# An Algol 68 Front-End for GCC (Fosdem redux edition)

José E. Marchesi
jemarch@gnu.org
https://jemarch.net

2026

# Outline

Algol 68

The GNU Algol 68 Working Group

The Algol 68 GCC Front-End

GNU extensions to Algol 68

Other projects

**Algol 68**

**The GNU Algol 68 Working Group**

**The Algol 68 GCC Front-End**

**GNU extensions to Algol 68**

**Other projects**

# Algol 68... what gives?

We better address this once and for all...

- ALGOL 60 was a very important pioneering language, nowadays superseded in every aspect. It remains only as a historical curiosity.
- Algol 68 is a modern, powerful and safe programming language that never had successors and that is comparable and often superior in many aspects to programming languages widely in use today.
- Writing programs in Algol 68 is fun, productive and an unique experience, almost addictive.

Clearly we need modern tools and documentation :)

# An Algol 68 timeline

- 1964 : ALGOL X
- 1965 - 1969 : Development
  - 1965 : Wirth, Seegmüller and Van Wijngaarden [MR76].
  - 1966 : Van Wijngaarden commissioned to make the Report.
  - 1967 : Draft [MR88].
  - 1968 : Drafts [MR93] [MR95]. Accepted Report [MR100].
  - 1969 : Final Report [MR101].
- 1970 - 1974 : Revision
  - 1970 : First implementation - ALGOL68-R
  - 1974 : Revised Report [RR]
- 1975 - 1979 : Active support
  - 1976 : Standard Hardware Representation.
  - 1978 : Modules and separated compilation facility.
- 1980 - 2025 : Dark ages
- 2025 : GNU work starts

Algol 68

**The GNU Algol 68 Working Group**

The Algol 68 GCC Front-End

GNU extensions to Algol 68

Other projects

# The GNU Algol 68 Working Group

https://algol68-lang.org

Informal group of hackers with a mission:

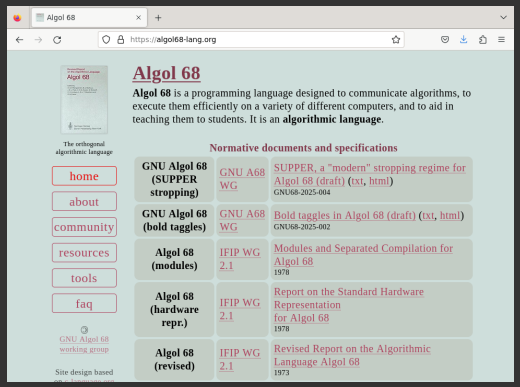
- To bring Algol 68 to the first line of programming and dispel the abundant myth and FUD surrounding it,
- to provide modern implementations of the language, well integrated in today's operating systems and computers,
- to produce documentation and other material to help people to learn the language, and
- to maintain, extend and evolve the language with the rigor and seriousness that it deserves and demands.

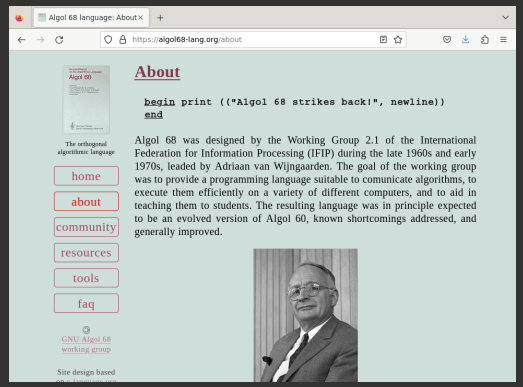
Not officially associated with the IFIP WG 2.1, which is still in charge of the language, at least theoretically. Contacts have been established.

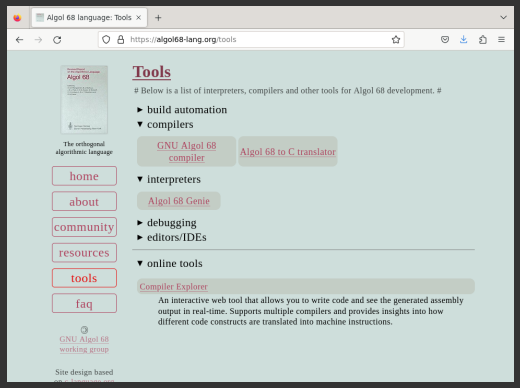
# GNU A68 WG - Projects and activities

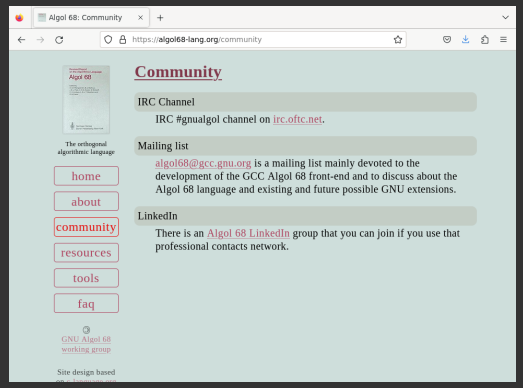
- Documentation
  - Algol 68 homepage.
  - The Algol 68 Jargon File.
- Language
  - GNU Algol 68 extensions.
  - Libraries and preludes.
- Editors and IDEs
  - Emacs a68-mode.
  - Emacs vw-mode.
- Toolchain
  - GCC front-end.
  - automake.
  - autoconf.
  - DWARF support.
  - GDB.
  - Compiler Explorer.
- Community
  - Mailing list.
  - IRC channel.
  - Conferences.

# A homepage for Algol 68




- https://algol68-lang.org
- https://git.sr.ht/~jemarch/algol68-lang

# The Algol 68 mailing list

algol68@gcc.gnu.org

- Development of the GCC Algol 68 front-end.
- Algol 68 programming.
- Language extensions.
- VW-Grammars.
- Working group related topics.

https://gcc.gnu.org/mailman/listinfo/algol68
https://gcc.gnu.org/pipermail/algol68

# The GNU Algol 68 IRC channel

irc.oftc.net/#gnualgol

Friendly real-time discussion of all topics related to Algol 68.

