Functions for defining patterns], page 35). The name used in the definition of the program can be used to call the program from another program.

The call of a program starts with the character **@** and is directly without a space followed by the name given to the program. An example:

```
@hello
```

will call the program with name **hello**.

## 7.4 Selecting a stack

Eight different stacks (**a – h**) are available. The first stack (**a**) is used to place the arguments of a macro. The stack to be used must be selected. The default selection after the macro execution is started is stack **a**. Thus if no other stack than the stack **a** is used it is not necessary to select it.

**stack\_?** [instruction]  
 Selects the new active stack. The **?** can be a letter from **a – h** representing stacks **a** to **h**.  
 pop: 0  
 push: 0

## 7.5 Manipulation on the stack

**dup** [instruction]  
 Duplicates the top entry of the active stack.  
 pop: 0  
 push: 1

**pop** [instruction]  
 Pops and thereby removes the top entry of the active stack.  
 pop: 1  
 push: 0

**pop\_to\_?** [instruction]  
 Will pop the stack to the position of the stack indicated by the number (**?**). The **?** can be a number from **0 – 9**.  
 pop: depending on the stack.  
 push: 0

**swap** [instruction]  
 Swaps the top entry and the top - 1 entry of the active stack.  
 pop: 0  
 push: 0

**swap-12** [instruction]

Swaps the top - 1 entry and the top - 2 entry of the active stack.

pop: 0

push: 0

**set\_base** [instruction]

Sets the base of stack **a** to the value on the top of the stack. The base is used as the bottom when multiple arguments are output in the default argument substitution. This can be used to implement a shift macro, that shifts the arguments.

pop: 1

push: 0

**base\_=\_1** [instruction]

Same as **set\_base** whereby the base is set as 1.

pop: 0

push: 0

## 7.6 Copy between stacks

**copyto\_?** [instruction]

Copy the top entry of the active stack to the top on another stack. The ? can be a letter from **a - h** representing stacks **a** to **h**.

pop: 0

push: 1

**copyfr\_?** [instruction]

Copy the top entry from another stack to the top of the active stack. The ? can be a letter from **a - h** representing stacks **a** to **h**.

pop: 0

push: 1

**putarg?** [instruction]

Copy the top entry of the active stack to position ? of stack **a** where the arguments are stored. The ? can be a number from 0 - 9 representing the name of the macro (position 0) and the first 9 arguments (positions 1 - 9).

pop: 0

push: 0

**putarg** [instruction]

Copy the top - 1 entry of the active stack to the position given by the top of the active stack of stack **a**.

pop: 1

push: 0

**getarg?** [instruction]  
 Copy the entry at position ? of stack **a** to the top entry of the active stack. The ? can be a number from 0 - 9 representing the name of the macro (position 0) and the first 9 arguments (positions 1 - 9).  
 pop: 0  
 push: 1

**getarg** [instruction]  
 Copy the entry of stack **a**, at the position indicated by the value at the top of the active stack, to the top entry of the active stack.  
 pop: 1  
 push: 1

## 7.7 Collecting argument strings

The following instructions are used for collecting arguments. The **begin** instructions will set the start position of the argument. The **end** instructions will set the end position of the argument and copy the argument to the (active) stack. If multiple **begin** instructions are used before an **end** instruction only the first use will set the start position.

The arguments are always pushed to the stack as a string.

**begin** [instruction]  
 Sets the start position of the argument string at the end position of the pattern match.  
 pop: 0  
 push: 0

**begin-?** [instruction]  
 Sets the start position of the argument string at the end position minus the number (?) of the pattern match. The ? can be 1 - 4.  
 pop: 0  
 push: 0

**begin+?** [instruction]  
 Sets the start position of the argument string at the begin position plus the number (?) of the pattern match. The ? can be 0 - 2. This instruction works only correct with a fixed length pattern part.  
 pop: 0  
 push: 0

**end** [instruction]  
 Sets the end position of the argument string at the begin position of the pattern match and copies this string to the top of the stack. This instruction works only correct with a fixed length pattern part.  
 pop: 0  
 push: 1

**end-?** [instruction]

Sets the end position of the argument string at the end position of the pattern match minus ? and copies this string to the top of the stack. The ? can be 0 - 4.

pop: 0

push: 1

**abort** [instruction]

Aborts the collection of arguments. No arguments are collected and the bytes used in the matching till so far are returned to the input. Used when no arguments are detected or no argument start is detected after the macro.

pop: 0

push: 0

**stop** [instruction]

This stops the pattern. Normally used when all arguments are collected e.g. after detecting stop character of arguments. It is necessary to use the **stop** instruction when collecting arguments, otherwise the complete input will be used for the arguments of the macro. This instruction is not needed for the single programs for macros or when using patterns to fill in the definition.

pop: 0

push: 0

**goback?** [instruction]

This instruction has only an effect if used before a **stop**. It sets the input back with the value of ?, which can be 1 or 2. Can be used when the position to stop collecting arguments can only be detected later than the real position to stop. The instruction will increment. Thus using it two times will back up the position with both values.

The value is reset between programs.

pop: 0

push: 0

## 7.8 Program flow

**ifthen** [instruction]

Pops the top of the stack and continues to the next instruction if **true** or skips to the instruction after the next instruction if **false**.

pop: 1

push: 0

**if else endif** [instruction]

Pops the top of the stack and continues to the next instruction if **true** or skips to the instruction after the **else** if **false**. The **endif** indicates the end of the instructions after the **else**. The **else** is optional.

pop: 1

push: 0

|                                                                                                                                                                    |               |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| <b>while</b>                                                                                                                                                       | [instruction] |
| Pops the top of the stack and continues to the next instruction if <b>true</b> or skips to the instruction after the <b>endwhile</b> instruction if <b>false</b> . |               |
| pop: 1                                                                                                                                                             |               |
| push: 0                                                                                                                                                            |               |
| <b>while_st</b>                                                                                                                                                    | [instruction] |
| Continues to the next instruction if the stack is not empty or skips to the instruction after the <b>endwhile</b> instruction if the stack is empty.               |               |
| pop: 0                                                                                                                                                             |               |
| push: 0                                                                                                                                                            |               |
| <b>endwhile</b>                                                                                                                                                    | [instruction] |
| Returns to the <b>while</b> or <b>while_st</b> instruction.                                                                                                        |               |
| pop: 0                                                                                                                                                             |               |
| push: 0                                                                                                                                                            |               |
| <b>nop</b>                                                                                                                                                         | [instruction] |
| No operation. Does nothing.                                                                                                                                        |               |
| pop: 0                                                                                                                                                             |               |
| push: 0                                                                                                                                                            |               |

## 7.9 Getting variables

Sometimes it is necessary to have information of the process available. This information is available in variables that can be accessed and put on a stack.

|                                                                                                                                                                                      |               |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| <b>argpos</b>                                                                                                                                                                        | [instruction] |
| Pushes the current position of the pattern matching to the stack. The first position of a pattern is position 0.                                                                     |               |
| pop: 0                                                                                                                                                                               |               |
| push: 1                                                                                                                                                                              |               |
| <b>argnum</b>                                                                                                                                                                        | [instruction] |
| Pushes the number of arguments collected on stack <b>a</b> to the stack.                                                                                                             |               |
| pop: 0                                                                                                                                                                               |               |
| push: 1                                                                                                                                                                              |               |
| <b>m_depth</b>                                                                                                                                                                       | [instruction] |
| Pushes the current depth of macro calls to the stack. Since the programs always run inside a macro, this should be at least 1. This is already available in the argument collection. |               |
| pop: 0                                                                                                                                                                               |               |
| push: 1                                                                                                                                                                              |               |

## 7.10 Comparing values

Several compare instructions for comparing values on the stack exist. These instructions push a **false** (= 0) or **true** (= 1) on the stack.

**=?** [instruction]  
 Pops the top of the stack and compares this with ?. The ? can be 0 - 3. If the values are equal then the return value is 1 otherwise it is 0.

pop: 1  
 push: 1

**>?** [instruction]  
 Pops the top of the stack and compares this with ?. The ? can be 0 - 3. If stack value is larger than the ? then the return value is 1 otherwise it is 0.

pop: 1  
 push: 1

**>=?** [instruction]  
 Pops the top of the stack and compares this with ?. The ? can be 0 - 3. If stack value is larger or equal than the ? then the return value is 1 otherwise it is 0.

pop: 1  
 push: 1

**<?** [instruction]  
 Pops the top of the stack and compares this with ?. The ? can be 0 - 3. If stack value is smaller than the ? then the return value is 1 otherwise it is 0.

pop: 1  
 push: 1

**<=?** [instruction]  
 Pops the top of the stack and compares this with ?. The ? can be 0 - 3. If stack value is smaller or equal than the ? then the return value is 1 otherwise it is 0.

pop: 1  
 push: 1

**=** [instruction]  
 Pops the two top values of the stack and compares these. If the values are equal then the return value is 1 otherwise it is 0.

pop: 2  
 push: 1

**!=** [instruction]  
 Pops the two top values of the stack and compares these. If the values are not equal then the return value is 1 otherwise it is 0.

pop: 2  
 push: 1

- >** [instruction]  
Pops the two top values of the stack and compares these. If the first value is larger than the second value then the return value is 1 otherwise it is 0.  
pop: 2  
push: 1
- >=** [instruction]  
Pops the two top values of the stack and compares these. If the first value is larger or equal than the second value then the return value is 1 otherwise it is 0.  
pop: 2  
push: 1
- <** [instruction]  
Pops the two top values of the stack and compares these. If the first value is smaller than the second value then the return value is 1 otherwise it is 0.  
pop: 2  
push: 1
- <=** [instruction]  
Pops the two top values of the stack and compares these. If the first value is smaller or equal than the second value then the return value is 1 otherwise it is 0.  
pop: 2  
push: 1

## 7.11 Mathematical instructions

- +** [instruction]  
Pops two values from the top of the stack and adds these values.  
pop: 2  
push: 1
- [instruction]  
Pops two values from the top of the stack and subtracts the top value from the top - 1 value.  
pop: 2  
push: 1
- \*** [instruction]  
Pops two values from the top of the stack and multiplies these values.  
pop: 2  
push: 1
- /** [instruction]  
Pops two values from the top of the stack and divides these values.  
pop: 2  
push: 1

**mod** [instruction]  
Pops two values from the top of the stack and returns the modulo (rest value after division).  
pop: 2  
push: 1

**power** [instruction]  
Pops two values from the top of the stack and returns the power (first value to the power of second value).  
pop: 2  
push: 1

## 7.12 Logical bit instructions

**shift\_l** [instruction]  
Pops two values from the top of the stack and bit shifts first value left by number of bits of second value.  
pop: 2  
push: 1

**shift\_r** [instruction]  
Pops two values from the top of the stack and bit shifts first value right by number of bits of second value.  
pop: 2  
push: 1

**<<1** [instruction]  
Bit shifts value at the top of the stack one bit to the left.  
pop: 1  
push: 1

**>>1** [instruction]  
Bit shifts value at the top of the stack one bit to the right.  
pop: 1  
push: 1

**bit\_and** [instruction]  
Pops two values from the top of the stack and returns the bit AND of these values.  
pop: 2  
push: 1

**bit\_or** [instruction]  
Pops two values from the top of the stack and returns the bit OR of these values.  
pop: 2  
push: 1

**bit\_exor** [instruction]  
Pops two values from the top of the stack and returns the bit EXclusive OR of these values.

pop: 2  
push: 1

**~** [instruction]  
Performs the bit NOT on the value on the top of the stack.

pop: 1  
push: 1

### 7.13 Logical instructions

**and** [instruction]  
Pops two values from the top of the stack and returns the logical AND.

pop: 2  
push: 1

**or** [instruction]  
Pops two values from the top of the stack and returns the logical OR.

pop: 2  
push: 1

**!** [instruction]  
Pops the top value and returns the logical NOT.

pop: 1  
push: 1

### 7.14 String instructions

**cat** [instruction]  
Concatenates the string on top to the string on top - 1 of the stack, leaving a single string on the stack.

pop: 2  
push: 1

**cat-1** [instruction]  
Concatenates the string on top -1 to the string on top of the stack, leaving a single string on the stack. This instruction concatenates the strings reverse compared to **cat**.

pop: 2  
push: 1

**strcmp** [instruction]

Compares the two strings from the top of the stack. If the strings are equal a **true** is returned to the stack, otherwise a **false** is returned to the stack.

pop: 2

push: 1

**ifcmpset** [instruction]

A special compare instruction to implement if else macros.

