- NAME
- SYNOPSIS
- DESCRIPTION
- Copyright
- Agreement
- Version&Author
- Contents
- - 1 - Generalities
- - 2 - In the event of bug
- - 3 - Distribution files
- - 4 - Configuration, compilation and installation of dcc
  - UNIX OS
  - VMS OS
  - WindowsXX OS
  - Other
- - 5 - Execution files
  - Starter file
  - Adjustment files
- - 6 - Test files
- - 7 - Article
- - 8 - Available d-pragmas
  - /*~AddedObj*/
  - /*~BackBranch*/
  - /*~CastTo <type> */
  - /*~ComposingHdr*/
  - /*~DccCompliant*/
  - /*~DollarSign*/
  - /*~DynInit*/
  - /*~EndLongjmpCallable*/
  - /*~ExactCmp*/
  - /*~FullEnum*/
  - /*~Generic*/
  - /*~Ignore*/
  - /*~IndexType <type> */
  - /*~Init <listVar>*/
  - /*~LiteralCst*/
  - /*~LocalAdr*/
  - /*~Masking*/
  - /*~MayModify*/
  - /*~NeverReturns*/
  - /*~NoBreak*/
  - /*~NoDefault*/
  - /*~NonConstExp*/

    - /*~NotUsed*/
    - /*~NoWarn*/
    - /*~OddCast*/
    - /*~PopWarn*/
    - /*~PortableQM*/
    - /*~Private*/
    - /*~PrivateTo "<fileName>" [, "<fileName>"]* */
    - /*~PseudoVoid*/
    - /*~Public*/
    - /*~ResultPtr*/
    - /*~ResultType*/
    - /*~RootType*/
    - /*~SameValue*/
    - /*~Saved*/
    - /*~SideEffectOK*/
    - /*~SizeOfMemBlk*/
    - /*~SizeOK*/
    - /*~TypeCombination <combList>*/
    - /*~Undef <listIdent>*/
    - /*~Utility*/
    - /*~VoidToOther*/
    - /*~Warn*/
    - /*~zif <boolExp> <stringCst> */
    - Notes
- - 9 - Indentation rules
- - 10 - Options
- - 11 - Environment variables
- - 12 - Exit statuses
- - 13 - Badly implemented features
- - 14 - Unimplemented features
- - 15 - Known bugs
- - 16 - Main changes with respect to version 2.6
- - 17 - About 16 bits int platforms

---

1 NAME

dcc - Disciplined C Checker (version 2.7X)

---

2 SYNOPSIS

```
Syntax : dcc [options] [sourceFile [sourceFile [sourceFile ...]]]
```

3 DESCRIPTION

dcc is a C checker program, described in the December 1995 issue of ACM SIGPLAN Notices (see also 'dccarticle.ps' and 'dccarticle.ascii' included files).

4 Copyright

Copyright Ecole Superieure d'Electricite ('Supelec'), France, 1995.
All Rights Reserved.

5 Agreement

TITLE. Title, ownership rights, and intellectual property rights in and to the Software shall remain in Supelec and/or its suppliers. The Software is protected by international copyright treaties.

DISCLAIMER OF WARRANTY. Since the Software is provided free of charge, the Software is provided on an "AS IS" basis, without warranty of any kind, including without limitation the warranties of merchantability, fitness for a particular purpose and non-infringement. The entire risk as to the quality and performance of the Software is borne by you. Should the Software prove defective, you and not Supelec assume the entire cost of any service and repair. This disclaimer of warranty constitutes an essential part of the agreement. SOME STATES/COUNTRIES DO NOT ALLOW EXCLUSIONS OF AN IMPLIED WARRANTY, SO THIS DISCLAIMER MAY NOT APPLY TO YOU AND YOU MAY HAVE OTHER LEGAL RIGHTS THAT VARY FROM STATE/COUNTRY TO STATE/COUNTRY OR BY JURISDICTION.