Algol 68

The GNU Algol 68 Working Group

**The Algol 68 GCC Front-End**

GNU extensions to Algol 68

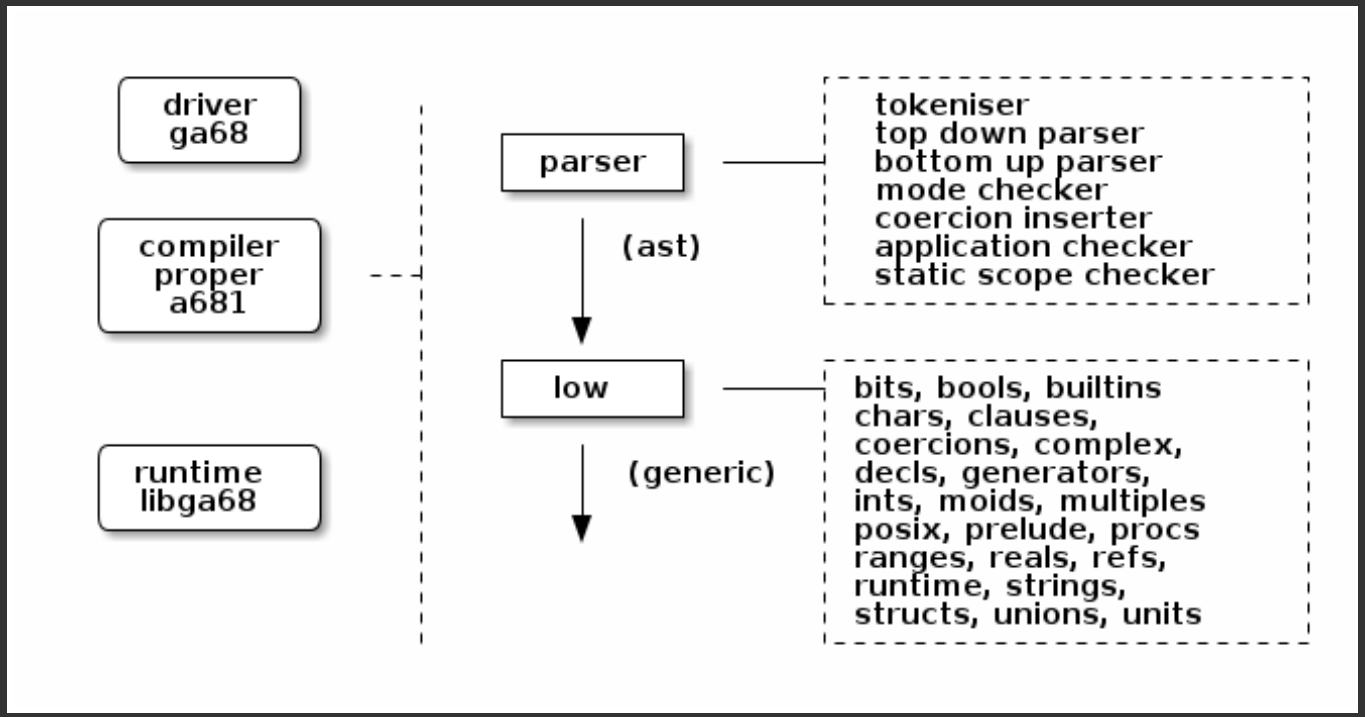
Other projects

# ga68 - the GNU Algol 68 compiler

```
$ ga68 --version
ga68 (GCC) 16.0.1 20260130 (experimental)
Copyright (C) 2026 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- Implements the full language plus GNU extensions -std=algol68 -std=gnu68.
- User manual and internals manual.
- Formal description of accepted language: `ga68.vw`
- **char** is an UTF-32 UTF codepoint.
- Already available in some distros
  - Debian, Ubuntu: ga68-15

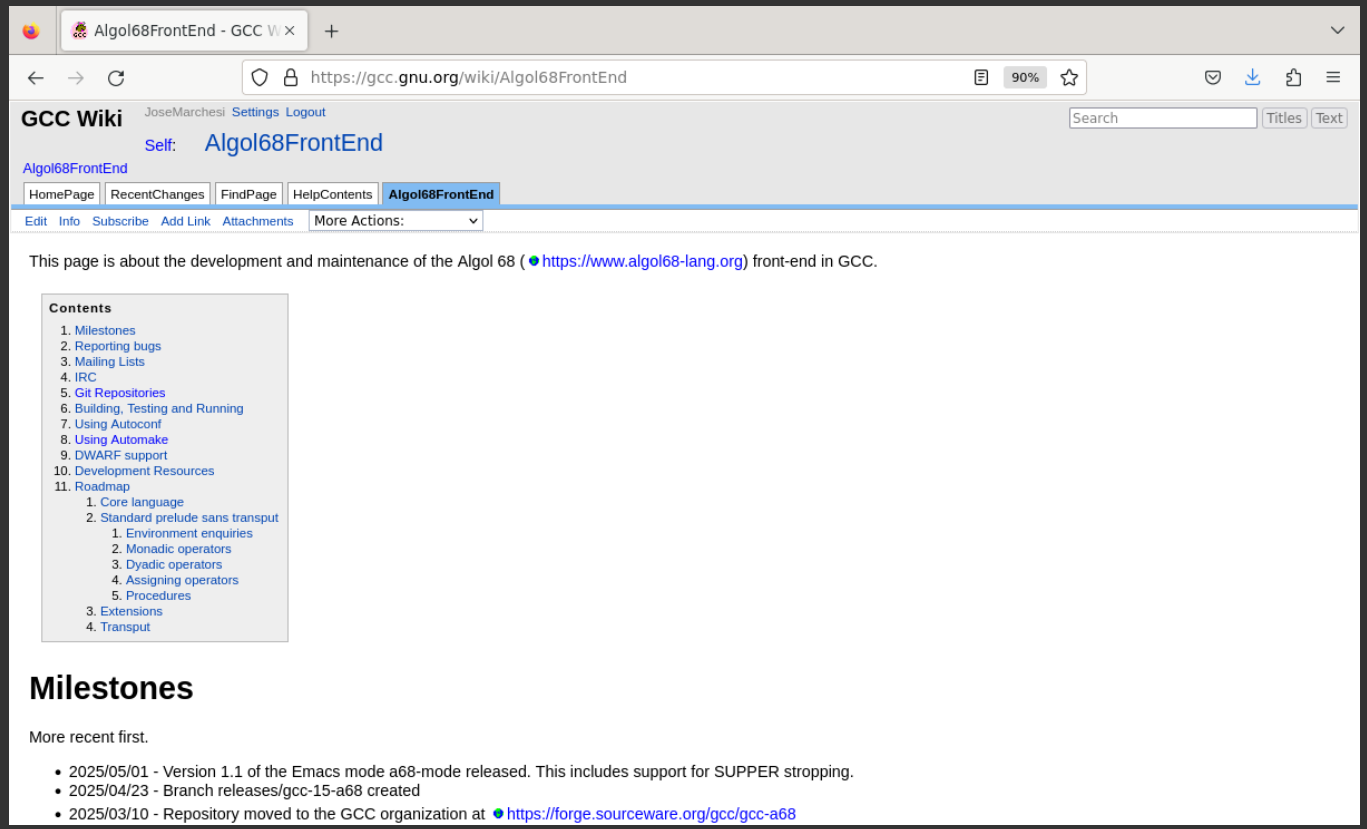