It works on the 5 or 3 top positions of the stack. This depends on the number of arguments (on stack a) collected so far, not on the actual number of arguments on the stack.

If the number of arguments is 3 or a multiple of 3, then the 5 top positions are used. If the number of arguments is 1 or a multiple of 3 more (like 4, 7, 10, etc.), then the 3 top positions are used.

The stack should look in case of 5 top positions as:

```
top      : string used if compare is true
top - 1: second string to compare
top - 2: first string to compare
top - 3: flag of previous compares
top - 4: valid string if flag == 1
```

In the case of 5 top positions, the instruction will:

- check flag of previous compares.
- if flag of previous compares == 0 then compare first and second string.
- if strings are equal copy string at top to top - 4 and set flag to 1.
- pop top 3 positions.

The stack should look in case of 3 top positions as:

```
top      : string used if flag is false
top - 1: flag of previous compares
top - 2: valid string if flag == 1
```

In the case of 3 top positions, the instruction will:

- check flag of previous compares.
- if flag of previous compares == 0 then copy string at top to top - 2.
- pop top position.

Using this instruction in a pattern enables the implementation of an **ifelse** macro.

pop: depends on values on stack

push: depends on values on stack

**strlen** [instruction]

Pops the top and returns the length of the string.

pop: 1

push: 1

**str\*** [instruction]

Pops the two top positions, multiplies the string at top by a number at the top -1 position and returns the string.

pop: 2

push: 1

## 7.15 Output manipulation instructions

These instructions are used by the argument substitution pattern. They are used to change the output while processing the input. Since the input of argument collection is never directly output (only through the stack), these instructions are normally not useful during argument collection.

The current position in the output, where these instructions are executed, is the end of the pattern part.

The **end** instructions should not be used in argument substitution. Instead the **get** instructions should be used. These instructions do not end a **begin** instruction. To end a **begin** use the **endbegin**.

**get\_st** [instruction]

Get pattern part from the start position set by **begin** of the pattern part and stores the string on the stack.

pop: 0

push: 1

**get\_st-?** [instruction]

Get pattern part from the start position set by **begin** of the pattern part up to the end - offset ? and stores the string on the stack.

The offset (?) can be 1 to 3.

Similar to the **end-?**, but does not enable a new **begin**.

pop: 0

push: 1

**get\_st+?** [instruction]

Get pattern part from the start position set by **begin** of the pattern part + offset (?) and stores the string on the stack.

The offset (?) can be 1 to 3.

pop: 0

push: 1

**getlast?** [instruction]

Get pattern part from the end position of the pattern part - offset (?) and stores the string on the stack.

The offset (?) can be 1 to 5.

pop: 0

push: 1

**putoutst** [instruction]

Replace output from start position set by **begin** up to the current position by string on the stack.

This instruction will end a **begin** to allow a new begin.

pop: 1

push: 0

**putout** [instruction]

Replace pattern part by string on the stack. Can be used if the length of the pattern part is a fixed length.

pop: 1

push: 0

**putout-?** [instruction]

Replace output from rearward position (?) up to the current position by string on the stack.

The rearward position (?) can be 0 to 5.

pop: 1

push: 0

**endbegin** [instruction]

Ends a begin and thereby allows a new begin.

pop: 0

push: 0

## 7.16 Instructions for overruling macro settings

The following instructions are meant to set the recursion of output of the calling macro (the **mcall**) by the called macro.

**recur\_y** [instruction]

Overrules the setting of recursive macro output. The macro output is set to recursive.

pop: 0

push: 0

**recur\_n** [instruction]

Overrules the setting of recursive macro output. The macro output is set to not recursive.

pop: 0

push: 0

**nooverr** [instruction]

Turns the overrule of the setting of recursive macro output off.

pop: 0

push: 0

The following instructions are only useful to influence the recursion during the collection of arguments.

**no\_macro** [instruction]

When a program uses this instruction, no macros are called anymore during the collection of arguments.

Normally only useful for very specific cases in emulation. It is used in the **m6** emulation for specific handling of argument collection with macros that select the following text in a definition text.

Outside of the collection of arguments this instruction will have no influence.

pop: 0

push: 0

## 7.17 Operator stack instructions

The following instructions can be used for parsing and work on the instruction stack.

The **opex** instructions execute the instructions on the instruction stack or store an instruction on the instruction stack depending on a condition.

**opexif>** [instruction]

Operator execute if (**opexif**) will execute the instructions on the instruction stack as long as the rank gotten from the top of the stack is larger than the rank of the instruction on the instruction stack.

Will push the instruction following the **opexif** on the instruction stack with the rank gotten from the top of the stack.

pop: 1

push: 0

**opexif>=** [instruction]

Same as **opexif>** but will execute the instructions as long as the rank is larger or equal than the rank of the instruction on the instruction stack.

pop: 1

push: 0

**opexif<** [instruction]

Same as **opexif>** but will execute the instructions as long as the rank is smaller than the rank of the instruction on the instruction stack.

pop: 1

push: 0

**opexif<=** [instruction]  
 Same as **opexif>** but will execute the instructions as long as the rank is smaller or equal than the rank of the instruction on the instruction stack.  
 pop: 1  
 push: 0

**opexall** [instruction]  
 Will execute all the instructions on the instruction stack.  
 pop: 0  
 push: 0

**opexto** [instruction]  
 Will execute all instructions on the instruction stack until an instruction is found on the instruction stack that is the same as the instruction after the **opexto**. This last instruction on the stack will be executed, the instruction after the **opexto** not.  
 pop: 0  
 push: 0

**oppush** [instruction]  
 Pushes the the instruction after the **oppush** on the instruction stack with the rank gotten from the top of the stack.  
 pop: 1  
 push: 0

## 7.18 Getting macro information

With the following instructions it is possible to get all the information of macros.

Macros are placed in a list and are retrieved from this list by an internal number. The internal number used in these instructions is 1 or larger for a valid macro.

**m?\_num** [instruction]  
 Returns the internal number of a macro. The stack should hold:  
 top            macro set where the macro is defined.  
 top - 1        name of the macro.  
 If the macro does not exist, the returned number is 0.  
 pop: 2  
 push: 1

**m?\_name** [instruction]  
 Returns the name of a macro. The top of the stack should hold the internal number.  
 pop: 1  
 push: 1

|                                                                                                                                            |               |
|--------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| <b>m?_def</b>                                                                                                                              | [instruction] |
| Returns the definition of a macro. The top of the stack should hold the internal number.                                                   |               |
| pop: 1                                                                                                                                     |               |
| push: 1                                                                                                                                    |               |
| <b>m?_int</b>                                                                                                                              | [instruction] |
| Returns the builtin function of a macro or the macro name of an <code>mcall</code> . The top of the stack should hold the internal number. |               |
| pop: 1                                                                                                                                     |               |
| push: 1                                                                                                                                    |               |
| <b>m?_opt</b>                                                                                                                              | [instruction] |
| Returns the options of a macro. The top of the stack should hold the internal number.                                                      |               |
| pop: 1                                                                                                                                     |               |
| push: 1                                                                                                                                    |               |
| <b>m?_col_p</b>                                                                                                                            | [instruction] |
| Returns the name of the pattern for the collection of arguments. The top of the stack should hold the internal number.                     |               |
| pop: 1                                                                                                                                     |               |
| push: 1                                                                                                                                    |               |
| <b>m?_sub_p</b>                                                                                                                            | [instruction] |
| Returns the name of the pattern for the substitution of arguments. The top of the stack should hold the internal number.                   |               |
| pop: 1                                                                                                                                     |               |
| push: 1                                                                                                                                    |               |
| <b>m?_prog</b>                                                                                                                             | [instruction] |
| Returns the program of a macro. The top of the stack should hold the internal number.                                                      |               |
| pop: 1                                                                                                                                     |               |
| push: 1                                                                                                                                    |               |
| <b>m?_info</b>                                                                                                                             | [instruction] |
| Returns the information field of a macro. The top of the stack should hold the internal number.                                            |               |
| pop: 1                                                                                                                                     |               |
| push: 1                                                                                                                                    |               |
| <b>m?_type</b>                                                                                                                             | [instruction] |
| Returns the macro type: <code>macro</code> or <code>mcall</code> . The top of the stack should hold the internal number.                   |               |
| pop: 1                                                                                                                                     |               |
| push: 1                                                                                                                                    |               |

## 7.19 Counter variables

Just as general variables to hold strings exist, there are variables to hold numbers. These variables are called counters and are globally available to programs. This means that they hold their value between macros and programs.

The number of counters is maximised to 255 and the first counter starts with number 1.

These counters can be used to transfer information between programs in different macros.

**cnt\_get** [instruction]

This instruction will put the value of a counter on the active stack. The top of the stack is the number of the counter.

pop: 1

push: 1

**cnt\_set** [instruction]

This instruction will set the value of a counter to the value on the top of the active stack. The top-1 of the stack is the number of the counter.

pop: 2

push: 0

**cnt\_clr** [instruction]

This instruction will set the value of a counter to 0. The top of the stack is the number of the counter.

pop: 1

push: 0

**cnt++** [instruction]

This instruction will increment the value of a counter. The top of the stack is the number of the counter.

pop: 1

push: 0

**cnt--** [instruction]

This instruction will decrement the value of a counter. The top of the stack is the number of the counter.

pop: 1

push: 0

## 7.20 Calling a builtin function

With the following instruction it is possible to make a macro wherein the builtin executed is dependent on arguments of the function.

This can be useful in an emulation to adapt the macro to specialities of the emulated macro.

**fun\_call** [instruction]  
The function call will call the builtin function with the name found on the top of the stack.  
If the name does not exist, an error is output.  
The function will be immediately called, so the arguments on stack should already be correct. It is possible to call multiple builtin functions in one macro.  
pop: 1  
push: 0

## 7.21 Manipulation of stacks

Normally the stacks are created at the start of a macro and removed or made free at the end of a macro. The following instructions manipulate this behaviour.