LIMITATION OF LIABILITY. UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, TORT, CONTRACT, OR OTHERWISE, SHALL SUPELEC OR ITS SUPPLIERS OR RESELLERS BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES. SOME STATES/COUNTRIES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS LIMITATION AND EXCLUSION MAY NOT APPLY TO YOU.

6 Version&Author

dcc version 2.7, February 19th 2001

Yves Noyelle (<Yves.Noyelle@supelec.fr>)
Supelec, Service Informatique,
Plateau de Moulon, F-91192 Gif/Yvette Cedex

---

7 Contents

1 - Generalities
2 - In the event of bug
3 - Distribution files
4 - Configuration, compilation and installation of dcc
5 - Execution files
6 - Test files
7 - Article
8 - Available d-pragmas
9 - Indentation rules
10 - Options
11 - Environment variables
12 - Exit statuses
13 - Badly implemented features (as of now)
14 - Unimplemented features (as of now)
15 - Known bug(s)
16 - Main changes with respect to previous version
17 - About '16 bits int' platforms

---

8 - 1 - Generalities

dcc verifies that a C source file complies with the precepts of Disciplined C, a set of rules aiming at elevating ANSI C to the rank of high level language (as opposed to portable assembly language), and making as much semantics as possible flow from the design to the source code. This is achieved mainly via quasi-compulsory types/constants naming and 'const' qualifier use.

It also detects most of the pitfalls of the C language, helps the programmer organize his program modules and write them so that they are well structured, well encapsulated and more portable, and finally gives some much-needed compile-time checking tools.

dcc is quite fast (from 5 to 10 times faster than a compiler), and is itself written in Disciplined C.

---

9 - 2 - In the event of bug

```
No guarantee is given that this version of dcc is free of bugs,
although every efforts are made to chase and eliminate them.

If you uncover one (or more...), please let me know (at e-mail
<dccsupport@supelec.fr>); a bug report should include a description of
the problem and a short source file causing it, along with the
version/release number of dcc (see paragraph 10) and the platform
(machine/system) used.
```

---

10 - 3 - Distribution files

There are five sets of files :

- program files

```
configdc.th
dccFiles.mng
dc.th
dcblk.c/h/ph
dcdecl.c/h/ph
dcdir.c/h/ph
dcexp.c/h/ph
dcext.c/h/ph
dcfmt.c/h
dcinst.c/h/ph
dcmain.c/h/ph
dcprag.c/h/ph
dcrec.c/h/ph
dcrecdir.c/ph/th
dctxttok.c/h
dcmsg.*
```

- installation files

```
adaptStarterFile     (UNIX)    (courtesy of C. Bocage, CRI Supelec)
adaptstarterfilevms.com (VMS)  (courtesy of C. Bocage, CRI Supelec)
askfordfltopt.com    (VMS)
chooseMsgLanguage    (UNIX)
chooseStarterFile    (UNIX)
createLocalFilesMngFile (UNIX) (courtesy of C. Bocage, CRI Supelec)
createlocalexec      (UNIX)
descrip.mms          (VMS)
extracthdrfilesvms.com (VMS)   (courtesy of J. Lauret, SUNY@SB, USA)
frsttime.mms         (VMS)
installdccvms.com    (VMS)
installfile          (UNIX)
makefile             (UNIX)    (courtesy of C. Bocage, CRI Supelec)
makefile.win32       (WINDOWS) (courtesy of C. Enache, UPB student,
                                                          Romania)
rmdirvms.com         (VMS)
```

- execution files

```
*.adj                (adjustment files)
dynarray.h           (header file for dynamic arrays, independent of
                                                                  dcc)
predefmacvmsvax.txt
predefmacvmsalpha.txt
starter.dccCcHP715   (courtesy of F. Mullet, alumnus)
starter.dccClWin32   (courtesy of E. Bezine, alumnus)
starter.dccDjgpp     (courtesy of J.F. Tilman, alumnus)
starter.dccGccAlphaSkel
starter.dccGccHPUXSkel
starter.dccGccLinuxSkel (courtesy of Ch. Duverger, alumnus)
starter.dccGccMipsSkel
starter.dccGccSunosSkel (courtesy of Ch. Duverger, alumnus)
starter.dccVmsAlpha
starter.dccVmsVax
starter.dccXlcAix    (courtesy of W. Briscoe, freelance, UK)
```