# ga68 - architecture



# ga68 - what’s up with development

- First patch series sent to gcc-patches January 2025.
- Off-tree for a while.
- Integrated in trunk in November 2025.
- Modules system implemented.
- Support for FFI implemented.

# ga68 - homepage




- https://gcc.gnu.org/wiki/Algol68FrontEnd
- Status of the port and instructions on how to contribute.

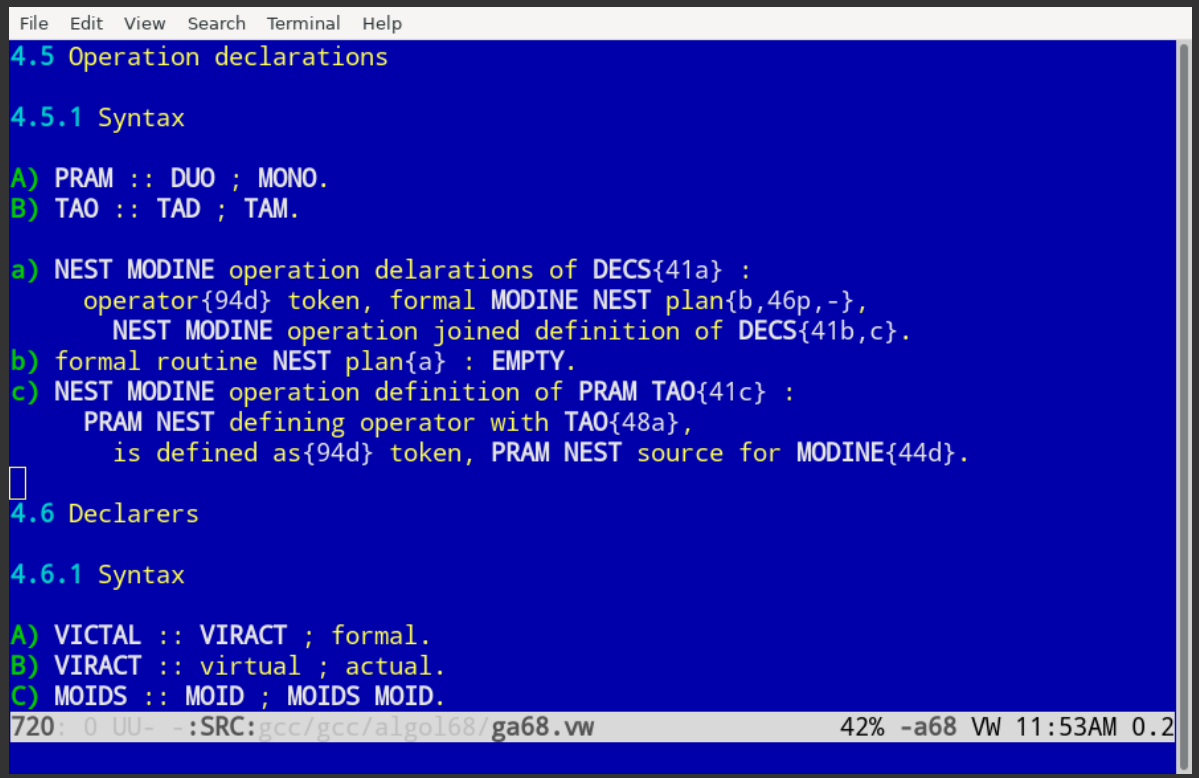
# ga68 - testing the front-end

- algol68/compile/compile.exp
- algol68/compile/mcts/compile.exp
- algol68/execute/execute.exp
- algol68/execute/mcgt/execute.exp
- algol68/execute/mcts/execute.exp