These instructions make the exchange of information between macros possible. This can be useful in an emulation.

**free\_no** [instruction]  
This instruction sets the active stack to not be made free or removed at the end of the macro.  
The stack of a macro can get additional entries from macros that are later called if these later called macros use this instruction. This means that macros called during the argument collection can push entries on a stack of the currently executing macro.  
pop: 0  
push: 0

**free\_yes** [instruction]  
This instruction sets the active stack to the default of being freed or removed at the end of the macro.  
pop: 0  
push: 0



## 8 Emulation examples

### 8.1 Introduction

The emulation examples are examples of defining a macro language in `m0`. It shows the different possibilities and solutions possible with `m0`.

### 8.2 M4 emulation example

#### 8.2.1 Introduction

The `m4` emulation is an example of defining a macro language in `m0`.

It is (at least now) not a goal to be 100% compatible with (GNU) `m4` or to be complete.

Since `m0` will also detect macros with invalid characters of `m4` macros, it will probably be very difficult to be 100% compatible for cases where certain characteristics of `m4` are used.

Also functions for regular expressions in `m0` seem not necessary as this functionality can also be accomplished with patterns. The macros related to regular expressions in (GNU) `m4` are therefore not implemented.

#### 8.2.2 Emulation

The following text of the emulation is generated from a source file for both text and code. From this source file different versions of `m4` are generated. These versions are:

|                    |                                                                                                                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>v7</code>    | The original version of <code>m4</code> , see <i>The M4 Macro Processor</i> , Brian W. Kernighan and Dennis M. Ritchie, Bell Laboratories, July 1, 1977.                                                                                                                         |
| <code>posix</code> | A version with only the macros defined by <code>posix</code> , see e.g. <code>m4</code> ( <a href="https://pubs.opengroup.org/onlinepubs/9699919799/utilities/m4.html">https://pubs.opengroup.org/onlinepubs/9699919799/utilities/m4.html</a> ).                                 |
| <code>gnu</code>   | Version with the macros defined in GNU <code>m4</code> , see GNU <code>M4</code> manual ( <a href="https://www.gnu.org/software/m4/manual/m4.html">https://www.gnu.org/software/m4/manual/m4.html</a> ).                                                                         |
| <code>bsd</code>   | Version with the macros defined in <code>m4</code> of BSD, see e.g. FreeBSD <code>m4</code> manual page ( <a href="https://man.freebsd.org/cgi/man.cgi?query=m4&amp;sektion=1&amp;format=html">https://man.freebsd.org/cgi/man.cgi?query=m4&amp;sektion=1&amp;format=html</a> ). |

Macros or code for all versions is indicated with `all`.

#### 8.2.3 Main design

The emulation of `m4` is using four macro sets:

1. The default macro set 0 is used to define all basic macros that are necessary to be able to define the `m4` macros. It is also used to hold some `m4` macros that are build using the basic macros. The emulation starts in this macro set and is switched to the `m4exec` macro set to start emulation of the `m4` input.
2. The `m4` macros are defined and executed in the macro set `m4exec`.
3. The `m4builtin` macro set is used as a collection of builtin `m4` macros that can be called from the `builtin` macro.
4. The `Args` macro set is used for the `m4` command line options.

A big number of the builtin **m4** macros are directly linked to the functions in **m0**. Several patterns are used to implement the syntax of **m4**. Difficult is a new definition of builtin macros using the **defn** macro. The use of **defn** requires that its use is detected by a define and the define is a copy or a define depending on this.

Some builtin **m4** macros are emulated using a collection of macros in the default macro set 0.

Quoting and comments are implemented as macros as there exists no built in functionality for this in **m0**.

Although it is not necessary to use three macro sets, it allows a relatively easy separation of the underlying macro code and the emulated macros. Another solution can be a proper naming of underlying macros such that these are unlikely accidentally called.

### 8.2.4 At the start

Applicable for: all

At the start the macros used to define the macros for **m4** are defined. This is all straightforward linking of macro names to builtin functions.

```
0_define:0_pattern;;;pattern;nr01;0
0_define:0_append_pattern;;;append_p;nr01;0
0_define:0_undefine;;;undefine;nr01;0
0_define:0_clr_pattern;;;clr_pat;nr01;0
0_define:0_copy_pattern;;;copy_pat;nr01;0
0_define:0_set_var;;;set_var;nr01;0
0_define:0_get_var;;;get_var;nr01;0
0_define:0_ifmacro;;;ifmacro?;rr01;0
0_define:0_argoptions;;;get_var;rr01;0;; stack_a pop_to_1 -7
0_define:0_chars_pattern;;;charpat;nr01;0
0_define:0_chars_args;;;chararg;nr01;0
0_define:0_specialchar;;;specialc;nr01;0
0_define:0_def_mcall;;;defmcall;nr01;0
0_define:0_atlast;;;at_last;nr01;0
0_define:0_atfirst;;;at_first;nr01;0
0_define:0_mset;;;macroset;nr01;0
0_define:0_info;;;info;nr01;0
0_define:0_infofield;$$9;info;nr01;0
0_define:0_setinfo;;;set_info;nr01;0
0_define:0_copy;;;copy;nr01;0
0_define:0_char;;;num2chr;nr01;0
0_define:0_program;;;program;nr01;0
0_define:0_divert;;;divert;nr01;0
0_define:0_undivert;;;undivert;nr01;0
0_define:0_divnum;;;get_var;nr01;0;; stack_a pop_to_1 -1
0_define:0_inter_pat;;;inter_p;nr01;0
0_define:0_tag_pat;;;tag_p;nr01;0
0_define:0_link_pat;;;link_prt;nr01;0
```

Setting the defaults. This is also the default of `m0`.

```
0_chars_pattern: []-\@+*?~
0_define:0_l1;0_char:1
;;nn00;;;0
0_define:0_r1;0_char:2
;;nn00;;;0
0_define:0_alt;0_char:3
;;nn00;;;0
0_define:0_slash;0_char:4
;;nn00;;;0
```

### 8.2.5 Large argument numbers

Applicable for: `gnu bsd`

Numbers can be larger than 0 – 9 (up to 9999) to select the arguments.

The second line defines the special characters for the arguments. This is missing the fifth character. It is thus not possible to select arguments from other stacks than the first stack.

```
0_chars_args:4$*@#;101;102;103;m4exec
0_chars_args:4$*@#;101;102;103;m4builtin
```

### 8.2.6 Argument with single number

Applicable for: `v7 posix`

Numbers can only be 0 – 9 to select the arguments.

```
0_chars_args:1$*@#;101;102;103;m4exec
0_chars_args:1$*@#;101;102;103;m4builtin
```

```
0_chars_args:1$*@$;111;112;103
```

### 8.2.7 Quoting in default macro set

Applicable for: `all`

In the next part a macro with a pattern is used to have quoting in the macros in the default macro set 0 operational. The quoting is thus not a builtin function in `m0`.

```
0_set_var:111;+--+
0_set_var:112;--+-
0_pattern:0string;[@00-@ff]~;stack_b 1 stack_a begin-1
0_pattern:0string;+--+;stack_b 1 +
0_pattern:0string;--+-;stack_b 1 - dup =0 if stack_a end stop endif
0_define:++-+;$1;;nn00 ;0string
```

And defining a beginning pattern to be copied.

```
0_pattern:0start;[@00-@ff]~;stack_b 1 stack_a begin-1
```

### 8.2.8 Special characters

Applicable for: all

First some special characters are defined to make the definition of macros easier.

The "\ " is used at the end of a macro to mean all characters except characters valid for a name. This ends the detection of a macro name. It is used for all macros which have optional arguments. Macros which must be followed by arguments end with a "(".

The "\B" is used at the start of a macro to mean all characters except characters valid for a name. This is used to not start a macro inside a word.

```
0_specialchar: ;[@00-@2f@3a-@40@5b-@5e@60@7b-@ff]
0_specialchar:B;[@00-@40@5b-@5e@60@7b-@ff]
```

### 8.2.9 Command line options

Applicable for: all

The collecting of command line options is executed in a separate macro-set **Args**.

```
0_define:==Options;;;get_var;rn00;;; stack_a pop_to_1 -8;Args
0_define:==Alloptions;;;get_var;rn00;;; stack_a pop_to_1 -7;Args
0_define:==Mset;;;macroset;nr01;0;;;Args
0_pattern:args;[@00-@ff]~;stack_a begin-1
0_pattern:args;=;stack_a end begin
0_pattern:args;+++
---;stack_a end stop
```

Setting the option of -P / --prefix-builtins to prefix all builtins with m4\_.

This is accomplished by deleting the macro m4\_. This macro is set as default to delete the m4\_ part of the macro names. Deleting this macro will leave the m4\_ part.

```
0_define:-P;;;undefine;nn00;;;stack_a "m4_" "0";Args
0_define:--prefix-builtins;;;undefine;nn00;;;stack_a "m4_" "0";Args
0_define:m4_;;;nn00
0_divert:-1
0_mset:Args
==Alloptions
==Mset:0
0_divert:0
```

The other command line options are executed at the start of a new file.

```
0_def_mcall:-D;define;m4builtin;nn00;args;;Args
0_def_mcall:--define=;define;m4builtin;nn00;args;;Args
0_def_mcall:-U;undefine;m4builtin;nn00;args;;Args
0_def_mcall:--undefine=;undefine;m4builtin;nn00;args;;Args
```

Setting options at the start of a file by an mcall to the ==Options macro in **Args**.

During this time diverting the output so that no output of options remain.

```
0_atfirst:m4_define(4_prev_div, divnum)m4_divert(-1)+++
```

```

4_Set_Options
---m4_divert(4_prev_div)m4_dnl++-+
---+
0_def_mcall:4_Set_Options;==Options;Args;+--+nn00
---+;;m4exec

```

### 8.2.10 Collecting arguments of macros

Applicable for: all

The collecting of arguments in `m4` macros is executed with a pattern named `m4` containing small programs.

```

0_specialchar:([@00-@27@29-@ff]
0_pattern:m4;[\(~;stack_b argpos =0 ifthen abort
0_pattern:m4;[@00-@ff]~;stack_c 0
0_pattern:m4;( ;stack_c 1 +
0_pattern:m4;[(,)[@00- ]*[@!-@ff];stack_a begin-1 ;1
0_pattern:m4;);stack_c 1 - dup =0 if stack_a end stop endif
0_pattern:m4;;stack_c dup <=1 if stack_a end endif

```

### 8.2.11 define

Applicable for: all

The `define` of `m4` can not be directly linked to the function `define`. In `m4` a builtin macro can be named by using `defn`. These builtin macros should be handled differently from a user macro. Some code is necessary to handle the two cases.

Without the builtin definition, the `define` could be directly linked as:

```

0_define:\Bdefine(;;define;nr11 ;m4;;+--+stack_d "" putarg3
"rr11" putarg4 "m4" putarg5 "" putarg6 "" putarg7 "m4exec"
putarg8 "\B" getarg1 cat "\ " cat putarg1---+;;m4exec

```

In this example a program, executed in the macro, sets default values for the `define` function.

