1 The M2/Macaulay2 Directory

This directory contains a large number of subdirectories. A few of note are

- m2: Macaulay2 code implementing top level functions and operations.
- d: Intermediate code (D language, but not that D language) implementing routines that call engine code.
- e: Engine code in C++.
- packages: Contributed packages consisting of Macaulay2 code.

2 Building

Use BUILD/dan for building M2 from source, as this keeps things cleaner.

3 Adding a C++ function

Adding a C++ function in the engine that can be called from top level involves adding several commands that thread the function and arguments through the interpreter. The following is an example.

Suppose you want to add a function divisionAlgorithm at the top level. This function will take as input a matrix of polynomials f, a matrix of polynomials g, and a strategy. It will return a matrix with columns the columns of f reduced by polynomial division by g. There are three changes that need to be made.

First, a declaration of the C++ function must be added in e/engine.h.

```
/************************************************/
/**** Division Algorithm ************************/
/************************************************/

const Matrix * rawDivisionAlgorithm(const Matrix *f,
                                    const Matrix *g,
                                    int strategy);
/* Reduce every column of f with respect to the columns of g. Currently strategy is ignored. */
```

Then the actual C++ function must be defined in some file in e. For example, we add the following to x-mat.cpp.

```
const Matrix * rawDivisionAlgorithm(const Matrix *f,
                                    const Matrix *g,
                                    int strategy)
{
  ERROR("Not implemented yet");
  return nullptr;
}
```

Next, we need code in d/interface.dd to thread the function through the interpreter to the engine. Add the following to that file.

```
export rawDivisionAlgorithm(e:Expr):Expr := (
  when e is s:Sequence do (
  if length(s) == 3 then (
  when s.0 is wf:RawMatrixCell do (f := wf.p;
  when s.1 is wg:RawMatrixCell do (g := wg.p;
  when s.2 is wstrategy:ZZcell do (
  if isInt(wstrategy) then (strategy := toInt(wstrategy);
```

```
    toExpr(Ccode(RawMatrixOrNull,
      "rawDivisionAlgorithm(",
        f, ",", g, ",", strategy,
      ")"
    ))
  ) else WrongArgSmallInteger(2)
  ) else WrongArgZZ(2)
  ) else WrongArg(1,"a raw matrix")
  ) else WrongArg(0,"a raw matrix")
  ) else WrongNumArgs(3)
  ) else WrongNumArgs(3)
  );
setupfun("rawDivisionAlgorithm",rawDivisionAlgorithm);
```

This code can be created by a helper function in the GenerateD package.

```
debug needsPackage "GenerateD"
fcns = {
    (
        "rawDivisionAlgorithm",
        "MatrixOrNull",
        toSequence {"a"=>"Matrix","b"=>"Matrix", "strategy"=>"int"}
        )
    }
str between("",for args in fcns list genFunctionCall args)
```

Finally, we need a top level function that initiates the call to our engine code. Many of these functions and operations are defined in the m2 directory, but for ease of testing and debugging we will instead define our top level function in a new package in the package directory. Inside a new package we write

```
export {"divAlg"}

debug Core;

divAlg = method();
divAlg(Matrix, Matrix) := Matrix => (f, g) -> (
    map(target f, source f, rawDivisionAlgorithm(raw f, raw g, 0))
    )
```