- test files

```
tstdcc
tstdccvms.com
trydcc.c/h
restrydcc.txt
srchdiffrestry.com
```

- files describing dcc

```
dccarticle.ps
dccarticle.ascii
```

```
FILES  FROM  OLD DISTRIBUTIONS SHOULD BE DISCARDED BEFORE INSTALLING A
NEW VERSION (except possibly local adjustment files).

Note :  each dcc source file begins with a  /* <fileName> */  comment,
        and ends with a /* End <fileName> */ comment, to  make it easy
        to check that it has not been truncated by error.
```

---

11 - 4 - Configuration, compilation and installation of dcc

```
The   configuration   phase   means   possible   adaptations   of   files
'configdc.th',  'dccFiles.mng',  and  'dcmsg.txt'.   Choice   of   file
'starter.dcc' is discussed paragraph 5.

File 'configdc.th' serves to configurate dcc (buffer sizes and so on);
most settings should be valid on any machine (except  perhaps  smaller
memory  machines or machines  where NULL is  not all-zeros);  comments
are  supposed to be  meaningful enough  to make clear  the use of each
defined symbol.
The  symbol  LONGLONG  should only be  defined on  platforms  having a
larger integer size than 'long'; its value is then the  naming of that
```

type (e.g. "#define LONGLONG long long" or "#define LONGLONG __int64".) In the latter case, a "#define __int64 long long" should be put in the starter file. If LONGLONG is not defined, the syntax for 'long long' numerical constants ("LL") is not recognized (except in system header files).
The macro IsVisibleChar governs what an 'invisible character' is; it is to be adapted to locally used character set extension.
The SignedChar/SignedIntField symbols may have to be adapted from their default value.
EBCDIC character coding could be supported by changing the 'charInfo' array (file 'dcrec.c').
Input/output is done entirely via fopen/fread/feof/fflush/fputs/fclose/getchar/setvbuf. The only other used system functions are: malloc/realloc/free, setjmp/longjmp, exit/abort, system (to pass control to the local compiler), fileno/isatty, and getenv.

File 'dccFiles.mng' is to be adapted to local conditions; it indicates the directory where dcc execution files (adjustment files and the starter file) are to be found, and the system header files directory. If there are several system header files directories, use symbol 'PredefDCCDFLTS' (see paragraph 10; for gcc platforms, the command file 'createLocalFilesMngFile' computes the value of this symbol). The content of environment variable DCCFILES, if defined, supersedes these values; its format is:
<dccExecFilesDir> or
<dccExecFilesDir> <space> <sysHdrDir>.

Files 'dcmsg.###' contain the text of all messages (including errors/warnings); since these messages are C string literals, they may be changed at will. '###' indicates the language used for messages; at present, there exist "eng"lish and "fre"nch versions. In these files, the character sequence '@x', where x is a digit in the range '1'-'9', is not outputted as such, but replaced by a text generated by dcc. The character sequence '@0' toggles a flip-flop controlling output of the '@0' parenthesized message chunks (output if in interactive mode after 'e'/'E' command, or if options '+zve', '+zvve' activated).

11.1 UNIX OS

On an UNIX system, once these files set, compilation is done by typing 'make'. The 'make install' command installs dcc into the system (first edit file 'installfile' to define where you want it to be; unmodified, this file installs dcc in /usr/local/bin/, and the 'execution' files in /usr/local/lib/DccExecFiles/ (if changed, change also value of DccExecFilesDir symbol in file 'dccFiles.mng'). Then test dcc on itself (command file 'tstdcc'): this should generate no error nor warning.

11.2 VMS OS

On a VMS system, use the 'installdccvms.com' file. The following DCL logical names have to be defined beforehand:

dccexecfilesdir

directory where dcc execution files are installed,

dccsyshdrdir

directory where system header files ('stdio.h' etc.) can be found (in text form).