In this emulation the `defn` returns the information of the builtin macro on stack `h`. When a builtin is returned by the `defn` stack `h` holds:

- empty or true (1)
- name of builtin macro
- info of builtin macro

A macro to check for this and select the correct option is defined to handle this. The macro is actually a large program that:

- Checks if there are no arguments or if the macro name is empty. If this is the case do not define a macro and stop.
- Copies the arguments to separate stacks for later use, stack `e` holds argument 1, stack `f` holds argument 2.

- Checks if stack h holds a true (1) for a builtin macro,
- for builtin macro does 2x a copy, one to copy the builtin and one to copy the macro in the m4exec macro set,
- or defines a new macro if a user macro is defined.

```

0_program:define;++-+
stack_e argnum =0 ifthen stop
getarg1 dup "" strcmp ifthen stop
stack_f getarg2
stack_h if
stack_a pop_to_2 copyfr_h swap "m4builtin" "m4builtin" "copy" fun_call
pop_to_1 "\B" copyfr_h cat stack_h pop stack_a copyfr_h cat "\B" copyfr_e cat
copyfr_h cat "m4exec" "m4exec" "copy" fun_call
else
stack_a pop_to_1 "\B" copyfr_e cat "\ " cat
copyfr_f "" "rr11" "m4" "" "" "m4exec" "" "define" fun_call
endif ---+

```

The program uses function calls, the macro is further empty because of this.

The `m4builtin` set is used to hold the information of the new builtin macro. This has as a side effect that this new macro can also be called using the `m4 builtin`, which is not possible in `m4`.

```

0_define:\Bm4_define(;;;nr11 ;m4;; @define ;m4exec
0_define:define;;;nn00;m4;; @define ;m4builtin
0_setinfo:define(;m4builtin

```

### 8.2.12 pushdef

Applicable for: `posix gnu bsd`

Similar to the `define`. The only difference is the use of `push` instead of `define`.

```

0_program:push;++-+
stack_e argnum =0 ifthen stop
getarg1 dup "" strcmp ifthen stop
stack_f getarg2
stack_h if
stack_a pop_to_2 copyfr_h swap "m4builtin" "m4builtin" "pushcopy" fun_call
pop_to_1 "\B" copyfr_h cat stack_h pop stack_a copyfr_h cat "\B" copyfr_e cat
copyfr_h cat "m4exec" "m4exec" "pushcopy" fun_call
else
stack_a pop_to_1 "\B" copyfr_e cat "\ " cat
copyfr_f "" "rr11" "m4" "" "" "m4exec" "" "push" fun_call
endif ---+
0_define:\Bm4_pushdef(;;;nr11 ;m4;; @push ;m4exec
0_define:pushdef;;;nn00;m4;; @push ;m4builtin
0_setinfo:pushdef(;m4builtin

```

### 8.2.13 defn

Applicable for: `posix gnu bsd`

`defn` should return the definition of a macro or in case of a builtin something so that in a `define` a new macro can be defined for the builtin.

These two possibilities require a different handling and thus a specific argument collection is defined that is similar to the `m4` argument collection, but puts the arguments to stack `b` and calls a program to handle a builtin or a normal macro.

A builtin is returned on stack `h` to be used by a `define`.

Stack `h` is not freed and holds:

- empty or true (1) if a builtin was found
- name of builtin macro
- information text of builtin macro

The information text of the macros in the `m4builtin` macro set is used to determine if a macro has optional arguments or not.

```
0_program:m4defnif;++-- stack_b end dup "m4builtin" m?_num dup if
copyto_h stack_h m?_info copyfr_b m?_name 1 free_no else
stack_b pop "\B" swap cat "\" cat "m4exec" m?_num
dup if
stack_a copyfr_b m?_def cat endif
endif --+-
0_pattern:m4defn;[\(\)~;stack_b argpos =0 ifthen abort
0_pattern:m4defn;[@00-@ff]~;stack_c 0 stack_a ""
0_pattern:m4defn;(;stack_c 1 +
0_pattern:m4defn;[(,)[@00- ]*[@!-@ff];stack_b begin-1 ;1
0_pattern:m4defn;);stack_c 1 - dup =0 if @m4defnif stop endif
0_pattern:m4defn;;stack_c dup <=1 if @m4defnif endif
0_define:\Bm4_defn($1;;nr11 ;m4defn;;;m4exec
0_define:defn;$1;;nn00;m4defn;;;m4builtin
0_setinfo:defn;(;m4builtin
```

### 8.2.14 popdef

Applicable for: `posix gnu bsd`

First a program is defined that selects between the two possible macros. It also sets the macro set.

```
0_program:undef;++-- stack_d "\B" getarg1 cat
dup "\" cat "m4exec" m?_num if
"\ " cat putarg1 else
("(" cat putarg1 endif
"m4exec" putarg2 --+-
```

The `popdef` is a link to the `pop` function with the correct input set by the program.

```
0_define:\Bm4_popdef(;;pop;nr11 ;m4;; @undef ;m4exec
```

```
0_define:popdef;;pop;nn00;m4;; @undef ;m4builtin
0_setinfo:popdef;(;m4builtin
```

### 8.2.15 undefine

Applicable for: all

Similar to popdef, but with an undefine function.

```
0_define:\Bm4_undefine(;;undefine;nr11 ;m4;; @undef ;m4exec
0_define:undefine;;undefine;nn00;m4;; @undef ;m4builtin
0_setinfo:undefine;(;m4builtin
```

### 8.2.16 Quoting in m4

Applicable for: all

In the next part a macro with a pattern is used to have quoting operational in the m4 emulation.

```
0_set_var:101;`
0_set_var:102;'
0_set_var:103;,
0_set_var:100;$1
0_pattern:string;[@00-@ff]~;stack_b 1 stack_a begin-1
0_pattern:string;`;stack_b 1 +
0_pattern:string;' ;stack_b 1 - dup =0 if stack_a end stop endif
0_define:0_get_var:101
;$1;;nn00 ;string;;m4exec
```

### 8.2.17 changequote

Applicable for: all

The changequote calls the macro 0\_changequote in macro set 0.

This enables the use of macros in the default macro set to perform the change of quotes.

```
0_def_mcall:\Bm4_changequote\ ;0_changequote;0;nr11 ;m4;;m4exec
0_def_mcall:changequote;0_changequote;0;nn00;m4;;m4builtin
0_setinfo:changequote;\ ;m4builtin
0_define:0_changequote;+--+0_undefine:0_get_var:101
;m4exec
0_set_var:101;+--+ $1--+
0_set_var:102;+--+ $2--+
0_copy_pattern:0start;string
0_pattern:string;+--+ $1--+;stack_b 1 +
0_pattern:string;+--+ $2--+;stack_b 1 - dup =0 if stack_a end stop endif
0_define:0_get_var:101
;0_get_var:100
;;nn00 ;string;;m4exec
+--+;rn00;0;;+--+stack_d argnum <=1 if "`" putarg1 "'" putarg2 endif+--+
```

### 8.2.18 Comments

Applicable for: posix gnu bsd

Also a comment is built as a macro.

```
0_set_var:104;#
0_pattern:comment;[@00-@ff]~;begin-1
0_pattern:comment;+--+
+--+;end stop
0_define:#+--++$0$1
+--+;;nn00 ;comment;;stack_a argnum =0 ifthen end-1;m4exec
```

### 8.2.19 changecom

Applicable for: posix gnu bsd

Similar to `changequote` but with the specific argument amounts of `changepcom`.

It uses the `__ifelse` macro defined further down like an `m4 ifelse` in the `0` macro set.

```
0_def_mcall:\Bm4_changepcom\ ;0_changepcom;0;nr11 ;m4;;m4exec
0_def_mcall:changepcom;0_changepcom;0;nr00;m4;;m4builtin
0_setinfo:changepcom;\ ;m4builtin
0_define:0_changepcom;+--+__ifelse(0_get_var:104
,,+--+0_undefine:0_get_var:104
;m4exec
+--+0_set_var:104;+--++$1+--+
0_copy_pattern:0start;comment
0_pattern:comment;+--++$2+--+;end stop
__ifelse(+--++$1+--, , ,+--+0_define:+--++$1+--+;+--++$--+0$+--+1+--++++$2+--+;;nn00 ;comment
+--+)+--+;;rn00;0;;+--+stack_d argnum <=1 if "
" putarg2 endif+--+
```

### 8.2.20 builtin

Applicable for: gnu bsd

Defines a macro call to a macro stored in the first argument. The macro set `m4builtin` is used for calling the builtin macros of `m4`.

The macros defined in the macro set `m4builtin` are only called from this macro. Therefore these macros should not recurse their output, because this is done by the `builtin` macro. Since they are never called normally, the settings for argument collection are irrelevant.

For macros that have a different argument collection than the `m4` of `builtin` this represents a difficulty.

The builtin macro itself is also defined in the macro set `m4builtin`.

```
0_def_mcall:\Bm4_builtin(;m4builtin;rr11 ;m4;;m4exec
0_def_mcall:builtin;m4builtin;nn00;m4;;m4builtin
```

```
0_setinfo:builtin;(;m4builtin
```

### 8.2.21 include

Applicable for: all

```
0_define:\Bm4_include(;;include;rr11 ;m4;;;m4exec
0_define:include;;include;nn00;m4;;;m4builtin
0_setinfo:include;(;m4builtin
```

### 8.2.22 sinclude

Applicable for: all

```
0_define:\Bm4_sinclude(;;sinclude;rr11 ;m4;;;m4exec
0_define:sinclude;;sinclude;nn00;m4;;;m4builtin
0_setinfo:sinclude;(;m4builtin
```

### 8.2.23 place

Applicable for: `bsd`

The `place` is like an `include`, but the contents of the file is not recursively scanned for macros.

