Tutorial: new functions; documentation, tests

Karim Belabas

http://pari.math.u-bordeaux.fr/

Initial write-up

Basic Scenario : function `foo()` is implemented in `blah.c`, either an existing file in the PARI distribution, or a new one (No need to fiddle with `Makefiles`, `Configure` will pick up new code modules.)

We assume it follows our *Coding Guidelines*

`http://pari.math.u-bordeaux1.fr/codeguide.html`

In particular :

- It does not introduce new global variables ; and every auxiliary function is either public (declared in `paridecl.h` and documented in `usersch[5-7].tex`) or `static`.
- It compiles cleanly (no warnings) on a C89 and C++ compiler.
- It does not introduce a `[BUG]` in the `make statest-all` suite, unless you have a good explanation for it.

Sanity checks

- Add type checks and try to recover from insane inputs, provided the checks are inexpensive. If not, you may document the behaviour as undefined when the preconditions are not met. (A bug report associated to undefined behaviour will be classified as a Bad Input Bug, usually `wontfix`.)
- Make sure your return values satisfy `gerepileupto` prerequisites ; also in degenerate cases. Beware of `mkvec` and friends !
- `install()` your public function in `gp` and test it on trivial inputs.
- Test it again, writing a basic test suite for your public function. Any mathematically significant test is good, esp. if the output is deterministic and does not depend on the architecture. It does not matter if the test is small : later bug reports will enrich it.
- Run the test again, under `valgrind`. Debug using `valgrind --db-attach=yes`. For instance using the `gp.dbg` to call your install'ed function.

Profiling

- Use `gprof`. Is the (time) complexity roughly what you expected ?
- Test with small `parisize`. Is the (space) complexity what you expected ? If not, add garbage collection.
- If you used random garbage collection, try to disable it :

  ```
  #define low_stack(a,b) (1)
  ```

  at the start or your module, then re-run your test.

Inclusion in GP

- Write a proper `functions/*/foo` description for your function. Required fields :

  `Function`

  `Section`

  `C-Name`,

  `Prototype`

  `Help`

  `Doc`

- Remove `install` statements from your tests and dump the test file, say `foo` in `src/test/in`. Build a reference output (`Configure`, `make test-foo`, `patch`); also for 32-bit architectures.

- Go back to `functions/*/foo` and add some examples in `Doc`.

That's it!