Mawk 1.9.9.6

The file, `mawk-1.9.9.6.tar.gz`, contains the source code for `mawk`, an implementation of the Awk programming language, that runs on unix computers. `mawk 1.9.9.6` is a beta release for `mawk 2.0.0`.

I am the developer, Mike Brennan `mawkeddy@gmail.com`. I first released `mawk 1.0` in 1991 and last released `mawk 1.3.3` in 1996. (A few people had `mawk 1.3.3.1` with `nextfile`, 1999.)

Why a 25 and 20 year anniversary release? Because I always knew a few things could be done better and design decisions that were right for the 90's were wrong for 21st century.

In my absence, there have been other developers that produced `mawk 1.3.4-xxx`. I started from `1.3.3` and there is no code from the `1.3.4` developers in this `mawk`, because their work either did not address my concerns or inadequately addressed my concerns or, in some cases, was wrong. I did look at the bug reports and fixed those that applied to `1.3.3`. I did switch to the FNV-1a hash function as suggested in a bug report.

Here is what is new.

(1) Oddly written but legal regular expressions could cause exponential blowup of execution time versus input length. Consider,

```
$ mawk '!/(a|aa)*Z/'    aN
```

where the contents of file `aN` is one line with `N` `a`'s terminated by `X`. E.g.,

`a5` is `aaaaaX`
`a10` is `aaaaaaaaaaX`
`a20` is `aaaaaaaaaaaaaaaaaaaaX`
etc

On a 5000 bogomips box, using `mawk133`, running times are:

`a5`	.002 sec
`a20`	.005 sec
`a40`	53.2 sec
`a50`	1 hour 41 min
`a1000`	more seconds than there are atoms in the universe

This released `mawk` does `a1000` in .005 seconds.

For reasonably written regular expressions and normal input, this bug for most people never came up. In that sense, it is a minor bug. However in the sense that a regular expression algorithm should have linear execution time relative to the input length in all cases, it was a major error by me.

(2) Fixed limit on the number of fields, `$1 $2 ...` is removed.

(3) Fixed limit on length of a string produced by `sprintf()` is removed.

(4) Sizes chosen for 1991-96 have been adjusted for the 21st century. Most important, the input buffer is bigger and grows faster to handle long input records. The memory allocator blocks are bigger. The hash tables have more slots.

(5) `gsub()` is no longer recursive which makes it faster and more reliable. `^` is handled correctly.

(6) `printf` and `sprintf` handle bigger integers. For example,

```
$ mawk 'BEGIN{ printf "%x %x %d\n", -1, 2^63, -2^63}'
ffffffffffffffff 8000000000000000 -9223372036854775808
```

Awk prints an integer as an integer (%d) and other numbers using OFMT (default to %.6g). The new mawk recognizes bigger integers.

```
$ mawk133 'BEGIN { print 2^33}'
8.58993e+09

$ mawk 'BEGIN { print 2^33}'
8589934592
```

In this area, there is a mild disagreement between gawk and mawk.

```
$ mawk 'BEGIN { print exp(37)}'
1.17191e+16

$ gawk 'BEGIN { print exp(37)}'
11719142372802612
```

The actual value is 11719142372802611.3086...

(7) The character '\0' (zero) can be an element of a string.

(8) Design of arrays was simplified. No effect from user perspective, but more maintainable from developer perspective.

(9) nextfile.

(10) length(A) where A is an array returns the number of elements in the array.

(11) Backslash in replacement strings.

```
$ echo ABC | mawk133 '{sub(/B/,"\\\\") ; print}'
A\C

$ echo ABC | mawk '{sub(/B/,"\\\\") ; print}'
A\\C
```

The 133 behavior follows the early 90's posix spec, but it is confusing that a string without & is altered. Gawk and Kernighan's awk do it differently and now mawk agrees with them.

\ escapes \ and \ escapes &, but only if the run of \ ends in &.

For example,

```
$ echo ABC | mawk '{sub(/B/,"\\\\&") ; print}'
A\BC
```

(12) Some years ago,

```
$ echo 0x4  inf nan | awk '{ print 7 + $1, 8 + $2, 9+$3}'
7 8 9
```

for all awk's, but now

```
$ echo 0x4  inf nan | mawk133 '{ print 7 + $1, 8 + $2, 9+$3}'
11 inf nan
```

What changed was the C-library strtod() started recognizing inf, nan and hex strings. But changes for a low level C library, are not right for a high level language like awk. So, in agreement with gawk, the new mawk gives the old result.

```
$ echo 0x4  inf nan | mawk '{ print 7 + $1, 8 + $2, 9+$3}'
7 8 9
```

(13) Regular expression character classes such as /[[:digit:]]/ are now supported. The complete list is alnum, alpha, blank, cntrl, digit, graph, lower, print, space, upper and xdigit.

The manual pages have not yet been updated. The options have minor changes that can be viewed with

```
$ mawk --help
```

Installation

Download `mawk-1.9.9.6.tar.gz` to a working directory.

```
$ tar xfz mawk-1.9.9.6.tar.gz
$ cd mawk-1.9.9.6
$ ./configure
$ make
$ make check
```

If all went well, you can copy `mawk` to a `bin` directory. `$ make install` will copy it to `$prefix/bin`, which is usually `/usr/local/bin`.

Changes from `1.9.9.2`

`1.9.9.3` changed `pipetest` to `./pipetest` in script, `test/mawktest`.

`1.9.9.4` renamed `man/mawk.doc` to `man/mawk.txt` and minor edits to output of `--help`.

`1.9.9.5` added `#include <sys/wait.h>` to `bi_funct.c`. In `fin.c`, `"/dev/stdin"` is treated like `"-"`, and input `fd 0` is not closed by `FINclose()`.

`1.9.9.6` replaced explicit constants with `INT64_MAX`, `INT64_MIN` and `UINT64_MAX` in `int.c`. Export `PATH` in `test/mawktest`. Moved `#include "int.h"` from `array.c` to `array.h`.