To have the `builtin` not to recursively scan the output the setting thereof is overruled with the `recur_n`.

```
0_define:\Bm4_place(;;include;nr11 ;m4;;;m4exec
0_define:place;;include;nn00;m4;;recur_n;m4builtin
0_setinfo:place;(;m4builtin
```

### 8.2.24 splace

Applicable for: `bsd`

Similar to `place` just silent for file errors.

```
0_define:\Bm4_splace(;;sinclude;nr11 ;m4;;;m4exec
0_define:splace;;sinclude;nn00;m4;;recur_n;m4builtin
0_setinfo:splace;(;m4builtin
```

### 8.2.25 divert

Applicable for: all

```
0_define:\Bm4_divert\ ;;divert;nr11 ;m4;;;m4exec
0_define:divert;;divert;nn00;m4;;;m4builtin
0_setinfo:divert;\ ;m4builtin
```

### 8.2.26 undivert

Applicable for: all

```

0_define:\Bm4_undivert\ ;;undivert;nr11 ;m4;;;m4exec
0_define:undivert;;undivert;nn00;m4;;;m4builtin
0_setinfo:undivert;\ ;m4builtin

```

### 8.2.27 divnum

Applicable for: all

```

0_define:\Bm4_divnum\ ;;get_var;nn11 ;m4;;stack_a pop_to_1 -1;m4exec
0_define:divnum;;get_var;nn00;m4;;stack_a pop_to_1 -1;m4builtin
0_setinfo:divnum;\ ;m4builtin

```

### 8.2.28 \_\_line\_\_

Applicable for: gnu bsd

```

0_define:\Bm4___line__\ ;;get_var;nn11 ;m4;;stack_a pop_to_1 -4;m4exec
0_define:__line__;;get_var;nn00;m4;;stack_a pop_to_1 -4;m4builtin
0_setinfo:__line__;\ ;m4builtin

```

### 8.2.29 \_\_file\_\_

Applicable for: gnu bsd

```

0_define:\Bm4___file__\ ;;get_var;nn11 ;m4;;stack_a pop_to_1 -5;m4exec
0_define:__file__;;get_var;nn00;m4;;stack_a pop_to_1 -5;m4builtin
0_setinfo:__file__;\ ;m4builtin

```

### 8.2.30 \_\_program\_\_

Applicable for: gnu

```

0_define:\Bm4___program__\ ;;get_var;nn11 ;m4;;stack_a pop_to_1 -6;m4exec
0_define:__program__;;get_var;nn00;m4;;stack_a pop_to_1 -6;m4builtin
0_setinfo:__program__;\ ;m4builtin

```

### 8.2.31 \_\_unix\_\_

Applicable for: gnu

```

0_define:\Bm4___0_get_var:-2
__\ ;;nn11;m4;;;m4exec
0_define:__0_get_var:-2
__;;nn00;m4;;;m4builtin
0_setinfo:__0_get_var:-2
__;\ ;m4builtin

```

**8.2.32 unix**

Applicable for: v7 bsd

```
0_define:\Bm4_0_get_var:-2
\ ;0_get_var:-2
;;nn11;m4;;;m4exec
0_define:0_get_var:-2
;0_get_var:-2
;;nn00;m4;;;m4builtin
0_setinfo:0_get_var:-2
;\ ;m4builtin
```

**8.2.33 errprint**

Applicable for: all

```
0_define:\Bm4_errprint(;;errprint;nr11 ;m4;;;m4exec
0_define:errprint;;errprint;nn00;m4;;;m4builtin
0_setinfo:errprint;(;m4builtin
```

**8.2.34 m4exit**

Applicable for: posix gnu bsd

```
0_define:\Bm4_m4exit(;;exit;nr11;m4;;;m4exec
0_define:m4exit;;exit;nn00;m4;;;m4builtin
0_setinfo:m4exit;(;m4builtin
```

**8.2.35 index**

Applicable for: all

```
0_define:\Bm4_index(;;strindex;rr11 ;m4;;;m4exec
0_define:index;;strindex;nn00;m4;;;m4builtin
0_setinfo:index;(;m4builtin
```

**8.2.36 substr**

Applicable for: all

```
0_define:\Bm4_substr(;;substr;rr11 ;m4;;;m4exec
0_define:substr;;substr;nn00;m4;;;m4builtin
0_setinfo:substr;(;m4builtin
```

**8.2.37 translit**

Applicable for: all

```
0_define:\Bm4_translit(;;strtrans;rr11 ;m4;;;m4exec
0_define:translit;;strtrans;nn00;m4;;;m4builtin
```

```
0_setinfo:translit;(;m4builtin
```

### 8.2.38 m4wrap

Applicable for: posix gnu bsd

```
0_define:\Bm4_m4wrap(;;at_last;nr11 ;m4;;;m4exec
0_define:m4wrap;(;at_last;nn00;m4;;;m4builtin
0_setinfo:m4wrap;(;m4builtin
```

### 8.2.39 esyscmd

Applicable for: gnu bsd

The output of the `shell` function is recursive. This is different from the `m4` macro `syscmd`. See the `rr11` settings string.

```
0_define:\Bm4_esyscmd(;;shell;rr11 ;m4;;;m4exec
0_define:esyscmd;(;shell;nn00;m4;;;m4builtin
0_setinfo:esyscmd;(;m4builtin
```

### 8.2.40 syscmd

Applicable for: all

The output of `syscmd` is not recursive. See the `n` in the `nr11` settings string.

```
0_define:\Bm4_syscmd(;;shell;nr11 ;m4;;;m4exec
0_define:syscmd;(;shell;nn00;m4;;;recur_n;m4builtin
0_setinfo:syscmd;(;m4builtin
```

### 8.2.41 sysval

Applicable for: all

```
0_define:\Bm4_sysval\ ;;get_var;nn11 ;m4;;;stack_a pop_to_1 -9;m4exec
0_define:sysval;(;get_var;nn00;m4;;;stack_a pop_to_1 -9 recur_n;m4builtin
0_setinfo:sysval;\ ;m4builtin
```

### 8.2.42 maketemp

Applicable for: posix gnu bsd

Making the "safe" temporary file instead of the "unsafe" file in the original `m4`.

```
0_define:\Bm4_maketemp(;;tempfile;rr11 ;m4;;;m4exec
0_define:maketemp;(;tempfile;nn00;m4;;;m4builtin
0_setinfo:maketemp;(;m4builtin
```

### 8.2.43 mkstemp

Applicable for: posix gnu bsd

Same as `maketemp`, but the output is not recursive.

```
0_define:\Bm4_mkstemp(;;tempfile;nr11 ;m4;;m4exec
0_define:mkstemp;;tempfile;nn00;m4;;recur_n;m4builtin
0_setinfo:mkstemp;(;m4builtin
```

### 8.2.44 `dnl`

Applicable for: `all`

The delete to newline as the `m4` macro and a special `0_dnl` (macro starts with a space!) used at the beginning and end of the input.

The `dnl` uses a similar pattern as comments to read a single argument up to a newline.

```
0_pattern:m4dnl;[@00-@ff]~;begin-1
0_pattern:m4dnl;+++
--+-;end stop
0_define:\Bm4_dnl\ ;;;++-+nn11
--+-;m4dnl;;m4exec
0_define:dnl;;++-+nn00
--+-;m4dnl;;m4builtin
0_setinfo:dnl;\ ;m4builtin
0_define: 0_dnl;;++-+nn00
--+-;m4dnl;;m4exec
```

### 8.2.45 `incr`

Applicable for: `all`

```
0_define:\Bm4_incr(;$1;;rr11 ;m4;;stack_d getarg1 1 + putarg1;m4exec
0_define:incr;$1;;nn00;m4;;stack_d getarg1 1 + putarg1;m4builtin
0_setinfo:incr;(;m4builtin
```

### 8.2.46 `decr`

Applicable for: `posix gnu bsd`

```
0_define:\Bm4_decr(;$1;;rr11 ;m4;;stack_d getarg1 1 - putarg1;m4exec
0_define:decr;$1;;nn00;m4;;stack_d getarg1 1 - putarg1;m4builtin
0_setinfo:decr;(;m4builtin
```

### 8.2.47 `shift`

Applicable for: `posix gnu bsd`

```
0_define:\Bm4_shift(;$@;;rr11 ;m4;;base=_1;m4exec
0_define:shift;$@;;nn00;m4;;base=_1;m4builtin
0_setinfo:shift;(;m4builtin
```

### 8.2.48 indir

Applicable for: gnu bsd

This will just put the macro called by `indir` in the output and recurse this macro.

```
0_define:\Bm4_indir(;$1($@);;rr11 ;m4;;base=_1;m4exec
0_define:indir;$1($@);;nn00;m4;;base=_1;m4builtin
0_setinfo:indir;(;m4builtin
```

### 8.2.49 ifdef

Applicable for: all

The `ifdef` calls the macro `0_ifdef` in macro set 0.

This enables the use of macros in the default macro set.

```
0_def_mcall:\Bm4_ifdef(;0_ifdef;0;rr11 ;m4;;m4exec
0_def_mcall:ifdef;0_ifdef;0;nn00;m4;;m4builtin
0_setinfo:ifdef;(;m4builtin
0_define:0_ifdef;+++0_ifmacro:\B+++$1---\ ;+++$2---;+++0_ifmacro:\B+++$1---(;+++$2
---;m4exec
---;rr00;0
```

### 8.2.50 len

Applicable for: all

```
0_define:\Bm4_len(;$1;;nr11 ;m4;;stack_a pop_to_2 argnum >0 if strlen else 0 endif;m4exec
0_define:len;$1;;nn00;m4;;stack_a pop_to_2 argnum >0 if strlen else 0 endif;m4builtin
0_setinfo:len;(;m4builtin
```

### 8.2.51 ifelse

Applicable for: all

The `ifelse` macro is implemented with a pattern using the special `ifcmpset` instruction in the programs.

The `ifelse` macro does not use the `m4` pattern and therefore the macro is cloned in the 0 macro set for use with the builtin macro. Using the intermediary macro `0_ifelse` to output the arguments to the clone macro `4_ifelse` so that the `m4if` pattern can be used.

```
0_pattern:m4if;[\(\)~;stack_b argpos =0 ifthen abort
0_pattern:m4if;[@00-@ff]~;stack_c 0 stack_a "" 0
0_pattern:m4if;(;stack_c 1 +
0_pattern:m4if;);stack_c 1 - dup =0 if stack_a end ifcmpset stop endif
0_pattern:m4if;[(,)[@00- ]*[@!-@ff];stack_a begin-1 ;1
0_pattern:m4if;;stack_c dup <=1 if stack_a end stack_d argnum 3 mod =0 if stack_a ifcmpset
0_define:\Bm4_ifelse(;$1;;rr11 ;m4if;; stack_a argnum =1 ifthen pop argnum =2 ifthen pop po
0_def_mcall:ifelse;0_ifelse;0;nn00;m4;;m4builtin
0_setinfo:ifelse;(;m4builtin
0_define:0_ifelse;4_ifelse($@);;rn00;m4;;;0
```

```
0_define:4_ifelse(;$1;;nr01;m4if;; stack_a argnum =1 ifthen pop argnum =2 ifthen pop pop;0
0_define:__ifelse(;$1;;rr01;m4if;; stack_a argnum =1 ifthen pop argnum =2 ifthen pop pop;0
```

### 8.2.52 eval

Applicable for: all

The arguments are parsed and evaluated during the argument collection. Therefore a specific pattern for the argument collection is defined in the same way as for the `ifelse` macro.

Like in the builtin `ifelse` also here an intermediary macro is used for the builtin macro.

