

What's GNU With You?





GNU Binary Utilities

Year in Digest

- Sep/2014 - GNU libiberty (-liberty) supports D2 demangling
 - Considered feature complete (nm phobos_unittest)

- All programs that use demangling part of library D aware.
 - addr2line -C dlang
 - nm -C dlang
 - objdump -C dlang
 - c++filt -s dlang
 - ld --demangle=dlang (or gold)
 - gprof --demangle=dlang
 - gdb (automatically used if DW_LANG_D)

What's happened since?

- ❑ Oct/2014: Removed use of strtold, no support for many platforms (Solaris 9)
- ❑ May/2015: Removed use of strtod, some don't accept hexadecimals (Solaris 9 ... again)
- ❑ Aug/2015: Add return parameters and attributes, cent/ucnt types
- ❑ Jan/2016: Add extern(Objective-C)
- ❑ Apr/2017: Add scope attributes

Present and Future

- D name mangling ABI now better documented (Mostly Rainer)

I've also updated the grammar (sic) to not need the special token QFT. This now reveals the conflict for 'V' and another one on 'M'. -- Rainer

- Support for mangle back references (Major ABI change)
- Support multiple format styles (types, params, attributes)
- Work may become part of core.demangle



GNU Project Debugger

GNU Project Debugger

```
ibuclaw@galago:~/src $ gdb a.out
GNU gdb (GDB) 8.0.50.20170426-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...done.
(gdb) start
Temporary breakpoint 1 at 0x403302: file lisp.d, line 13.
Starting program: /home/ibuclaw/src/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Temporary breakpoint 1, D main () at lisp.d:13
13      auto interp = new Interpreter("interp> ");
(gdb) list
8      import std.conv;
9      import std.exception;
10
11     void main()
12     {
13         auto interp = new Interpreter("interp> ");
14         assert(interp.unittests());
15         interp.go();
16     }
17
(gdb) █
```


GNU Project Debugger

```
lisp.d
1 // Scheme Interpreter in 90 lines of D (not counting lines after the first 90).
2 // Copyright (c) 2011 Iain Buclaw.
3
4 import std.stdio;
5 import std.array;
6 import std.string;
7 import std.bigint;
8 import std.conv;
9 import std.exception;
10
11 void main()
12 {
13     auto interp = new Interpreter("interp> ");
14     assert(interp.unittests());
15     interp.go();
16 }
17
18 ////////// Cell //////////
19
20 enum CellType
21 {
22     Symbol,
23     Number,
24     List,
25     Proc,
26     Lambda,
27 }
28
29 // a variant that can hold any kind of lisp value
30 struct Cell
31 {
32     alias Cell function(Cell[]) ProcType;
33
34     CellType type;
35     string val;
36     Cell[] list;
```

multi-thre Thread 0x7ffff7fca5 In: D main
(gdb)

L13 PC: 0x403302

GNU Project Debugger

```
1 // Scheme Interpreter in 90 lines of D (not counting lines after the first 90).
2 // Copyright (c) 2011 Iain Buclaw.
3
4 import std.stdio;
5 import std.array;
6 import std.string;
7 import std.bigint;
8 import std.conv;
9 import std.exception;
10
11 void main()
12 {
13     → auto interp = new Interpreter("interp> ");
14     assert(interp.unittests());
15     interp.go();
16 }
17
18 ////////// Cell //////////
19
20 enum CellType
21 {
22     Symbol,
23     Number,
24     List,
25     Proc,
26     Lambda,
27 }
28
29 // a variant that can hold any kind of lisp value
30 struct Cell
31 {
32     alias Cell function(Cell[]) ProcType;
33
34     CellType type;
35     string val;
36     Cell[] list;
37     ProcType proc;
38     Environment env = null;
39 }
40
41 /home/ibuclaw/src/lisp.d
(gdb)
```

Year in Digest

- Jan/2014: D language support given a revamp
 - Aware of 'D main' (start)
 - Primitive types made GDB built-ins
 - Hooked in new demangling capabilities
- Feb/2014: New D expression parser (YACC)
 - Expressions match D grammar until PrimaryExpression
 - Types grammar incomplete
 - Recognize many common keywords
 - Parse integer and floating pointer notations
 - Literals (true, false, null)

What's happened since?

