Form Workshop 2026

June 2026

1 Introduction

The goal of the hands-on sessions is not only to fix bugs, but to practise the whole development loop for FORM:

1. reduce a user reported issue to a small Form program,
2. find the relevant compiler/runtime/manual code,
3. fix the bug,
4. add a regression test, and
5. prepare a pull request that is easy to review and easy to bisect.

Good practice:

- Work on your own fork of the repository. Iterate there, force-push there if you like.
- Create a clean pull request back into form-dev/form.
- Each commit mergeed needs to work properly on its own, so that later bisection is meaningful. CI helps enforce this, but local tests are still the fastest feedback.
- See also `https://github.com/form-dev/form/wiki/VS-Code-Tips-for-FORM-Developers` for more tips.
- The current issue tracker is `https://github.com/form-dev/form/issues`.

2 Exercises

2.1 Learn the source code

These exercises are meant to make the sources less mysterious. They do not need to end in a pull request.

1. **Trace one statement from parser to execution.** Start with `DropCoefficient`. Find the compiler entry, the opcode or statement representation, and the runtime implementation. Temporarily change its behaviour so that dropping the coefficient leaves for example coefficient `2/1` instead of `1/1`. Does the test suite now fail?

   Hint: use `grep` or `rg`, e.g. `rg -i DropCoefficient`.

2. **Inspect a term.** Add `MesPrint("term = %a",*term,term);` at a place where terms are processed. Which words correspond to term size, coefficient, symbols, functions, vectors, etc?

3. **Add your own statement.** Add a statement such as `Workshop;` that compiles, prints a message, and does nothing to the expression. Hint: see how `DropCoefficient` is implemented.

4. **Write coverage tests**, see also 3.1. This is a good way to learn the source code.

2.2 Bug-fix candidates

1. Find minimal reproducing Form programs for reported Issues.

2. Fix a bug from the Milestones

2.3 Easier (probably?) exercises

1. Implement `Dropcoefficient;` for the floats. Do we need to extend the syntax, e.g. `Dropcoefficient, float;`? Don't forget to change the manual as well.

2. Print a warning the first time a higher-level sort, in the "sub-buffers", spills into a disk file.

3. Try to increase `FIRSTUSERFUNCTION`, `ARGHEAD` or `FUNHEAD`. It probably compiles, but does it pass the test suite?

4. `ModuleOption statistics;` Enable statistics printing for single module only.

5. User-defined kind label for C print mode: C++11 has [user-defined literals]. `Format C, _kind;`

6. Extend the new `On SortVerbose;` statistics mode, to also collect the maximal term size encountered at any stage of the sort. Hint: add a member to the `SORTING` struct, and check term sizes in `StoreTerm` and `PutOut/PutToMaster`.

7. Fix [836], float function after `#EndFloat`, so they become a regular function. Hint: an active `float_` function is usually at the end of the term, however, regular functions are not necessary at the end. This might break the test suite.

2.4 Medium exercises

1. `On InParallel;` Multi-module `InParallel;`

2. Fix [#445] Tensors with 0 for argument vanish.

3. a. Fix `ranperm_(t,?a)` when `t` is a tensor (see also #751). For e.g., `ranperm_(t,1,2,3) -> t(20_,2,20_,3,20_,1)`.

   b. Fix `replace_(fun,tens)` at compiler level and runtime (from dollar var).

4. There are already predefined functions `tofloat_` and `torat_`, but they are not hooked up to anything yet. Implement them.

5. Add the option for `TryReplace x,<number>;` in the compiler routine `CoTryReplace`. More advanced, also allow for `TryReplace x,y^2;` and more difficult replacement patterns that are allowed in the `replace_` function.

2.5 Harder, longer term

1. Parallelize `Local G = F;`. This can be a large performance bottleneck for large expressions.

2. Compress the scratch files (zlib, zstd).

   - Complication due to Bracket index.
   - Disable if an index is created? Do this first?
   - Compress each bracket's content (possibly poor performance?)

3. Remove/improve `MAXSUBEXPRESSIONS` limitation? Annoying when loading enormous text files.

4. `#includemma` load Mathematica-format text files directly?

5. Factorized `PolyRatFun`? `FactPolyRatFun`? `PolyFactFun`?

   - Factorizing denominators has been beneficial for IBP reduction (FIRE+Symbolica).