Also, 'dcc' has to be defined as a "foreign" command ($ dcc :== $ <complete executable file name>). The test file is 'tstdccvms.com'.

11.3 WindowsXX OS

On WindowsXX, with Microsoft VisualC++ 4.0, use 'makefile.win32' file; do not forget to set the INCLUDE, LIB, and PATH environment variables, e.g. :

```
set INCLUDE=C:\Program Files\DevStudio\VC\include
set LIB=C:\Program Files\DevStudio\VC\lib
set PATH=C:\Program Files\DevStudio\VC\bin;C:\Program Files\
                                          DevStudio\SharedIDE\bin
```

11.4 Other

On other systems, you are on your own...

Note : possible local compiler warnings while compiling dcc may safely be ignored (provided the compiler is mature).

---

12 - 5 - Execution files

Several files are needed by dcc to execute correctly:

12.1 Starter file

Its purpose is to define those symbols that are predefined by the local compiler. It also gives the name of the compiler (via a string literal, that must be the value of the '__dcc' symbol), and can override local features (such as specific specifiers/qualifiers, e.g. the "globalvalue" specifier of VMS C compilers). Starter files for VMS C compilers (both VAXs and ALPHAs), the ULTRIX gcc compiler, the OSF1

gcc compiler, the HP-UX cc/gcc compilers, the LINUX gcc compiler, the SUNOS gcc compiler, the AIX xlc compiler, the Djgpp compiler and the Windows VisualC++ cl compiler are included. For SUNOS, the system header files directory must be the gcc header files directory.

When the value of the '__dcc' symbol indicates gcc (or cc on a VMS/Alpha platform), a 'skeleton' starter file is automatically completed with the compiler predefined macros (command files adaptStarterFile*). This could be done for other compilers, provided they give a mean to obtain these macros.

Note 1 : the called compiler can be changed from the default by redefining the '__dcc' symbol in the dcc command line ('-D' or '/DEF' option), but then the starter file may have to be changed.

Note 2 : system include files are configurated via several symbols, such as _XOPEN_SOURCE or _POSIX_C_SOURCE; these symbols are not defined by default in dcc (except if the called compiler is gcc or VMS/Alpha cc), which may result in 'undefined function' messages. To set such a symbol, either define it in the 'starter.dcc' file (then it is global for all users), or define it using the '-D' command line option (possibly via the DCCDFLTS environment variable, see paragraph 10).

12.2 Adjustment files

Their purpose is to amend system header files, so that they do not cause dcc to issue undue warnings. Conceptually, an adjustment file is appended to the end of the corresponding header file (after the first inclusion, or after each inclusion if symbol ADJFILES_INCLUDED_AT_EACH_INCL is defined in the starter file; cf file 'time.adj').

An adjustment file cannot declare/define anything not declared/defined by the corresponding header file (except if something is missing in it, e.g. the prototype for the "sbrk" function in stdlib.h; see then d-pragma /*~AddedObj*/, paragraph 8). They may redeclare at will system function, objects, typedefs (even already used ones) and macros.

An identifier not declared in the corresponding system header file is ignored. There may be several redefinitions for a given function/structure/union; the one that is chosen is the one of the same sort and with the same number of parameters/fields.

A system macro (with parameter, that is not a symbol), for example "setjmp", after having been met once in an adjustment file, is not expanded any more (being replaced by a function prototype).

The 'bool' type is implicitely defined inside an adjustment file, but precautions should be taken if is also defined by the corresponding system header file (cf. 'curses.adj').

Any preprocessor feature can be used in an adjustment file. Besides, the 'cdefined' function, similar to the 'defined' preprocessor function but answering True if its parameter is a defined system 'program' identifier (as opposed to a 'preprocessor' identifier), can also be used (see for example 'signal.adj').