- Aug/2015: Non-local symbol lookup
 - DW_TAG_imported_module
 - DW_TAG_imported_declaration
- Aug/2015: Switched to eagerly resolving symbols (in yylex)
- Oct/2015: Started adding simple property expressions (.sizeof)
- Feb/2016: Various bug fixes
- Mar/2017: Some more small bug fixes (just in time for gdb-8.0)

Where's the rest?

```
<andrew>
(gdb) p url
$1 = "http://www.google.com/path"
(gdb) p url[0]
Invalid binary operation specified.
(gdb) p *url
Structure has no component named operator*.
(gdb) p *(url.ptr)
Internal error: 'this' is not an aggregate
(gdb) ptype url
type = struct _Array_char {
    unsigned long length;
    char *ptr;
}
(gdb) p url[0 .. 1]
cannot take slice of non-array
<ibuclaw> Please raise bugs though for any missing niceties. :)
<andrew> ok...if i must :P
<ibuclaw> You might have a point there with better handling of dynamic arrays
<andrew> you think url[0] should work?
<ibuclaw> Yes, and url[x..y]
<ibuclaw> The debugger should at least support most simple built-in operations.
<andrew> i have opened issues for [n] and [n .. o]
```

Where's the rest?

<**ibuclaw**> Apparently I implemented D array slicing back in 2014.

<**andrew**> and someone broke it?

<**ibuclaw**> No


<**ibuclaw**> I never committed it

Near or Far Future


- Support printing D types (module, ulong, __vector)
- Finish off the D grammar
- Support all operations that don't have external dependencies
 - Array operations
 - typeid()
 - Special casts (delegates, arrays, dynamic_cast)
- Function call overloading
- On-demand D compilation and code injection (one can dream...)



GNU D Compiler



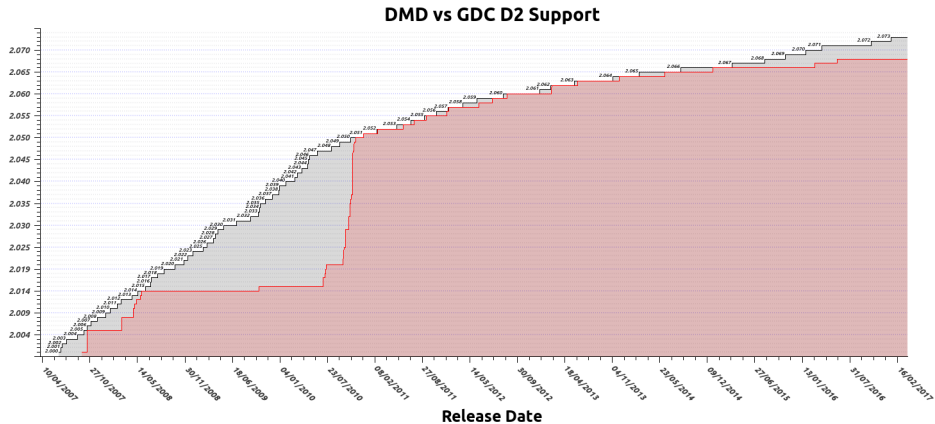
*First of all, thank you for your tremendous work on GDC!
Fellow developers and me were also pretty stunned by you
maintaining a quite large amount of different GDC versions in
parallel without a huge team - that's some impressive work!
-- Matthias, Debian Maintainer (appstream, dustmite, vibe.d)*



What is the thing that's blocking GDCs GCC inclusion? Just manpower? It would probably be awesome to have a summary blogpost or similar on the state of GDC, that could potentially also attract volunteers.

-- Matthias, Debian Maintainer (appstream, dustmite, vibe.d)

A Brief History



GCC Integration

- Oct/2011: Digital Mars assigns past, present and future changes to the GNU D Compiler.
- Oct/2011: First submission attempt - abstract.
- Aug/2012: Second submission attempt - technical

GCC Integration

□ Positive:

- In general we welcome contributions like this. The biggest problem in the past has always been continued maintainership.
- There is a list of most of the requirements for a new frontend at <http://gcc.gnu.org/onlinedocs/gccint/Front-End.html>.
- The merge of Go could be a good example to follow. It was finally committed here, so be patient and persevere.

□ Negative:

- Does D really require a new calling convention?
- Also does it really require naked support?
- Some parts are duplication of other GCC frontends - think about refactoring to share it.
- All significant contributors to need to have papers on file at FSF.
- Original GDC author MIA since 2007.


GCC Integration

□ Negative:

- How did you test this? You include poisoned headers
- Lacking testsuite (DejaGNU)
- Project directories incorrectly structured
- Many functions have no leading comment
- GNU coding standards are not followed either
- DMD frontend missing copyright notices in many sources
- DMD frontend has notice in many files:

This file has been patched from the original DMD distribution to work with the GDC compiler

- DMD license is GPLv2 - we need an explicit notice (approved by the copyright holder) saying that **any later version** may be used
- What is d/d-asm-i386.h for? It looks like i386 is a special case throughout the frontend
- Why is there support for prior versions of GCC?
- Macros, macros, everywhere (V1, V2) - why?
- Target and host specific code, everywhere - why?