The `4_eval` macro can not be called with the arguments quoted, because the quotes used (+++ and ---) interfere with the evaluation by the `m4eval` pattern.

```
0_pattern:m4eval;[\(\)~;stack_b argpos =0 ifthen abort
0_pattern:m4eval;[@00-@ff]~;stack_c 0 stack_d 1
0_specialchar:S;[@00-@2a@2b@2c@2d@2e@2f@3a-@41@43-@51@53-@57@59-@61@63-@71@73-@77@79-@ff]
0_specialchar:*;[@00-@29@2b-@ff]
0_specialchar:>;[@00-@3c@3f-@ff]
0_specialchar:<;[@00-@3b@3e-@ff]
0_specialchar:=;[@00-@20@22-@3b@3f-@ff]
0_specialchar:&;[@00-@25@27-@ff]
0_specialchar:|;[@00-@7b@7d-@ff]
0_specialchar:D;[@00-@2f@3a-@ff]
0_specialchar:R;[@00-@2f@3b-@40@5b-@60@7b-@ff]
0_pattern:m4eval;\S[0-9];stack_b begin+1
0_append_pattern:m4eval;[0-9a-zA-Z:]*\R;stack_b end-1 stack_d pop_to_0 0
0_pattern:m4eval;+[0-9];stack_d =0 if stack_b 19 opexif>= + else stack_b 0 30 opexif>= + en
0_pattern:m4eval;-[0-9];stack_d =0 if stack_b 19 opexif>= - else stack_b 0 30 opexif>= - en
0_pattern:m4eval;+\\D;stack_b 19 opexif>= + stack_d 1
0_pattern:m4eval;-\\D;stack_b 19 opexif>= - stack_d 1
0_pattern:m4eval;\\*\\*;stack_b 20 opexif>= * stack_d 1
0_pattern:m4eval;\\*\\*;stack_b 21 opexif> power stack_d 1
0_pattern:m4eval;/;stack_b 20 opexif>= / stack_d 1
0_pattern:m4eval;%;stack_b 20 opexif>= mod stack_d 1
0_pattern:m4eval;~;stack_b 29 opexif>= ~ stack_d 1
0_pattern:m4eval;!\\=;stack_b 29 opexif>= ! stack_d 1
0_pattern:m4eval;<<;stack_b 18 opexif>= shift_l stack_d 1
0_pattern:m4eval;>>;stack_b 18 opexif>= shift_r stack_d 1
0_pattern:m4eval;\\>>\\>;stack_b 17 opexif>= > stack_d 1
0_pattern:m4eval;\\<<\\<;stack_b 17 opexif>= < stack_d 1
0_pattern:m4eval;>=;stack_b 17 opexif>= >= stack_d 1
0_pattern:m4eval;<=;stack_b 17 opexif>= <= stack_d 1
0_pattern:m4eval;==;stack_b 16 opexif>= = stack_d 1
0_pattern:m4eval;!\\=;stack_b 16 opexif>= != stack_d 1
0_pattern:m4eval;this part is doing nothing, it is here to test the memory use of pattern;
0_pattern:m4eval;\\==\\=;stack_b 16 opexif>= = stack_d 1
0_pattern:m4eval;\\&&\\&;stack_b 15 opexif>= bit_and stack_d 1
```

```

0_pattern:m4eval;^;stack_b 14 opexif>= bit_exor stack_d 1
0_pattern:m4eval;\||\||;stack_b 13 opexif>= bit_or stack_d 1
0_pattern:m4eval;&&;stack_b 12 opexif>= and stack_d 1
0_pattern:m4eval;||;stack_b 11 opexif>= or stack_d 1
0_pattern:m4eval;( ;stack_c 1 + dup >=2 if stack_b 0 oppush nop endif
0_pattern:m4eval;);stack_c 1 - dup >=1 if stack_b opexto nop endif stack_c dup =0 if stack_
0_pattern:m4eval;[(,)[@00- ]*[@!-@ff];stack_a begin-1 ;1
0_pattern:m4eval;;stack_c dup <=1 if stack_a end argnum =1 if stack_b opexall putarg1 endi
0_define:\Bm4_eval(;;num2str;rr11 ;m4eval;;m4exec
0_def_mcall:eval;0_eval;0;nn00;m4;;m4builtin
0_setinfo:eval;( ;m4builtin
0_define:0_eval;4_eval($*);;rn00;m4;;;0
0_define:4_eval(;;num2str;nn01;m4eval;;;0

```

### 8.2.53 expr

Applicable for: `bsd`

Same as the `eval` macro of `m4`.

```

0_define:\Bm4_expr(;;num2str;rr11 ;m4eval;;m4exec
0_def_mcall:expr;0_eval;0;nn00;m4;;m4builtin
0_setinfo:expr;( ;m4builtin

```

### 8.2.54 format

Applicable for: `gnu bsd`

This macro uses a pattern for the argument substitution to implement the `printf` like output.

```

0_pattern:m4format;[@00-@ff]~;stack_d 2
0_pattern:m4format;%;stack_e begin+0
0_tag_pat:m4format;1
0_append_pattern:m4format;-;
0_tag_pat:m4format;2
0_append_pattern:m4format;[0-9];stack_f begin+0
0_append_pattern:m4format;[0-9]*\D;stack_f get_st-1 endbegin
0_tag_pat:m4format;3
0_inter_pat:m4format;[0-9]+s;stack_e copyfr_f copyfr_d getarg copyto_d strlen - " " str* pu
0_tag_pat:m4format;4
0_inter_pat:m4format;[0-9]+s;stack_e copyfr_f copyfr_d getarg dup putoutst strlen - " " str
0_tag_pat:m4format;5
0_inter_pat:m4format;s;stack_e copyfr_d getarg putoutst stack_d 1 +
0_tag_pat:m4format;6
0_inter_pat:m4format;[1-9][0-9]*d;stack_e copyfr_f copyfr_d getarg 0 + copyto_d strlen - "
0_tag_pat:m4format;7
0_inter_pat:m4format;[0][0-9]+d;stack_e copyfr_f get_st+2 copyfr_d getarg 0 + copyto_d strl
0_tag_pat:m4format;8
0_inter_pat:m4format;d;stack_e copyfr_d getarg 0 + putoutst stack_d 1 +

```

```

0_link_pat:1;2;3;5;6;7;8
0_link_pat:2;4
0_define:\Bm4_format(;;;rr11 ;m4;m4format;;;m4exec
0_define:format;;;nn00;m4;m4format;;;m4builtin

```

### 8.2.55 dumpdef

Applicable for: posix gnu bsd  
 to do

```
0_define:\Bm4_dumpdef(;;;nr11 ;m4;;;m4exec
```

### 8.2.56 regexp

Applicable for: gnu bsd  
 to do

### 8.2.57 patsubst

Applicable for: gnu bsd  
 to do

### 8.2.58 traceon

Applicable for: posix gnu bsd  
 to do  
 nop

```

0_define:\Bm4_traceon\ ;;;nr11 ;m4;;;m4exec
0_define:traceon;;;nn00;m4;;;m4builtin
0_setinfo:traceon;\ ;m4builtin

```

### 8.2.59 traceoff

Applicable for: posix gnu bsd  
 to do  
 nop

```

0_define:\Bm4_traceoff\ ;;;nr11 ;m4;;;m4exec
0_define:traceoff;;;nn00;m4;;;m4builtin
0_setinfo:traceoff;\ ;m4builtin

```

### 8.2.60 debugmode

Applicable for: gnu  
 not implemented

### 8.2.61 debugfile

Applicable for: `gnu`  
 not implemented

### 8.2.62 end of emulation file, start of m4 files

Applicable for: `all`

Finally switch to the `m4exec` macro set.

The special chars for defining a set of characters is changed from "[" and "]" to bytes with values 1 and 2. This avoids an issue when similar characters in `m4` are used to set quotes or comments.

Extra input is inserted to be sure that all macros are recognised at the start and the end of the input file. These macros do not generate output.

```
0_chars_pattern:0_l10_r1-\@+*?~
0_atlast:++-+ 0_dnl
--+-
0_mset:m4exec
0_dnl
```

## 8.3 M6 emulation example

### 8.3.1 Introduction

The `m6` emulation is an example of defining a macro language in `m0`.

The `m6` macro processor is a predecessor of `m4`. It does not seem to be used anymore, but as an example it seems to be still useful.

Since there is no working `m6` for current systems, the testing of this emulation is limited.

### 8.3.2 Emulation

The following text of the emulation is generated from a source file for both text and code.

The original `m6` was written in Fortran and a later version can be found in Unix v6 written in a pre K&R C (see <https://minnie.tuhs.org/cgi-bin/utree.pl?file=V6/usr/source/m6>).

However in this emulation only one version that is based on *The M6 Macro Processor*, Andrew D. Hall, Bell Laboratories, April 12, 1972 is made and indicated with `all`.

### 8.3.3 Main design

A similar approach as is used in the `m4` emulation could be used for the `m6` emulation. However, because the macros in `m6` look a lot like S-expressions as known from Lisp, a different solution is chosen.

In the macro set for execution of `m6` macros a main macro is defined that will trigger on all macros and will call the macro placed in the first argument. This gives some difficulties,

because 3 macros (GO, GOBK and DNL) have a different behaviour for the collection of arguments. Thus the emulation design of m4 might be a better choice, but for demonstration the current design is also fine.

The main macro is defined in the m6exec macro set. The builtin and user macros are defined in the m6def macro set.

### 8.3.4 At the start

Applicable for: all

At the start the macros used to define the macros for m6 are defined. This is all straightforward linking of macro names to builtin functions.

```
0_define:0_pattern;;;pattern;nr01;0
0_define:0_set_var;;;set_var;nr01;0
0_define:0_get_var;;;get_var;nr01;0
0_define:0_chars_pattern;;;charpat;nr01;0
0_define:0_chars_args;;;chararg;nr01;0
0_define:0_specialchar;;;specialc;nr01;0
0_define:0_def_mcall;;;defmcall;nr01;0
0_define:0_atlast;;;at_last;nr01;0
0_define:0_mset;;;macroset;nr01;0
```

Setting the defaults. The first is also the default of m0. The second line defines the special characters for the arguments. This is missing the fifth character. It is thus not possible to select arguments from other stacks than the first stack.

```
0_chars_pattern:[]-\@+*?~
0_chars_args:1$*@#;101;102;103;m6exec
0_chars_args:1$*@#;101;102;103;m6def
0_chars_args:1$*@$;111;112;103
```

### 8.3.5 Quoting in default macro set

Applicable for: all

In the next part a macro with a pattern is used to have quoting in the macros in the default macro set 0 operational. The quoting is thus not a builtin function in m0.

```
0_set_var:111;+--+
0_set_var:112;-+-
0_pattern:0string;[@00-@ff]~;stack_b 1 stack_a begin-1
0_pattern:0string;+--+;stack_b 1 +
0_pattern:0string;-+-;stack_b 1 - dup =0 if stack_a end stop endif
0_define:+-+;$1;;nn00 ;0string
```

### 8.3.6 Quoting

Applicable for: all

In the next part a macro with a pattern is used to have quoting in m6.

```
0_set_var:101;<
```

```

0_set_var:102;>
0_set_var:103;,
0_pattern:string;[@00-@ff]~;stack_b 1 stack_a begin-1
0_pattern:string;<;stack_b 1 +
0_pattern:string;>;stack_b 1 - dup =0 if stack_a end stop endif
0_define:<;$1;;nn00 ;string;;;m6exec

```

### 8.3.7 Collecting arguments of macros

Applicable for: all

The collecting of arguments in m6 macros is executed with a pattern named m6 containing small programs.

The macros GO or GOBK are special cases. Therefore the pattern will detect these and then all arguments after the second argument are eaten when the argument equals 1. If the argument does not equal 1, then the pattern stops, so that the text after these macros is output.

In this case also an undocumented implementation feature is used to let the collection of arguments continue until the end of the buffer. In the implementation of m0, the replacement text is put in a memory buffer that is used as input to the bitap algorithm. At the end of the buffer the collection of arguments (in this case only for these two macros) ends and the execution of the macro continues. This results in the effect as described in m6 for these two macros.

