GNU Modula-2 status, whole program optimisation and language interoperability

Gaius Mulley
<gaius@gnu.org>

Department of Computer Science
University of South Wales
CF37 1DL

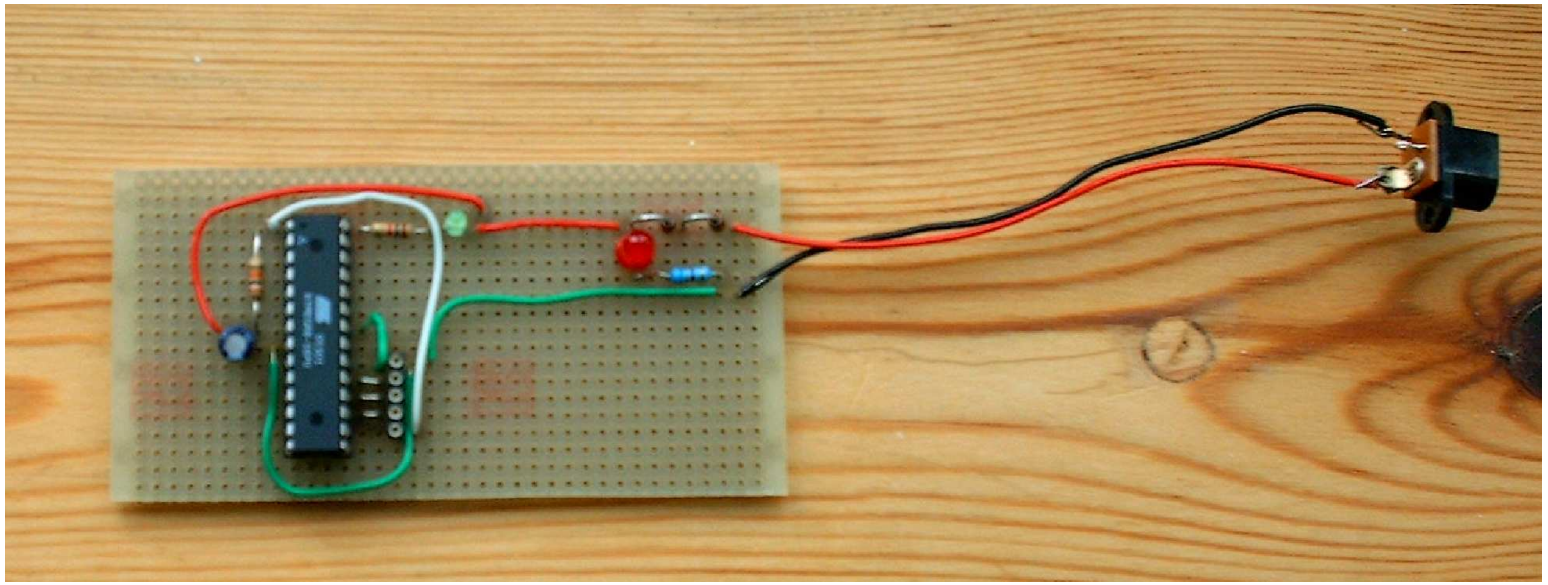
Why Modula-2 today?

- “source code which cannot port to another architecture will die”

  Mike Gancarz

- legacy code
  - academic
  - industrial
- embedded systems of today
  - low memory footprint, bit manipulation, memory mapped variables, coroutines, interrupt priorities
- great teaching language

Embedded systems and Modula-2



- the microprocessor is an ATmega328p
  - 32 KB of flash memory and 2 KB of RAM (£ 1.00)

About GNU Modula-2

- GNU Modula-2 supports PIM2, PIM3, PIM4 and ISO Modula-2
- PIM support was finished in 2006
- ISO libraries and language were completed in 2010
- dejagnu regression testsuite
  - 1670 individual tests
  - run multiple times with varying command line options
- the front end, libraries and testsuite are all GPL 3.0

History of GNU Modula-2

- development started in 1999 after receiving an email from `rms`
- initially borrowed much code from m2f (now renamed x86_m2) written in 1992 (PIM2 compiler)
- it also had a `-students` option, which performed semantic checking on legal but poor code:
  - declaring variables using keyword names, variables using same name different case in the same scope
  - typically trying to guide first year programming students to use better coding style