```
                === algol68 Summary ===

# of expected passes            10755
```

# ga68 - description of the compiled language




- gcc/algol68/ga68.vw.
- Formal description of the language and GNU extensions.
- Always up to date.

Algol 68

The GNU Algol 68 Working Group

The Algol 68 GCC Front-End

**GNU extensions to Algol 68**

Other projects

# GNU extensions to Algol 68 (I)

The core language is perfect and eternal in its orthogonal frame, but there is still plenty of room for improvement:

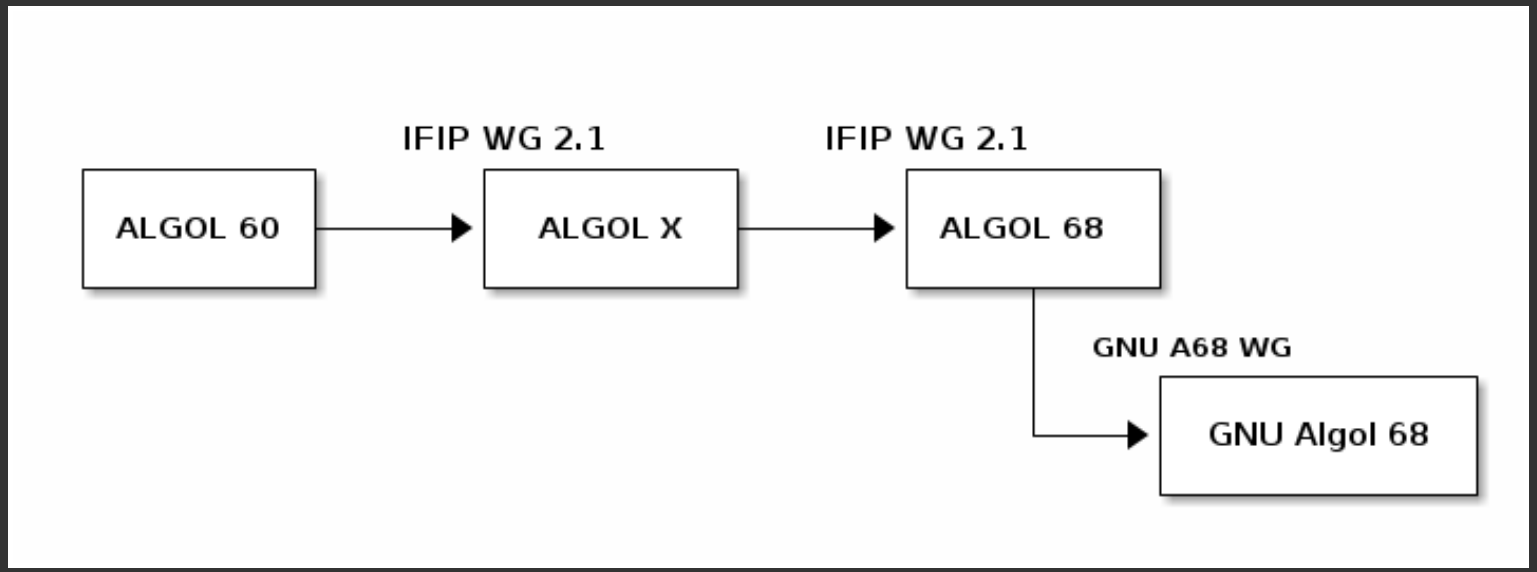
- **Expansion** via new source constructs.
- **Modernization** of program representation, transput (input/output), preludes/libraries, system integration, etc.

**[RR page 8]**
“The Working Group has decided that this present revision should be “the final definition of the language ALGOL 68” [...]”

**[RR page 52]**
“[...] a superlanguage of ALGOL 68 might be defined by additions to the syntax, semantics or standard-prelude, so as to improve efficiency or to permit the solution of problems not readily amenable to ALGOL 68.”

# GNU extensions to Algol 68 (II)

The set of GNU extensions conforms **GNU Algol 68**, a strict superlanguage of Algol 68 developed and maintained by the GNU Algol 68 Working Group.




| | | |
|---|---|---|
| **GNU68-2025-001** | An unsafe clause for Algol 68. | draft |
| **GNU68-2025-002** | Bold taggles in Algol 68. | draft |
| **GNU68-2025-003** | Short-circuit logical pseudo-operators for Algol 68. | draft |
| **GNU68-2025-004** | SUPPER, a modern stropping regime for Algol 68. | draft |
| **GNU68-2025-???** | Separated compilation and modules for Algol 68. | wip |
| **GNU68-2025-???** | Modals, generic programming for Algol 68. | wip |

# GNU extensions to Algol 68 (III)

- Extensions are prepared and hosted in `https://git.sr.ht/~jemarch/gnu68`
- Eventually published in `https://algol68-lang.org`
- Each extension is expected to provide:
  - A notice based on IFIP's TUM-10.
  - An **informal description**.
  - A **formal description**, in the form of an addendum to some normative document, such as the Revised Report or the Standard Hardware Representation.
- GNU extensions are enabled by **`-std=gnu68`** in GCC.

# GNU extensions to Algol 68 (IV)

Let's take a look with some detail at some examples of GNU extensions to Algol 68, already implemented in GCC:

- The modernization of the program representation via the introduction of a new stropping regime.
- Nestable comments.
- The addition of a new source construct, an **unsafe** clause.
- The addition of a POSIX-like standard prelude (library).

# Representation of Algol 68 programs

- The strict language.
  - Terminal productions of production trees, i.e. a sequence of symbols:

```
letter f symbol, letter o symbol, letter o symbol,
becomes symbol, digit one symbol, go on symbol
```

- Representation languages.
  - Provide representations for symbols in the strict language.
  - Can be classified depending on their purpose:

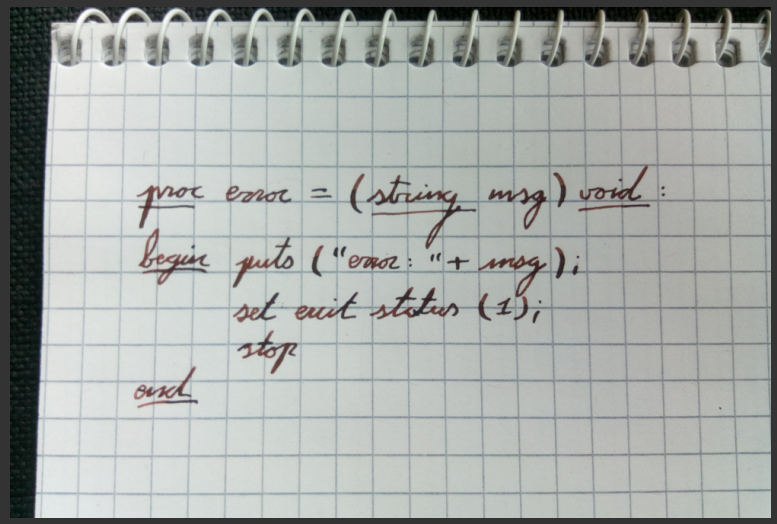
| | **read by human** | **read by program** |
|---|---|---|
| **written by human** | publication languages | programming languages |
| **written by program** | | hardware languages |

- Reference language
  - Recommended representation for the symbols, recognizable "without further elucidation":

| strict language | reference language |
|---|---|
| bold begin symbol | **begin** |
| bold letter a symbol | **a** |
| letter f symbol | f |
| becomes symbol | := |
| go on symbol | ; |

# Publication languages

```
proc error = (string msg) void:
begin puts ("error: " + msg);
      set exit status (1);
      stop
end
```




- In printed paper or electronic publications bold words are usually represented by actual bold glyphs, exploiting rich typographic capabilities.
- In manuscripts and napkins bold words are often represented with underlining.

# Hardware Languages