The DNL macro is also a special case and is also performed in this pattern. The argument collection is stopped when a "\n" is encountered.

In the normal case all arguments 9 and larger are also collected together into argument 9. This is so defined in m6.

```

0_pattern:m6;[@00-@ff]~;begin-1 stack_b 0 stack_c 0
0_pattern:m6;++GO[:,;]-+;stack_b argnum <1 if 1 endif
0_pattern:m6;++GOBK[:,;]-+;stack_b argnum <1 if 1 endif
0_pattern:m6;++DNL[:,;]-+;stack_c argnum <1 if 1 endif
0_pattern:m6;;;argnum 9 < if stack_a end begin endif
0_pattern:m6;++;-+; recur_n
0_pattern:m6;++[:;]-+;+-+stack_a end stack_b dup =0 if
stack_c dup =0 ifthen stop
else
dup =1 if
  getarg2 =1 if
    2
    no_macro
  else
    stop
  endif
endif
endif -+-
0_pattern:m6;+-+

```

```
--;stack_c dup =1 if stack_a end stop endif
```

### 8.3.8 Main macro for executing all macros

Applicable for: `all`

The main macro will collect all arguments including the macro name and will make an `mcall` to the macro.

This is therefore a relatively simple macro.

```
0_def_mcall:###m6def;rr00 ;m6;;m6exec
```

### 8.3.9 GO, GOBK

Applicable for: `all`

These two macros are a bit different from normal macros. They will remove or keep the rest of a replacement text after the macro. They are similar to an `if` but their influence is outside of the macro.

Therefore the `m6` pattern will do the check of the `if` of the argument for these macros. The called macros are `nop` except for setting the no recursive in the `GO` macro.

```
0_define:GOBK;;;nn00;m6;;;m6def
0_define:GO;;;nn00;m6;;recur_n;m6def
```

### 8.3.10 DEF

Applicable for: `all`

The macros will be defined in the `m6def` macro set.

They will be called from the main macro. Therefore the argument collection has no significance.

```
0_define:DEF;;define;nn00;m6;;stack_a pop_to_3 "" "nn00" "" "" "" "m6def";m6def
```

### 8.3.11 COPY

Applicable for: `all`

```
0_define:COPY;;copy;nn00;m6;;stack_a pop_to_3 "m6def" "m6def";m6def
```

### 8.3.12 SEQ

Applicable for: `all`

```
0_define:SEQ;$1;;nn00;m6;;stack_a pop_to_3 strcmp;m6def
```

### 8.3.13 SNE

Applicable for: `all`

```
0_define:SNE;$1;;nn00;m6;;stack_a pop_to_3 strcmp !;m6def
```

**8.3.14 GT**

Applicable for: all

```
0_define:GT;$1;;nn00;m6;;stack_a pop_to_3 >;m6def
```

**8.3.15 GE**

Applicable for: all

```
0_define:GE;$1;;nn00;m6;;stack_a pop_to_3 >=;m6def
```

**8.3.16 LT**

Applicable for: all

```
0_define:LT;$1;;nn00;m6;;stack_a pop_to_3 <;m6def
```

**8.3.17 LE**

Applicable for: all

```
0_define:LE;$1;;nn00;m6;;stack_a pop_to_3 <=;m6def
```

**8.3.18 EQ**

Applicable for: all

```
0_define:EQ;$1;;nn00;m6;;stack_a pop_to_3 =;m6def
```

**8.3.19 NE**

Applicable for: all

```
0_define:NE;$1;;nn00;m6;;stack_a pop_to_3 !=;m6def
```

**8.3.20 IF**

Applicable for: all

The IF will call a macro in the 0 macro set to redo the arguments with the m6if pattern. This pattern will do the to do the comparisons and set the correct answer.

```
0_pattern:m6if;[@00-@ff]~;stack_a begin-1
0_pattern:m6if;[:,~];+--+stack_a end begin
stack_b argnum 2 mod =0 if
argnum 1 - getarg =1 if
argnum getarg
endif endif+--+
0_pattern:m6if;[:,~]; stop
0_def_mcall:IF;6_IF;0;nn00;m6;;m6def
0_define:6_IF;0_IF$@:;;rn00;m6;;;0
0_define:0_IF;$b0;;nr00;m6if;;;0
```

**8.3.21 SIZE**

Applicable for: all

```
0_define:SIZE;$1;;nn00;m6;;stack_a pop_to_2 strlen;m6def
```

**8.3.22 SUBSTR**

Applicable for: all

```
0_define:SUBSTR;;substr;nn00;m6;;;m6def
```

**8.3.23 ADD**

Applicable for: all

```
0_define:ADD;$1;;nn00;m6;;stack_a pop_to_3 +;m6def
```

**8.3.24 SUB**

Applicable for: all

```
0_define:SUB;$1;;nn00;m6;;stack_a pop_to_3 -;m6def
```

**8.3.25 MPY**

Applicable for: all

```
0_define:MPY;$1;;nn00;m6;;stack_a pop_to_3 *;m6def
```

**8.3.26 DIV**

Applicable for: all

```
0_define:DIV;$1;;nn00;m6;;++stack_b getarg2 =0 if
"" putarg1 else stack_a pop_to_3 / endif--;m6def
```

**8.3.27 EXP**

Applicable for: all

```
0_define:EXP;$1;;nn00;m6;;++stack_b getarg2 <0 if
"" putarg1 else stack_a pop_to_3 power endif--;m6def
```

**8.3.28 DNL**

Applicable for: all

The DNL does nothing. All the work is performed in the m6 pattern.

```
0_define:DNL;;;nn00;;;m6def
```

### 8.3.29 Source, End, Trace

Applicable for: `all`

These macros are not implemented.

### 8.3.30 end of emulation file, start of m6 files

Applicable for: `all`

Finally switch to the `m6exec` macro set.

`0_mset:m6exec`

## 8.4 GPM emulation example

### 8.4.1 Introduction

The `gpm` emulation is an example of defining a macro language in `m0`.

The `gpm` macro processor is an early macro processor and is considered a very early predecessor of `m4`. It does not seem to be used anymore, but as an example it seems to be still useful.

Since there is no working `gpm` macro processor for current systems, the testing of this emulation is limited.

### 8.4.2 Emulation

The following text of the emulation is generated from a source file for both text and code.

In this emulation the version that is described in *GPMX A portable general purpose macro processor adapted for preprocessing FORTRAN*, Robert C. Gammill, The Rand Corporation, Santa Monica, California, 7 June 1976 is made and indicated with `all`.

### 8.4.3 Main design

A similar approach as is used in the `m6` emulation is used for the `gpm` emulation.

In the macro set for execution of `gpm` macros a main macro is defined that will trigger on all macros and will call the macro placed in the first argument. The use of a macro call is necessary because it seems a practise that called macros are first defined during argument collection. Thus the called macro is not yet defined when called.

The main macro is defined in the `gpmexec` macro set. The builtin and user macros are defined in the `gpmdef` macro set.

GPM has only 6 macros builtin. It is thus small and the characteristics of `gpm` are used to add complexity.

### 8.4.4 At the start

Applicable for: `all`

At the start the macros used to define the macros for `gpm` are defined. This is all straightforward linking of macro names to builtin functions.

```
0_define:0_pattern;;;pattern;nr01;0
0_define:0_set_var;;;set_var;nr01;0
```

```

0_define:0_get_var;;;get_var;nr01;0
0_define:0_chars_pattern;;;charpat;nr01;0
0_define:0_chars_args;;;chararg;nr01;0
0_define:0_specialchar;;;specialc;nr01;0
0_define:0_def_mcall;;;defmcall;nr01;0
0_define:0_atlast;;;at_last;nr01;0
0_define:0_mset;;;macroset;nr01;0
0_define:0_char;;;num2chr;nr01;0
0_define:0_program;;;program;nr01;0

```

Setting the defaults. These are different from the defaults of `m0`. The second line defines the special characters for the arguments. This is missing the fifth character. It is thus not possible to select arguments from other stacks than the first stack.

```

0_chars_pattern:{}-\@+*?~
0_chars_args:1%*#@#;101;102;103;gpmexec
0_chars_args:1%*#@#;101;102;103;gpmdef
0_chars_args:1$*#@#$;111;112;103

```

### 8.4.5 Quoting in default macro set

Applicable for: all

In the next part a macro with a pattern is used to have quoting in the macros in the default macro set 0 operational. The quoting is thus not a builtin function in `m0`.

```

0_set_var:111;+--+
0_set_var:112;-+-
0_pattern:0string;{@00-@ff}~;stack_b 1 stack_a begin-1
0_pattern:0string;+--+;stack_b 1 +
0_pattern:0string;-+-;stack_b 1 - dup =0 if stack_a end stop endif
0_define:+-+;$1;;nn00 ;0string

```

### 8.4.6 Quoting

Applicable for: all

In the next part a macro with a pattern is used to have quoting in `gpm` in a similar way as `m6`.

```

0_set_var:101;<
0_set_var:102;>
0_set_var:103;;
0_pattern:string;{@00-@ff}~;stack_b 1 stack_a begin-1
0_pattern:string;<;stack_b 1 +
0_pattern:string;>;stack_b 1 - dup =0 if stack_a end stop endif
0_define:<;%1;;nn00 ;string;;;gpmexec

```

### 8.4.7 Collecting arguments of macros

Applicable for: all

The collecting of arguments in `gpm` macros is executed with a pattern named `gpm` containing small programs.

This pattern will also recognise the arithmetic for the `BAR` macro. It will push the operators to the operator stack. The `BAR` macro can execute these operators.

```
0_pattern:gpm;{@00-@ff}~;stack_a begin-1
0_pattern:gpm;;;stack_a end begin
0_pattern:gpm;];stack_a end stop
0_pattern:gpm;+::;stack_a 0 oppush +
0_pattern:gpm;-::;stack_a 0 oppush -
0_pattern:gpm;*::;stack_a 0 oppush *
0_pattern:gpm;/::;stack_a 0 oppush /
```

### 8.4.8 Substitution of arguments

Applicable for: `all`

The emulation is not using the default argument substitution, because the local defined macros should be deleted at the end of a macro. The argument substitution is executed at the end of a macro and is therefore the correct place to delete local macros.

Stack `h` holds the names of macros defined. This stack is not freed at the end of a `DEF`.

A char with value 0 is used in macros to execute the deletion of the macros whereby the name of the macros are stored on stack `h`.

A char with value 3 is used in the `DEF` macro to execute the deletion of macros and to store the defined macro on stack `h`.

A char with value 2 is used in the `VAL` macro to output the second argument and delete the last char (the char with value 0).

```
0_define:0_zero;0_char:0
;;nn00;;;0
0_define:0_three;0_char:3
;;nn00;;;0
0_define:0_two;0_char:2
;;nn00;;;0
0_pattern:arggpm;%{0-9};stack_b getlast1 getarg putout
0_pattern:arggpm;0_zero;+-+ stack_a pop_to_2 "gpmdef"
stack_h while_st putarg1 "pop" fun_call pop endwhile
"" putout -+-
0_pattern:arggpm;0_three;+-+ stack_a pop_to_2 copyto_c "gpmdef"
stack_h while_st putarg1 "pop" fun_call pop endwhile
copyfr_c free_no "" putout -+-
0_pattern:arggpm;0_two;stack_b getarg2 putout "" putout
```

### 8.4.9 Main macro for executing all macros

Applicable for: `all`

The main macro will collect all arguments including the macro name and will make an `mcall` to the macro.