Current release

- the current stable release is `gcc-4.7.4+gm2-1.1.5.tar.gz` which is available from ⟨`http://www.nongnu.org/gm2/download.html`⟩

Current development

- focused on `gcc-5.2.0` (`gcc-5.3.0` and `trunc`).
- in particular in the last 9 months the bootstrapping has undergone a huge change
- `p2c` has been really useful over the past 24 years
  - `p2c` has been replaced by `mc`
- `mc` borrowed the parser and approx 25 implementation modules from `gm2`
- unlike `p2c`, `mc` understands full Modula-2 syntax and GNU Modula-2 extensions
  - no forward references required

Current development

- `p2c` required too many changes to enable it to generate `g++` compliant C
- `mc` was completed this week and it will be used to bootstrap `gm2`
- the rewritten modules in `mc` will be back-ported to `gm2`
- initiating a partial rewrite of `gm2`

Observations

- whenever a change occurs in GCC which effects a front end it nearly always results in a cleaner front end!
- the argument handling changes from a few years ago also caused tidying of the code base
- introduction of gimple interface in the 4.1.2 era tidied up code
- the move to building with `g++` caused, in part, `mc` to be constructed
- in turn this has shown the `gm2` code base from the 4.7.4 era to look a little old

Features

- GNU Modula-2 implements all dialects of Modula-2 PIM2, PIM3, PIM4, ISO and supporting libraries

- it tries to follow the GCC ethos of extensions:
  - `ASM` is exactly the same as C/C++ etc
  - `-fcpp` runs the C preprocessor over the source using the `cpp -lang-asm -traditional-cpp` option (aka GNU Fortran)

Extensions to allow access to C libraries

- **gcc-versionno/gcc/gm2/gm2-libs/libc.def**

```
DEFINITION MODULE FOR "C" libc ;

EXPORT UNQUALIFIED printf ;

PROCEDURE printf (format: ARRAY OF CHAR; ...) : [ INTEGER ] ;

END libc.
```

  - optional return value, strings passed by address and varargs

- when using `DEFINITION MODULE FOR "C"` pointers are mapped onto: `SYSTEM.ADDRESS`
  - `ARRAY OF type` is mapped onto `type *`
  - all other types are mapped onto their C counterparts

Extensions: access to assembly language

```
PROCEDURE Example (i, j: CARDINAL) : CARDINAL ;
VAR
   k: CARDINAL ;
BEGIN
   ASM VOLATILE ("movl %1,%eax; \
                  addl %2,%eax; movl %eax,%0"
                  : "=g" (k)           (* outputs *)
                  : "g" (i), "g" (j)   (* inputs  *)
                  : "eax") ;           (* we trash *)
   RETURN( k )
END Example ;
```

Extensions: access to GCC built-ins

- access to `alloca`, `memcpy`, `sin` and friends

- **gcc-versionno/gcc/gm2/gm2-libs/MathLib0.def**

```
DEFINITION MODULE MathLib0 ;

CONST
   pi  =3.14159265358979323846264338327950288419 72;
   exp1=2.71828182845904523536028747135266249775 72;

PROCEDURE __BUILTIN__ sqrt (x: REAL) : REAL ;
PROCEDURE __BUILTIN__ sin (x: REAL) : REAL ;
PROCEDURE __BUILTIN__ cos (x: REAL) : REAL ;
PROCEDURE tan (x: REAL) : REAL ;
```

Extensions: access to GCC built-ins

- **gcc-versionno/gcc/gm2/gm2-libs/MathLib0.mod**

```
IMPLEMENTATION MODULE MathLib0 ;

IMPORT cbuiltin;

PROCEDURE __ATTRIBUTE__ __BUILTIN__ ((__builtin_sqrt))
          sqrt (x: REAL): REAL;
BEGIN
   RETURN cbuiltin.sqrt (x)
END sqrt ;
```

Language extensions

- enhances a number of language features:
  - sets can be declared from any ordinal type
  - abstract data types are not restricted to a pointer type
  - types, variables, constants, may be declared in any order
    - technically not a language extension but `gm2` does not need forward declarations