```
<prog>
 <tag>error</tag>
 <equals/>
 <open/>
   <btag>string</btag>
   <tag>msg</tag>
 <close/>
 ...
</prog>
```




```
[{ "tag" : "error" },
 "equals",
 "open",
 { "btag" : "string" },
 { "tag" : "msg" },
 "close",

 ...
]
```

```
71 00 00 00 03 30 00 14 00 00 00 02
74 61 67 00 06 00 00 00 65 72 72 6f
72 00 00 02 31 00 07 00 00 00 65 71
75 61 6c 73 00 02 32 00 05 00 00 00
6f 70 65 6e 00 03 33 00 16 00 00 00
02 62 74 61 67 00 07 00 00 00 73 74
72 69 6e 67 00 00 03 34 00 12 00 00
00 02 74 61 67 00 04 00 00 00 6d 73
...
```

# Programming languages

- Problem: how to distinguish `foo` and **foo**?
- Stropping is a technique used to explicitly mark bold words in program texts without resorting to rich typography.
- **Not** the same than coding conventions.
- Different stropping regimes have been used for Algol 68 at different times, based on the capabilities of available installations.

Algol 68 stropping regimes

| | |
|---|---|
| Classic pre-standard | QUOTE stropping |
| Classic standard | POINT stropping, RES stropping, UPPER stropping |
| GNU | SUPPER stropping |

# Classic QUOTE stropping

```
'PROC' FOR EACH ELEM = ('JSONVAL' VAL,
                        'PROC'('REF''JSONVAL')'VOID' H) 'VOID':
  'CASE' VAL
  'IN' ('REF''JSONARR' A):
       ('REF''JSONELM' ELEM := ELEMENTS 'OF' A;
        'WHILE' ELEM :/=: JSON NO ELM
        'DO' H (VALUE 'OF' ELEM);
           ELEM := NEXT 'OF' ELEM
        'OD')
  'ESAC'
```

- The original “stropping”, originating in ALGOL 60.
- Suitable for single-alphabet installations.
- Blank characters are generally superfluous.
- Predates the Standard Hardware Representation.

# Classic POINT stropping

```
.PROC FOR EACH ELEM = (.JSONVAL VAL,
                       .PROC(.REF .JSONVAL).VOID H) .VOID:
  .CASE VAL
  .IN (.REF .JSONARR A):
        (.REF .JSONELM ELEM := ELEMENTS .OF A;
         .WHILE ELEM :/=: JSON NO ELM
         .DO H (VALUE .OF ELEM);
            ELEM := NEXT .OF ELEM
         .OD)
  .ESAC
```

- Fundamental standard stropping regime.
- Bold words are preceded by a point character. Requires disjunctors.
- Tags are composed of taggles, each taggle optionally finished by an underscore, maybe separated by blanks.

# Classic RES stropping

```
PROC FOR_EACH_ELEM = (.JSONVAL VAL,
                      PROC(REF .JSONVAL)VOID H) VOID:
   CASE VAL
   IN (REF .JSONARR A):
         (REF .JSONELM ELEM := ELEMENTS OF A;
          WHILE ELEM :/=: JSON NO ELM
          DO H (VALUE OF ELEM);
             ELEM := NEXT OF ELEM
          OD)
   ESAC
```

- Bold words listed in [RR 9.4.1] become reserved words.
- Explicit POINT stropping still necessary for user-defined mode indications and operator indications.
- Taggles in identifiers/tags are sometimes adjacent to underscores to avoid conflicts with reserved words: `IF` vs `IF_`.

# Classic UPPER stropping

```
PROC for each elem = (JSONVAL val,
                      PROC(REF JSONVAL)VOID h) VOID:
   CASE val
   IN (REF JSONARR a):
         (REF JSONELM elem := elements OF a;
          WHILE elem :/=: json no elm
          DO h (value OF elem);
             elem := next OF elem
          OD)
   ESAC
```

- Exploits installations with two alphabets: upper- and lower-case.
- There are no upper-case digits, so POINT stropping is still necessary.
- Supported by GCC, `-fstropping=upper`.

# GNU extension - Bold taggles

```
PROC for each elem = (JSON_VAL val,
                      PROC(REF JSON_VAL)VOID h) VOID:
   CASE val
   IN (REF JSON_ARR a):
         (REF JSON_ELM elem := elements OF a;
          WHILE elem :/=: json no elm
          DO h (value OF elem);
             elem := next OF elem
          OD)
   ESAC
```

- GNU extension GNU68-2025-002.
- Bold words re-defined as composed of taggles.
- Makes mode indications more readable in all stropping regimes.
- Supported by GCC, `-fstd=gnu68`.

# GNU SUPPER stropping (I)

```
proc for_each_elem = (JSON_Val val,
                      proc(ref JSON_Val)void h) void:
   case val
   in (ref JSON_Arr a):
         (ref JSON_Elm elem := elements of a;
          while elem :/=: json_no_elm
          do h (value of elem);
             elem := next of elem
          od)
   esac
```

- GNU extension GNU68-2025-004.
- Advantages of both RES and UPPER. Fully expressive: no more need for POINT stropping. No blanks in identifiers.
- Adapts better to modern systems and taste.
- Supported by GCC, `-fstropping=supper` (default).

# GNU SUPPER stropping (II)

## Formal description - addendum to SHR

```
3.5.4 SUPPER Stropping

Bold words.

- A bold word is written as a sequence of {one or more} taggles
  whose worthy letters and digits correspond, in order, to the bold
  faced letters and digits in the word.

- A bold word must start with an upper-case letter.  Upper- and
  lower-case letters may be otherwise intermixed in a bold word.

- Upper-case letters may be written only in bold words and
  character-glyphs.

Tags.

- A tag is written as a sequence of {one or more} taggles.  The
  taggles cannot be separated by typographical display features.

- A taggle is written by writing, in order, the corresponding
  worthy letters and digits optionally followed by an underscore.

- A taggle must be adjacent to an underscore if its letters and
  digits correspond, in order, to those of a reserved word.

- If a taggle does not end with an underscore, it must be followed
  by a disjunctor.
```

# GNU extension - Nestable comments

- The traditional Algol 68 comment styles are non-nestable.