This is therefore a relatively simple macro.

```
0_def_mcall:[;;gpmdef;rr00;gpm;;gpmexec
```

#### 8.4.10 DEF

Applicable for: all

The macros will be defined in the `gpmdef` macro set.

They will be called from the main macro. Therefore the argument collection has no significance.

A program to set all the options for the `define` or `push` function is defined to be used by the `DEF` and the `UPDATE` macro. The definition of a new macro is appended with a zero character to activate the deletion of macros at the end of a macro by the `arggpm` argument substitution.

```
0_program:defargs;--+
stack_a pop_to_3 "" "nn00" "gpm" "arggpm" "" "gpmdef"
stack_c getarg2 +--"0_zero"+--+ cat putarg2+--
```

The `DEF` uses the `push` function so that it can also be popped at the end of macros.

```
0_define:DEF;0_three;push;nn00;gpm;arggpm; @defargs ;gpmdef
```

#### 8.4.11 UPDATE

Applicable for: all

Is similar to `DEF`, but uses a `define` instead of a `push`.

```
0_define:UPDATE;0_zero;define;nn00;gpm;arggpm; @defargs ;gpmdef
```

#### 8.4.12 BIN

Applicable for: all

This macro has nothing to do. In `m0` integers and strings are automatically converted.

```
0_define:BIN;%10_zero;;nn00;gpm;arggpm;;gpmdef
```

#### 8.4.13 DEC

Applicable for: all

Similar to `BIN`.

```
0_define:DEC;%10_zero;;nn00;gpm;arggpm;;gpmdef
```

#### 8.4.14 VAL

Applicable for: all

Links to `info`, but needs to also output char with 0 to the output.

```
0_define:VAL;0_two0_zero;info;nn00;gpm;arggpm;recur_n;gpmdef
```

### 8.4.15 BAR

Applicable for: `all`

The operators are already stored by the `gpm` pattern. So operator needs only to be executed here.

```
0_define:BAR;%20_zero;;nn00;gpm;arggpm;stack_a opexall;gpmdef
```

### 8.4.16 end of emulation file, start of gpm files

Applicable for: `all`

Finally switch to the `gpmexec` macro set.

```
0_mset:gpmexec
```



## Appendix A Index for m0 builtins and stack instructions

### A.1 Builtin functions

This index covers all m0 builtin functions. References are exclusively to the places where a function is introduced the first time.

#### A

|                |    |
|----------------|----|
| append_p ..... | 35 |
| at_first ..... | 42 |
| at_last .....  | 41 |

#### C

|                |    |
|----------------|----|
| chararg .....  | 37 |
| charpat .....  | 36 |
| clr_pat .....  | 36 |
| copy .....     | 29 |
| copy_pat ..... | 36 |

#### D

|                |    |
|----------------|----|
| define .....   | 25 |
| defmcall ..... | 31 |
| divert .....   | 41 |

#### E

|                |    |
|----------------|----|
| errprint ..... | 42 |
| exit .....     | 42 |

#### G

|               |    |
|---------------|----|
| get_var ..... | 39 |
|---------------|----|

#### I

|                |    |
|----------------|----|
| ifmacro? ..... | 40 |
| include .....  | 41 |
| info .....     | 40 |
| inter_p .....  | 35 |

#### L

|                |    |
|----------------|----|
| link_prt ..... | 31 |
|----------------|----|

#### M

|                |    |
|----------------|----|
| macroset ..... | 31 |
|----------------|----|

#### N

|               |    |
|---------------|----|
| num2chr ..... | 38 |
| num2str ..... | 39 |

#### P

|                |    |
|----------------|----|
| part_m .....   | 30 |
| pattern .....  | 35 |
| pop .....      | 29 |
| program .....  | 36 |
| pshmcall ..... | 34 |
| push .....     | 29 |
| pushcopy ..... | 29 |

#### S

|                |    |
|----------------|----|
| set_info ..... | 30 |
| set_var .....  | 39 |
| shell .....    | 42 |
| sinclude ..... | 41 |
| specialc ..... | 36 |
| strindex ..... | 38 |
| strtrans ..... | 38 |
| substr .....   | 38 |

#### T

|                |    |
|----------------|----|
| tag_m .....    | 31 |
| tag_p .....    | 35 |
| tempfile ..... | 42 |

#### U

|                |    |
|----------------|----|
| undefine ..... | 29 |
| undivert ..... | 41 |

### A.2 Instructions

This index covers all all instructions used in programs that work on the stacks. References are exclusively to the places where an instruction is introduced the first time.

**!**

|          |    |
|----------|----|
| ! .....  | 55 |
| != ..... | 52 |

**\***

|         |    |
|---------|----|
| * ..... | 53 |
|---------|----|

**+**

|         |    |
|---------|----|
| + ..... | 53 |
|---------|----|

**-**

|         |    |
|---------|----|
| - ..... | 53 |
|---------|----|

**/**

|         |    |
|---------|----|
| / ..... | 53 |
|---------|----|

**<**

|           |    |
|-----------|----|
| < .....   | 53 |
| <<1 ..... | 54 |
| <= .....  | 53 |
| <=? ..... | 52 |
| <? .....  | 52 |

**=**

|          |    |
|----------|----|
| = .....  | 52 |
| =? ..... | 52 |

**>**

|           |    |
|-----------|----|
| > .....   | 53 |
| >= .....  | 53 |
| >=? ..... | 52 |
| >>1 ..... | 54 |
| >? .....  | 52 |

**~**

|         |    |
|---------|----|
| ~ ..... | 55 |
|---------|----|

**A**

|              |    |
|--------------|----|
| abort .....  | 50 |
| and .....    | 55 |
| argnum ..... | 51 |
| argpos ..... | 51 |

**B**

|                |    |
|----------------|----|
| base=_1 .....  | 48 |
| begin .....    | 49 |
| begin+? .....  | 49 |
| begin-? .....  | 49 |
| bit_and .....  | 54 |
| bit_exor ..... | 55 |
| bit_or .....   | 54 |

**C**

|                |    |
|----------------|----|
| cat .....      | 55 |
| cat-1 .....    | 55 |
| cnt++ .....    | 62 |
| cnt-- .....    | 62 |
| cnt_clr .....  | 62 |
| cnt_get .....  | 62 |
| cnt_set .....  | 62 |
| copyfr_? ..... | 48 |
| copyto_? ..... | 48 |

**D**

|           |    |
|-----------|----|
| dup ..... | 47 |
|-----------|----|

**E**

|                |    |
|----------------|----|
| end .....      | 49 |
| end-? .....    | 50 |
| endbegin ..... | 58 |
| endwhile ..... | 51 |

**F**

|                |    |
|----------------|----|
| free_no .....  | 63 |
| free_yes ..... | 63 |
| fun_call ..... | 63 |

**G**

|                |    |
|----------------|----|
| get_st .....   | 57 |
| get_st+? ..... | 57 |
| get_st-? ..... | 57 |
| getarg .....   | 49 |
| getarg? .....  | 49 |
| getlast? ..... | 58 |
| goback? .....  | 50 |

**I**

|                     |    |
|---------------------|----|
| if else endif ..... | 50 |
| ifcmpset .....      | 56 |
| ifthen .....        | 50 |

**M**

|                |    |
|----------------|----|
| m?_col_p ..... | 61 |
| m?_def .....   | 61 |
| m?_info .....  | 61 |
| m?_int .....   | 61 |
| m?_name .....  | 60 |
| m?_num .....   | 60 |
| m?_opt .....   | 61 |
| m?_prog .....  | 61 |
| m?_sub_p ..... | 61 |
| m?_type .....  | 61 |
| m_depth .....  | 51 |
| mod .....      | 54 |

**N**

|                |    |
|----------------|----|
| no_macro ..... | 59 |
| nooverr .....  | 59 |
| nop .....      | 51 |

**O**

|                |    |
|----------------|----|
| opexall .....  | 60 |
| opexif< .....  | 59 |
| opexif<= ..... | 60 |
| opexif> .....  | 59 |
| opexif>= ..... | 59 |
| opexto .....   | 60 |
| oppush .....   | 60 |
| or .....       | 55 |

**P**

|                |    |
|----------------|----|
| pop .....      | 47 |
| pop_to_? ..... | 47 |
| power .....    | 54 |
| putarg .....   | 48 |
| putarg? .....  | 48 |
| putout .....   | 58 |
| putout-? ..... | 58 |
| putoutst ..... | 58 |

**R**

|               |    |
|---------------|----|
| recur_n ..... | 58 |
| recur_y ..... | 58 |

**S**

|                |    |
|----------------|----|
| set_base ..... | 48 |
| shift_l .....  | 54 |
| shift_r .....  | 54 |
| stack_? .....  | 47 |
| stop .....     | 50 |
| str* .....     | 57 |
| strcmp .....   | 56 |
| strlen .....   | 57 |
| swap .....     | 47 |
| swap-12 .....  | 48 |

**W**

|                |    |
|----------------|----|
| while .....    | 51 |
| while_st ..... | 51 |



## Appendix B How to make copies of the overall M0 package

This appendix covers the license for copying the source code of the overall M0 package. This manual is under a different set of restrictions, covered later (see Appendix C [Copying This Manual], page 111).

### B.1 License for copying the M0 package

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

#### Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic

pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

## TERMS AND CONDITIONS

### 0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

### 1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

## 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

## 3. Protecting Users’ Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

#### 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

#### 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

## 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether

the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

## 7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

- b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

## 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have

been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such

a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work.

The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public 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.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that

most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

## END OF TERMS AND CONDITIONS

### How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author
```

```
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.
```

```
This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <https://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <https://www.gnu.org/licenses/why-not-lgpl.html>.

## B.2 License of the SDS library used in the M0 package

SDSLib 2.0 – A C dynamic strings library

Copyright © 2006-2015, Salvatore Sanfilippo <antirez at gmail dot com>

Copyright © 2015, Oran Agra

Copyright © 2015, Redis Labs, Inc

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Redis nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## Appendix C How to make copies of this manual

This appendix covers the license for copying this manual.

### C.1 License for copying this manual

Version 1.3, 3 November 2008

Copyright © 2000–2002, 2007–2008, 2022–2025 Free Software  
Foundation, Inc.  
<https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

#### 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful 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 non-commercially. Secondly, 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, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

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. (Thus, 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. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

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 Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document 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, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. 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, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include 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, PostScript or PDF 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.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to

be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 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 (or copies in media that commonly have printed covers) 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 computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. 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 fewer than five), unless they release you from this requirement.
- 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", Preserve 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. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, 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 to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

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, and that you preserve all their Warranty Disclaimers.

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, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which 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 half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket 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, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have

been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

#### 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 <https://www.gnu.org/licenses/>.

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. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

#### 11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## 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 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

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.