- Saves on `MaxTermSize` budget.
- `fprf(num, lst (den1,pow1), lst (den2,pow2), ...)` ? `fprf(num, den1,pow1, den2,pow2, ...)` ?

6. Facilities for rational reconstruction from samples over prime fields:

```
#startreconstruct F
...
#endreconstruct
```

7. Automatic replacement of dot products with symbols?

```
Symbol s;
Vector q1,q2;
DotProduct q1.q1 = 0, q2.q2 = 0, q1.q2 = s/2;
Local test = g_(1,q1,q2,q1,q2);
Tracen 1;
Print;
.end
```

8. `Save -gzip filename;` plus of course the corresponding reading in the `Load` statement.

9. Look for statements that forget to expand dollar variables in their parameter field. (Note that some commands intentionally don't expand, for eg gcd_ etc.)

10. What would be needed to remove dollar variables from the administration? Can you implement that? Use `Delete` statement?

11. What needs to be done to create a `#return` inside a procedure?

3 Improve Robustness of Form

3.1 Test suite and Coverage

Improve coverage with targeted tests. Pick one source file or one statement, inspect uncovered branches using the coverage report, and add one or two tests that exercise real behaviour. AI-generated tests are allowed, but every assertion must be checked by a human. Preferred tests are those that document behaviour users care about. Good start: find statements that are not covered yet, see also `compcomm.c`.

Useful local commands:

```
# run the complete test suite
make check
```

```
# run one regression test while iterating
cd check
./check.rb ../sources/form Issue267
```

The format for tests is documented in `check/README.md`. Tests go in either one of the following files:

- `check/coverage.frm`: Tests whose primary purpose is increasing test suite coverage.
- `check/examples.frm` Tests for the diagram generator.
- `check/examples.frm` Examples from the manual.
- `check/features.frm`: Tests for new features.
- `check/fixes.frm` The most common destination for bug-fix tests.
- `check/polynomial.frm` Flint poly tests.
- `check/user.frm`: User tests which don't fit the above.

Hints: use `assert succeeded?` for ordinary successful programs, `assert result("F") = expr("...")` for printed expressions, `assert runtime_error?` for expected runtime failures, and `assert preprocess_error?("...")` or `assert compile_error?("...")` for diagnostics.

You can either view the coverage here: coverage report, or run it locally on your own laptop with the following script. You can also fetch the `formlib` files directly for the tests `check/extra` with this script.

3.2 Documentation and examples

Improve the documentation. In particular, improve descriptions in the manual, fix mistakes and add more useful examples to the manual. When adding new examples, don't forget to add the example also as a test to `check/examples.frm`.

- [#801] **Missing documentation for `illegal coupling const`.**
- [#635] **Document how to match unequal symbols.**
- [#506] **Collect developer tips.**
- [#498] **Refresh part of the developer reference.**

3.3 Tools

1. **Fuzzing with afl++.** Start from [#663].
2. **Run benchmarks.** Run the available benchmark script. Compare FORM 5 with the current master branch on your favourite FORM program. How much faster is it? Or do you see any regressions?

4 What we want to decide at the end of the week

1. Which legacy features should be deprecated? Do we want to make the float, flint etc features mandatory builds. Do we get rid of
   - parform,
   - native Windows version,
   - 32-bit version of Form?
2. Mandatory libraries/features: GMP, MPRF (float system), zlib, zstd, flint?
3. Syntax decisions?
   - block comments?
   - `DotProduct q1.q1 = 0;`?
   - `Save -gzip filename;`? `Save filename.gz;`? Or zip by default?
   - `FactPolyRatFun`? `fprf(num,den1,pow1,den2,pow2)`? `fprf(num,list_(den1,pow1),list_(den2,pow2))`? Could also re-use `list_` for `mpl_`?
   - `DropCoefficient, float;` Or just `DropCoefficient;`
   - `Transform int multiaddargs(4,1,last);`
     For e.g. instead of

     ```
     Repeat Identify int(n1?,n2?,n3?,n4?)*int(m1?,m2?,m3?
     ,m4?) = int(n1+m1,n2+m2,n3+m3,n4+m4);
     ```

     we could use

     ```
     ChainIn int;
     Transform int multiaddargs(4,1,last);
     ```

   - Rename a function name when inside that function with `Argument;`. Use `Transform` for this?
4. Package manager for Form
5. Namespaces for Form.