Swig compatible

- `-fswig` generates a interface file, gm2 will examine the procedure parameters and attempt to work out the input/output parameters
- works reasonably well and in a couple of command lines will allow Python to call Modula-2
- can throw an exception in Modula-2 and catch it in Python (or another scripting language)

Mixed language support

- likewise a C++ module can throw an exception and Modula-2 can catch it

- `libtool` is supported, can be used to build shared libraries

Mixed language support

- **gcc-version/gcc/gm2/examples/pge/Make-file.in**

```
swig -outdir . -o pgeif_wrap.cxx -c++ -python $(srcdir)/pgeif.i
libtool --tag=CC --mode=compile g++ -g -c pgeif_wrap.cxx \
  -I/usr/include/python$(PYTHON_VERSION) -o pgeif_wrap.lo
gm2 -c -g -I$(SRC_PATH_PIM) -fmakelist $(srcdir)/pgeif.mod
gm2 -c -g -I$(SRC_PATH_PIM) -fmakeinit -fshared $(srcdir)/pgeif.mod
libtool --tag=CC --mode=compile g++ -g -c _m2_pgeif.cpp \
    -o _m2_pgeif.lo
libtool --tag=CC --mode=link gcc -g _m2_pgeif.lo \
    $(PGELIBSOURCES:%.mod=%.lo) \
    pgeif_wrap.lo buffers.lo \
   -L$(prefix)/lib64 \
   -rpath ‘pwd‘ -lgm2 -liso -lgcc -lstdc++ -lpth -lc \
   -lm -o libpgeif.la
cp .libs/libpgeif.so _pgeif.so
```

- `libtool` is also used when multiarch building the PIM, ISO, LogiTech compatible, Ulm, Min, Coroutines libraries (`gcc-version/libgm2`)

Data type correspondence with C/C++

GNU Modula-2	GNU C
INTEGER	int
LONGINT	long long int
SHORTINT	short int
CARDINAL	unsigned int
LONGCARD	long long unsigned int
SHORTCARD	short unsigned int
BITSET	unsigned int
BOOLEAN	int
REAL	double
LONGREAL	long double
SHORTREAL	float
CHAR	char
SHORTCOMPLEX	complex float
COMPLEX	complex double
LONGCOMPLEX	complex long double

Embedded system support

- ASM keyword supported as per GNU C
- options to turn off exception handling and link against minimal runtime libraries
-

```
$ avr-gm2 -mmcu=atmega328 -g -Os -fno-exceptions -O2 -c \
  flashled328.mod -flibs=min
$ avr-gm2 -mmcu=atmega328 -g -Os -fno-exceptions -O2 \
  -fno-pth -fonlylink flashled328.mod -flibs=min -o flashled328.elf

$ avr-size flashled328.elf
  text  data    bss    dec    hex  filename
  370      0      0    370    172  flashled328.elf
```

- users can specify the initialisation order of modules which are statically calculated via import lists and dependancies

Embedded system support

- request the initialisation list, modify it and continue to use this during the link
- used to create the `main` scaffold (in either C or C++)

Implementing unbounded arrays using trees

```
PROCEDURE concat (VAR a: ARRAY OF CHAR;
                  b: ARRAY OF CHAR) ;
VAR
   i: CARDINAL ;
BEGIN
   i := HIGH(a) ;
   a[0] := 'a' ;
   a[i] := 'b'
```

- `gm2` implements unbounded arrays by internally creating an unbounded type:

```
unbounded = RECORD
               _arrayAddress: POINTER TO arrayType ;
               _arrayHigh   : CARDINAL ;
            END ;
```

Implementing unbounded arrays using trees

- callee must save contents of an array for each non VAR parameter
  - achieved using `memcpy` and `alloca`

- **gcc-version/gcc/gm2/gm2-compiler/M2GenGCC.mod:MakeCoptAndUse**

```
High := GetSizeOfHighFromUnbounded(tokenno, param) ;
Addr := GetAddressOfUnbounded(param) ;
Type := Mod2Gcc(GetType(param)) ;

NewArray := BuiltinAlloca(location, High) ;
NewArray := BuiltinMemcpy(location, NewArray, Addr, High) ;

(* now assign  param.Addr := ADR(NewArray).  *)
BuildAssignmentTree(location,
                    BuildComponentRef(Mod2Gcc(param),
                    Mod2Gcc(GetUnboundedAddressOffset(UnboundedType))),
                    NewArray)
```