By convention, for a system header file named 'xxx.h', the corresponding adjustment file is named 'xxx.adj', and is searched by default in the DccExecFilesDir directory. If one wants another suffix, or wants to put (some) adjustment files in other directories, one has to create an 'adjFiles.dcc' file in the DccExecFilesDir directory, file whose format is (for each line):

```
  "<system header file name>" , "<corresponding adjustment file
                                  full name (with absolute path)>" ;
```

Example of 'adjFiles.dcc' file:

```
  #define X11AdjDir "/usr/local/include/DccAdj/"
  "icon.h", X11AdjDir "icon.adj";
  "Kn/Knvas.h", X11AdjDir "Knvas.adj";
  ...
  #undef X11AdjDir
```

Should any header file reveal to be unamendable, it can be purely replaced by the corresponding adjustment file. It must then be described in the 'adjFiles. dcc' file, and the corresponding line must be prefixed by an exclamation point ('!').

Adjustment files for most C system header files (curses.h: courtesy of W. Briscoe, freelance, UK) are included; adjustment files for the X-Window library are available, but only partially checked.

---

13 - 6 - Test files

The command files 'tstdcc*' just check dcc on its own source files; this check should not generate any error nor warning.

You can get a feeling of what dcc can do for you by typing the command "dcc trydcc.c". You can also limit dcc scrutiny by using appropriate options (see paragraph 10) or by conditional compilation using the '__dcc' symbol.

There are many more non-regression check files, but they are not included.

---

14 - 7 - Article

Unfortunatly, the wrong version of the paper was published in SIGPLAN Notices; so the right version (in fact updated and somewhat more detailed) is included, in Postscript form, along with a pure ASCII form (for 'diff' purposes).

15 - 8 - Available d-pragmas

For the following, a "file identifier" is defined to be composed of a "file access path" (machine name, directories...), followed first by a "file name" and then by a "file suffix" that begins on the last dot '.' seen in the file identifier.

A "header" file is a source file whose file suffix contains the letter 'h' at least once, and which is included via the '#include' directive; a "system header" file is a header file whose '#include' directive uses the '<...>' form. A "body" file is a source file which is neither a header file, nor a file included by a header file.

Except /*~DccCompliant*/, /*~DollarSign*/, /*~NoWarn*/, /*~PopWarn*/, /*~Private*/, /*~PrivateTo*/, /*~Public*/, /*~TypeCombination*/, /*~Warn*/, d-pragmas have purely local effect, that is they have to be used each time the corresponding situation occurs.

15.1 /*~AddedObj*/

allows a new function/object/type to be added in an adjustment file:

```
void sbrk/*~AddedObj*/(long);
```

15.2 /*~BackBranch*/

avoids warning on backward branches:

```
goto alrdDefLabel /*~BackBranch*/;
```

15.3 /*~CastTo <type> */

allows pseudo-cast between parallel types; also to be used to avoid warning on comparison of difference(s) of unsigned int:

```
typedef struct {int x;}Ts; typedef Ts Ts1;
Ts s; Ts1 s1; unsigned int ui1, ui2;
...
s1 = /*~CastTo Ts1*/ s;
if (/*~CastTo unsigned int */ (ui2 - ui1) > 1) ...
```

15.4 /*~ComposingHdr*/

tells that the whole service provided by the included header file is to be part of the service offered by the including header file:

```
#include <math.h> /*~ComposingHdr*/
```

Permits also a header file to carry a name different from its body file (through a dummy header file carrying the right name).

15.5 /*~DccCompliant*/

in a system header file, indicates that the (remaining portion of the) file conforms to dcc requirements, and that 'typedefs' will create parallel types if applicable; usable anywhere (in a system header file).

15.6 /*~DollarSign*/

must be at beginning of module (possibly after comments); autorizes '$' in identifiers.

15.7 /*~DynInit*/

avoids warning on dynamic initialization (by constants) of compound objects:

```
    [auto] struct _s toto = {...} /*~DynInit*/;
```

15.8 /*~EndLongjmpCallable*/

tells the 'volatile' qualifier checking algorithm that the "longjmp" function won't be called beyond this point. Usable anywhere in the body of a function, after a "setjmp"-controlled 'if' statement.
.