```
COMMENT this is a comment COMMENT
CO so is this CO
# and this as well #
```

- GNU68-2025-005 introduces two styles of nestable comments.

```
NOTE this comment can be nested ETON
{ and this one as well }
```

- GCC enables all comment styles in classic UPPER stropping, but only { and } in modern SUPPER stropping.

# GNU extension - unsafe clause (I)

```
proc get_fields_by_name = (Rec_Record record,
                           string field_name) []ref Rec_Field:
begin int nfields := 0;
      [record_num_fields (record)]ref Rec_Field fields;

      rec_mset_map (mset of record,
                    (int pos, Rec_MSET_Data d) void:
                       case d
                       in (ref Rec_Field field):
                             if name of field = field_name
                             then nfields +:= 1;
                                  fields[nfields] := field
                             fi
                       esac);
      fields[:nfields]
end;
```

```
$ ga68 -Wscope -g -O2 -c -o recsel.o recsel.a68
recsel.a68:153:53: warning: potential scope violation [-Wscope]
153 |                                     fields[nfields] := field
    |                                                     ^
```

# GNU extension - unsafe clause (II)

```
proc get_fields_by_name = (Rec_Record record,
                           string field_name) []ref Rec_Field:
begin int nfields := 0;
      [record_num_fields (record)]ref Rec_Field fields;

      rec_mset_map (mset of record,
                    (int pos, Rec_MSET_Data d) void:
                       case d
                       in (ref Rec_Field field):
                             if name of field = field_name
                             then nfields +:= 1;
                                  unsafe (fields[nfields] := field)
                             fi
                       esac);
      fields[:nfields]
end;
```

- New clause **unsafe**.
- Tells compiler the controlled clause is actually safe.
- Impacts both static and dynamic safety compilation modes.

# GNU extension - unsafe clause (III)

Formal description - addendum to RR

```
1.2.2

A) ENCLOSED :: ... ; unsafe.

3.7 Unsafe clauses

3.7.1 Syntax

a) SOID NEST unsafe clause :
     unsafe{94f} token, SOID NEST ENCLOSED clause{a,31a,c,34a,35a,-}.

{ Examples:
   a) UNSAFE (ptr := dst)  }

3.7.2 Semantics

a) A SOID-NEST-unsafe-clause N, in an environ E, is elaborated as
   follows:
   . The yield of N in E is the yield of the ENCLOSED-clause of N
     in E.
```

# A POSIX-like prelude for Algol 68

Prelude (library) providing low-level transput (input/output) capabilities.

- Environment: getenv, argc, argv.
- Error handling: errno, perror, strerror.
- Files: fopen, fcreate, fclose, fsize.
- Sockets: fconnect.
- Low-level transput: putchar, puts, fputc, getchar, fgetc, gets, fgets.

Implemented in libga68 and documented in the ga68 manual.

Algol 68

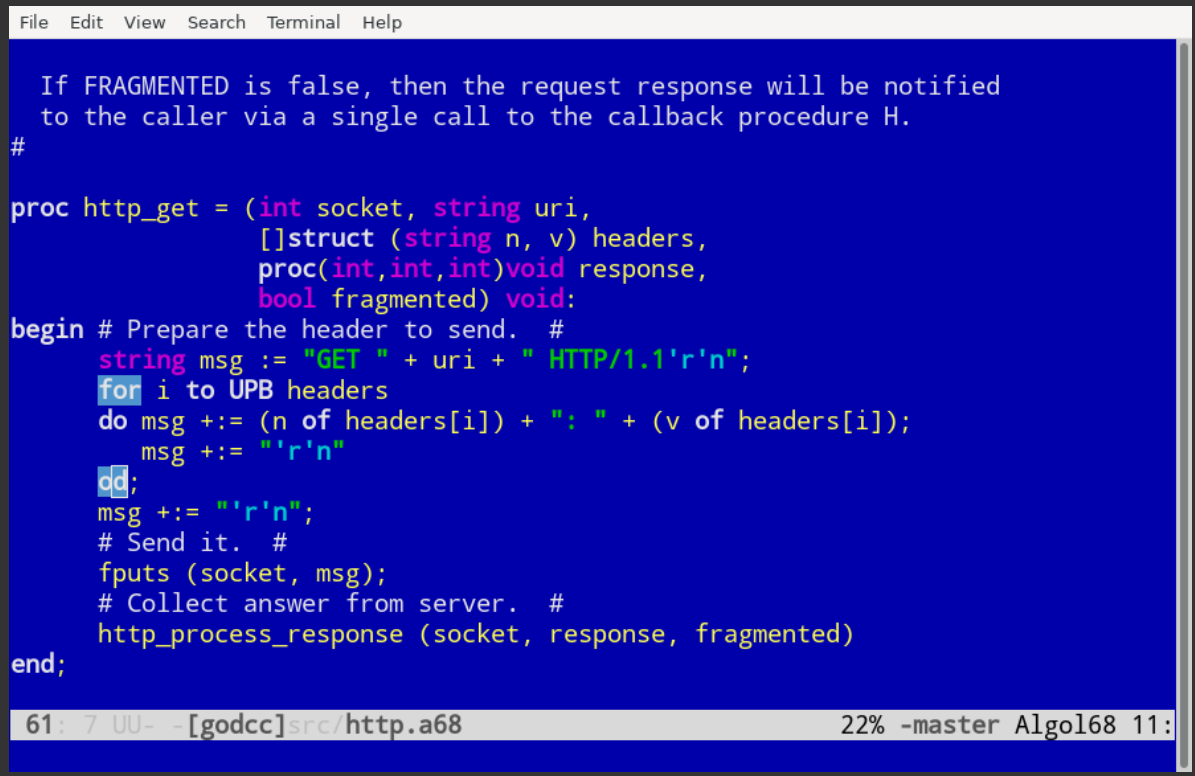
The GNU Algol 68 Working Group

The Algol 68 GCC Front-End

GNU extensions to Algol 68

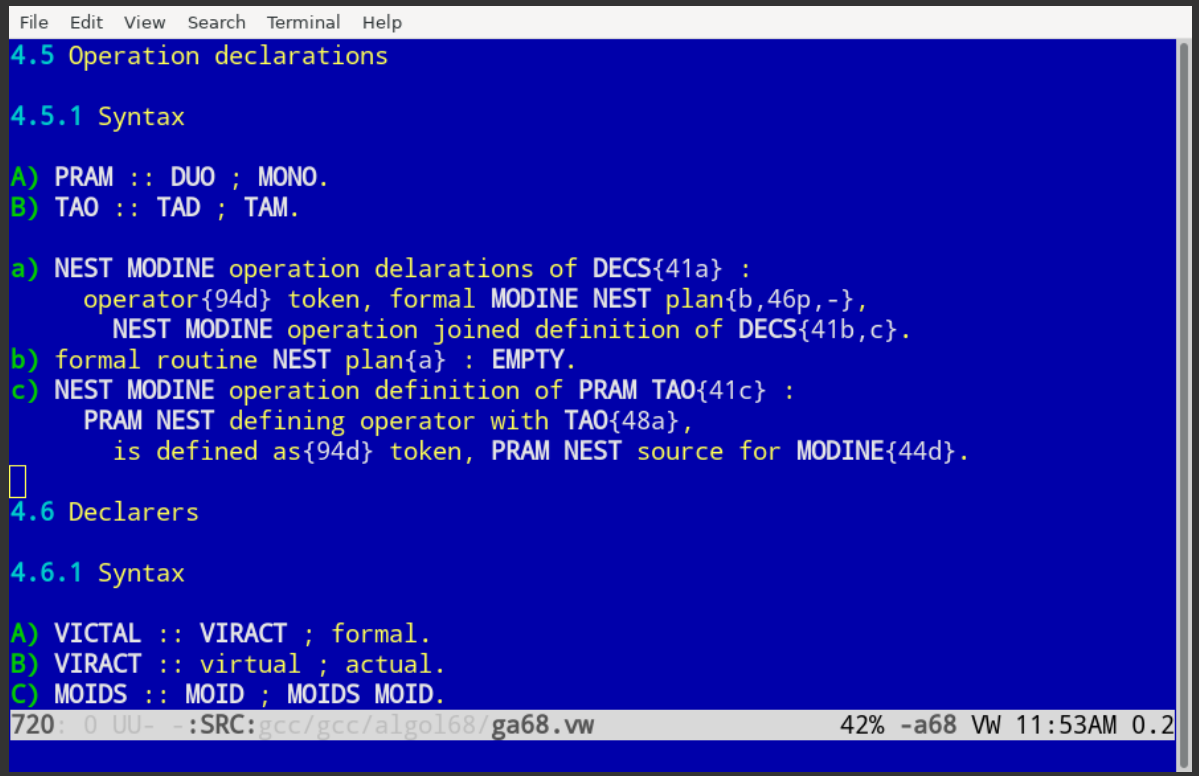
**Other projects**

# The Algol 68 Emacs mode




- https://git.sr.ht/~jemarch/a68-mode
- Syntax-driven automatic indentation and highlighting.
- Support for UPPER and SUPPER stropping regimes.
- Available in GNU ELPA via M-xpackage-install.

# The VW Emacs mode




- Mode to edit RR-style formal language descriptions
  https://git.sr.ht/~jemarch/vw-mode
- Created to help in the edition of gcc/algol68/ga68.vw.

# Algol 68 support in Autoconf

configure.ac