- due to in-lining of `alloca` and `memcpy` good code is emitted

The GNU Modula-2 front end is written in C and Modula-2

- uses `flex` and a Modula-2 parser generator to construct a top down recursive descent parser with error recovery
  - uses flex to build a dynamic buffer of all source tokens

- a six pass compiler which uses double entry book keeping
  - lexical analysis, parsing, modules and associated filenames
  - scopes, enumerated types, imports and exports
  - constants and types
  - [aggregate constants]
  - quadruple generation
  - gcc tree generation

The GNU Modula-2 front end is written in C and Modula-2

- each front end Modula-2 symbol table entry is translated into `trees` and finally the quadruples are translated into `trees`
  - front end ensures that only legal symbols and legal source are ever passed to GCC

Compiler options

- `-Wstudents`
  - checks for bad programming style
- `-Wpedantic`
  - reject nested `WITH` statements referring to the same record type
  - reject code which uses a `FOR` loop indice outside the loop without being reset
- `-Wpedantic-param-names`
  - check definition module parameter names match their implementation counterparts
- `-fextended-opaque`
  - opaque types can be implemented as any type
  - also enables full type declaration to the debugger

Compiler options

- `-fsoft-check-all`
  - turns on `-fnil`, `-frange`, `-findex`, `-fwholediv`, `-fcase` and `-freturn`.
- `-fno-exceptions` ensure no references are generated to the exception libraries
- `-fiso`, `-fpim2`, `-fpim3`, `-fpim4`
- for the complete list visit ⟨`http://www.nongnu.org/gm2/homepage.html`⟩

Whole program optimisation

- credit to David Edelsohn (at the GCC Cauldron 2014 in Cambridge)
- interesting to note that it took about 3 weeks to implement whole program improvement, should have done this long ago as the improvement was substantial for the minimal coding effort
- all following timings were taken on Debian Jessie GNU/Linux AMD FX(tm)-8350 Eight-Core Processor, 4GHz, using gm2-1.1.5, gm2 (GCC) 4.7.4

Whole program optimisation

- maze generator compiled with `-O3`

time ./map > m1	
component	seconds
real	2.511
user	2.508

- compiled with `-O3 -fm2-whole-program`

time ./map.whole > m1.whole	
real	1.415
user	1.412

- 44% reduction in user time

Whole program optimisation

- physics game engine compiled with `-O3`

time ./pge > /dev/null	
real	10.330
user	6.880
sys	3.440

- physics game engine compiled with `-O3 -fm2-whole-program`

time ./pge-whole > /dev/null	
real	8.875
user	5.428
sys	3.440

- 22% reduction in user time, 14% reduction in real time

Immediate future

- complete graft onto `gcc-5.2.0`
- and track `trunc`
- current open branches in `gm2` are:
  - `gcc-4.1.2`, `gcc-4.7.4`, `gcc-5.2.0`, `gcc-5.3.0`, and `trunc`

Future work

- implement `-fwholevalue`
  - detect integer overflow at runtime
- prepare the source tree to implement `M2R10`
  - 2010 revision of Modula-2 which is being standardised
- ⟨`http://modula-2.info/m2r10/pmwiki.php/Spec/LanguageReport`⟩
- ⟨`http://modula-2.info/m2r10/pmwiki.php/Project/FAQ`⟩

GNU Modula-2 code comparison on the gcc-4.7.4 branch

Category under gcc-4.7.4	Lines (wc -l)
gcc/*.[ch]	787,368
gcc/fortran/*.[ch]	155,499
ada/*.{ads,adb}	1,031,376
gcc/gm2/*/*.{def,mod}	157,189
gcc/gm2/*/*.[ch]	27257

Conclusions and acknowledgements

- double entry book keeping is pragmatic and works well
- whole program optimisation should not be an after thought for a GCC front end
  - the gains are too good and coding effort is small
- huge thanks to
  - all GCC developers for great target architecture coverage
  - all `gm2` users for all their feedback, bug reports and test programs