What happened to the effort to merge GNU D? The discussion thread last October seemed to end without any resolution. I wanted to make sure that the contribution process had not discouraged you. We would like to include GDC when it is ready.

-- David, GCC Maintainer (AIX, SPU, RS6000 ports)

Battle Plan

- Remove D1 support
- Only support one GCC version - trunk/master
 - ▣ GCC releases should go into separate branches.
- Drop everything in the language that is tied to a target.
 - ▣ D inline assembler
 - ▣ D calling convention
- Make no assumptions based on target
 - ▣ Ask: Do I need to prefix symbols?
 - ▣ Not: Am I compiling for OSX?
- DMD Frontend must be shared/unmodified

Developments

- Jan/2014:
 - Addressed points raised on D frontend with the core community
 - Introduced 'Target' hook to remove many GDC-specific changes
 - GCC sources switched to C++, many problems are now irrelevant
 - Got very close to a “common” frontend with DMD

- Feb/2014: DMD frontend replaces glue interfaces with visitors
 - DMD migrated in straight-forward conversions

- Sep/2014: Received patches via private email to migrate GDC

- I had another idea in mind...

The Visitor Problem

Old Interface:

□ DMD

- `ctype ::toCtype`
- `Symbol ::toSymbol`
- `elem ::toElem`
- `dt_t ::toDt`
- `IRState ::toIR`
- `void ::toObjFile`

□ GDC

- `ctype ::toCtype (typedef tree)`
- `Symbol ::toSymbol`
- `elem ::toElem (typedef tree)`
- `dt_t ::toDt (typedef tree)`
- `IRState ::toIR`
- `void ::toObjFile`

The Visitor Problem

New Interface:

□ DMD

- ctype ToCtypeVisitor
- Symbol ToSymbol
- elem ToElemVisitor
- dt_t xxxToDt
- IRState S2irVisitor
- void ToObjFile

□ GDC

- tree TypeVisitor
- tree get_symbol_decl ()
- tree ExprVisitor
- [deleted]
- void IRVisitor
- void DeclVisitor

The Visitor Problem



Deciding to go down this route was not a straightforward change. More a total rewrite of the GDC internals.

More Developments

- Aug/2015: Half of old interfaces had been removed or upgraded
 - Types
 - Expressions
 - Statements

- Aug/2015: DMD replaces C++ frontend with D
 - Introduced many interoperability regressions
 - As of writing, most are solved - still many unknowns

More Developments

- Jun/2016: Updated to DMD 2.068.2 FE branch
- Aug/2016: Phobos and Druntime built as shared library
 - ▣ Passing unittests and testsuite.
- Oct/2016: Original GDC author assigns past changes to FSF
 - ▣ Legally, we are now unblocked!
- Dec/2016: Scaffolding for old glue interface being torn down.
 - ▣ Remove stubs for Symbol, IRState

The DDMD Problem

- DMD itself is moving along at a fast speed and neither LDC or GDC are keeping up
 - Regressions are noticed only after a release - makes having a “vanilla” frontend impossible
- There are upstream fixes to remove need for frontend changes, but only in D implementation
 - Some have dependencies on other changes too
- At some point GDC should switch to DDMD frontend
 - Likely many changes that affect codegen pass
 - Meaning maintain two branches, keeping both in sync!
- Can't go on updating once every release
 - Need a “stable” target to base GDC on

Even More Development

- Dec/2016: Sync with last C++ version of 2.069-development
 - Merge C++ headers with upstream/stable
 - Backport fixes and features that are related
 - extern(C++, struct)
 - Fix bug 33/34
 - Support DIP25/DIP1000
- Feb/2017: Backport DMD regression fixes from all versions up to 2.071.2
- Mar/2017: Updated Phobos to 2.071.2
- Apr/2017: Headers in sync with DMD stable

Current Status

- ❑ GDC now in freeze for documentation/refactor mode
- ❑ Fixing the coding style once and for all
- ❑ Documenting every single function - referencing D spec where applicable
- ❑ Grouping common codegen routines into separate files
- ❑ GCC-8 stage1 opened (20th April 2017)
 - ❑ GCC-7 release (due 2nd May 2017)

While Submission is Happening

- More extensive documentation to come.
 - The gcc.builtins module
 - GDC compiler extensions (extended assembler)
 - Predefined version identifiers used by GDC
 - Configure time options
 - Build and bundle in documentation for Phobos library
 - Interoperability with C/C++ (maybe)

- Improve our Platform and Architecture support

- Build up a testing infrastructure (ARMv7, ARMv8, x86)



Hitchhikers Guide to Porting GDC



And That's It!