15.9 /*~ExactCmp*/

avoids warning on floating point comparison for (in)equality:

```
    if (fltVar == 0 /*~ExactCmp*/) ...
```

15.10 /*~FullEnum*/

asks dcc to warn if not every constant of the (enum) switch type has been used as case values; only useful if a 'default' clause is used (for example to catch possible "strange" values of the enum expression):

```
    default: /*~FullEnum*/ ...
```

15.11 /*~Generic*/

tells dcc that the returned type of a function is compatible with any non-closed descendant of this type (subtyping) or, in case of 'void *' returning function, with any pointer. Not valid if any parameter is marked /*~ResultType*/ or /*~ResultPtr*/;incompatible with /*~Utility*/:

```
    int atoi(const char *) /*~Generic*/;
    double sin(double) /*~Generic*/;
    void *allocateChunk() /*~Generic*/;
```

The variant /*~Generic <fctList> */ is allowed only in adjustment files (see /*~RootTyp*/).
/*~Generic*/ can also be used with function parameters, to specify that the so-qualified formal parameter is compatible with any descendant (closed or not):

```
    size_t strlen(const char * /*~Generic*/);
    void (*savPSCF)(char /*~Generic*/);
```

15.12 /*~Ignore*/

tells dcc to ignore object-like or function-like calls to defined pseudo-macro. Allowed only in starter files:

```
#define /*~Ignore*/ __asm
```

15.13 /*~IndexType <type> */

specifies type of index values usable for an array, allocated either statically or dynamically (default index type for static arrays: type of bound, except if bound given by a plain arithmetic constant; then any integral arithmetic type), or that can be added to a pointer. To be used in array/pointer declaration:

```
Color arr[ArrSiz /*~IndexType ColorIndex */];
bool * /*~IndexType Trow*/ * /*~IndexType Tcol*/ twoDimSwitchArray;
```

or pointer creation:

```
& /*~IndexType Tcol*/ col
```

15.14 /*~Init <listVar>*/

tells the initialization-checking algorithm that, from now on, all variables in the list can be considered as initialized. Very specific (see X-Window applications, or adjustment files). Usable anywhere (in the scope of the variables):

```
XtSetArg(arg[0], XmNchildren, /*~Init children*/ &children);
```

15.15 /*~LiteralCst*/

tells that expressions of a so qualified parallel type accept unnamed constants without warnings (see also options '-zuc', '-zgpr', paragraph 10):

```
typedef int Int /*~LiteralCst*/;
```

15.16 /*~LocalAdr*/

avoids warning when returning address of local object from a function, or assigning such address to global/external pointer:

```
gblPtr = &localObject + 1 /*~ LocalAdr */;
```

15.17 /*~Masking*/

avoids warning if a macro name is the same as an already existing identifier:

```
#define /*~Masking*/ macroName macroBody
```

15.18 /*~MayModify*/

indicates that modifications may occur through a supposedly const-pointing pointer parameter (via casting), or through a struct/union parameter containing non-const pointers. To be used in functions whose name implies modifications (such as freeing functions), or in case of

```
(possible) slight modifications:
    void freeTree(const struct _tree *x /*~MayModify*/);
    void clipTree(struct _tree x /*~MayModify*/);
See also /*~ResultPtr*/ d-pragma.
```

15.19 /*~NeverReturns*/

```
documents that a void function never returns control:
    static void errExit(...) /*~NeverReturns*/;
The variant  /*~NeverReturns <fctList> */  is allowed only in adjustment
files (see /*~RootTyp*/).
```

15.20 /*~NoBreak*/

```
avoids warning if falling through the end of a 'case' statement:
    case C1 : i = 1;  /*~NoBreak*/
    case C2 : i++;  break;
```

15.21 /*~NoDefault*/

```
avoids  warning  if no 'default'  case at  end of 'switch'  statement on
enumeration  (not  needed of course if all  constants have  been used as
case values):
    case Cn : i = ... ;
    /*~NoDefault*/
```