```
AC_INIT([whatever],[1.0])

AC_CONFIG_AUX_DIR([build-aux])
AM_INIT_AUTOMAKE

AC_PROG_A68

AC_CONFIG_FILES(Makefile src/Makefile)
AC_OUTPUT
```

```
jemarch@termi:~/gnu/hacks/godcc/build$ ../configure
[...]
checking for ga68... ga68
checking whether the Algol 68 compiler works... yes
checking for Algol 68 compiler default output file name... a.out
[...]
```

Support in git master upstream. Not released yet.

# Algol 68 support in Automake

src/Makefile.am

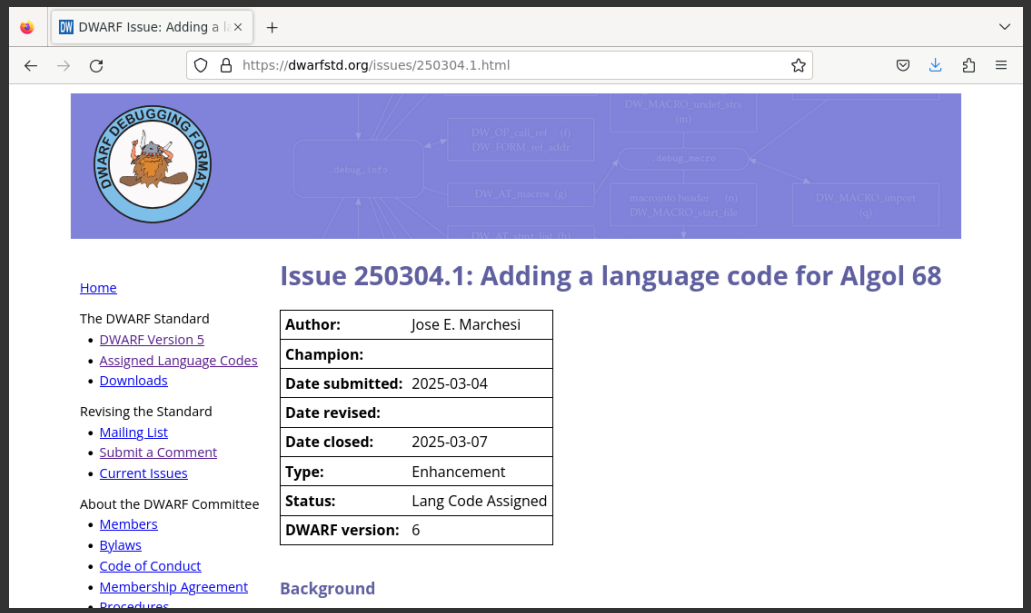
```
AM_A68FLAGS = -fa68-nil-checking

bin_PROGRAMS = godcc
godcc_SOURCES = godcc.a68
```

```
jemarch@termi:~/gnu/hacks/godcc/build$ make
Making all in src
make[1]: Entering directory '/home/jemarch/gnu/hacks/godcc/build/src'
ga68 -fa68-nil-checking -g -O2 -c -o godcc.o ../../src/godcc.a68
ga68 -fa68-nil-checking -g -O2   -o godcc godcc.o
[...]
```

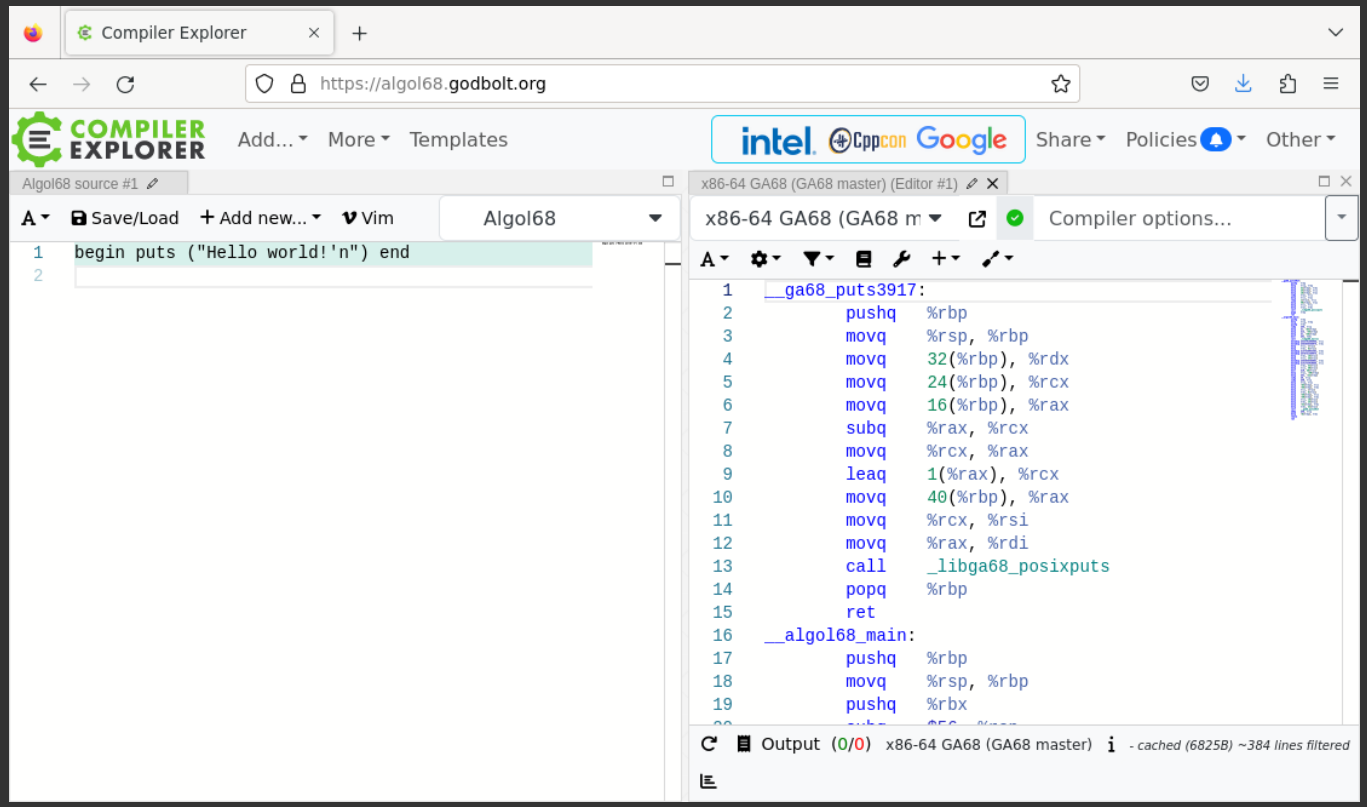
Released in Automake 1.18. Support for modules dependencies is WIP.

# Algol 68 in DWARF




- We got official DWARF codes assigned to Algol 68 https://dwarfstd.org/issues/250304.1.html
- For DWARF 5, DW_LANG_Algol68 = 0x44.
- For DWARF 6, DW_LNAME_Algol68 = 0x2e with versioning schema YYYY, starting with 1973 for the original Revised language. The language extensions we are working on will be encoded in subsequent versions, 2025 etc.

# Algol 68 in Compiler Explorer




- Work by Marc Poulhiès.
- ga68-snapshot - x86-64 GA68 (GA68 master)

# The Algol 68 Jargon File




- “A language in which fundamental concepts combine in an orthogonal way requires very precise terminology.” - Lindsey
- Published in HTML, PDF and manpages
  https://jemarch.net/a68-jargon
- Texinfo sources
  https://git.sr.ht/~jemarch/a68-jargon

# godcc - an interface to Compiler Explorer

```
$ godcc list compilers | grep a68
ga68-snapshot                           | x86-64 GA68 (GA68 master)
$ godcc compile -c cbpfgtrunk foo.c
# Compilation provided by Compiler Explorer at https://godbolt.org/
foo:
        *(u32 *) (r10+-4) = r1
        r0 = *(u32 *) (r10+-4)
        if w0 s<= 0 goto .L2
        r0 = *(u32 *) (r10+-4)
        w0 += 1
        r0 <<= 32
        r0 s>>= 32
        exit
.L2:
        r0 = 0
        exit
```

- Full-fledged command-line interface to Compiler Explorer.
- Fully written in Algol 68.
- Also serves as demo, showcase and compiler testcase.
- https://git.sr.ht/~jemarch/godcc

# Recap

- GNU Algol 68 WG
  - https://algol68-lang.org
  - https://git.sr.ht/~jemarch/algol68-lang
  - https://git.sr.ht/~jemarch/gnu68
- Algol 68 Jargon File
  - https://jemarch.net/a68-jargon
  - https://git.sr.ht/~jemarch/a68-jargon
- GCC front-end
  - https://gcc.gnu.org/wiki/Algol68FrontEnd
  - https://forge.sourceware.org/gcc/gcc-a68
- Autoconf
  - https://git.sr.ht/~jemarch/autoconf-a68
- DWARF
  - https://dwarfstd.org/issues/250304.1.html
- Emacs modes
  - https://git.sr.ht/~jemarch/a68-mode
  - https://git.sr.ht/~jemarch/vw-mode
- godcc
  - https://git.sr.ht/~jemarch/godcc


BACK
TO THE FUTURE
ALGOL 68