15.22 /*~NonConstExp*/

```
makes dcc believe that a (parenthesized) expression is not constant:
    if ((~0==-1)/*~NonConstExp*/) ...
Notice  that the  Pascalian  form "while (TRUE)"  can be  advantageously
replaced by the C idiom "for (;;)",
```

15.23 /*~NotUsed*/

```
prevents  warning on  not-used  object or enum constant  or formal para-
meter (function or macro):
    enum {Ce1, Ce2 /*~NotUsed*/, Ce3 ...}
    static void shift(Tstring x, TtypeElt y /*~NotUsed*/);
    #define Sink(x /*~ NotUsed */)
    static const char *versionName = "Version 3.3.1" /*~NotUsed*/;
```

15.24 /*~NoWarn*/

```
to be  used in front of  sections of code not (yet)  dcc-compliant  (see
also /*~Warn*/, /*~PopWarn*/ d-pragmas);  usable anywhere.  Warnings are
automatically disabled inside system header files.
```

15.25 /*~OddCast*/

to make dcc swallow a cast it frowns upon:

```
ptrInt = (int * /*~OddCast*/) ptrStruct;
```

15.26 /*~PopWarn*/

goes back to previous Warn/NoWarn state (see /*~Warn*/, /*~No Warn*/ d-pragmas); usable anywhere. At least 16 levels kept.

15.27 /*~PortableQM*/

to make dcc swallow a cast it has good reasons to believe not portable:

```
ptrFloat = (float * /*~PortableQM*/)ptrDbl;
```

15.28 /*~Private*/

if used in header file X.*h*, is strictly equivalent to /*~PrivateTo "<currentFile>", "X.c" */.

15.29 /*~PrivateTo "<fileName>" [, "<fileName>"]* */

indicates that struct/union members, or enum constants, declared thereafter are only visible from the indicated file(s), or from macros defined in the indicated file(s) or called by such macros. Also needed to indicate in which file is defined an incomplete struct/union (if /*~Private* */ indicates several files, this file is taken as the last one).
Usable anywhere (in a header file).
Scope: until next /*~Private*/, /*~PrivateTo*/ or /*~Public*/ d-pragma; an '#include' of (another) header file creates a hole in the scope for the duration of the include. File names may contain joker characters ('*' matches any set of characters; '%' matches any character):

```
/*~PrivateTo "dcrec.c", "dc%%%.*h*" */
```

15.30 /*~PseudoVoid*/

authorizes a so-declared (non void-returning) function to be used as a statement:

```
char *strcpy(char *x, const char *y) /*~PseudoVoid*/;
```

The variant /*~PseudoVoid <fctList> */ is allowed only in adjustment files (see /*~RootTyp*/).

15.31 /*~Public*/

indicates end of last /*~Private#*/ scope; usable anywhere (in a header file).

15.32 /*~ResultPtr*/

to be used for (pointer) parameters that are returned as result; same effect as /*~ResultType*/, plus allows (at calling level) "should be 'const'" pointer checking propagation:

```
    char *strchr(const char * /*~ResultPtr*/, char);
```

15.33 /*~ResultType*/

indicates that the result type of a function call is the type (or the smallest common type) of the current actual parameter(s) corresponding to the so qualified formal parameter(s):

```
    void *realloc(void *old /*~ResultType*/, size_t size);
    Window mergeWindow(Window/*~ResultType*/, Window/*~ResultType*/);
    long sum(long m /*~ResultType*/, long n /*~ResultType*/);
    short s; signed char b; unsigned char ub;

    s = sum(b, s);  /* OK; returned type behaves as 'short' from dcc
                                                        standpoint. */
    s = sum(ub, s); /* idem */
    b = sum(SCHAR_MAX + 1, b); /* WRONG, because return type is
                                                          'short'. */
```

15.34 /*~RootType*/

closes (renders incompatible with its hierarchy) the type subtree headed by the so qualified (parallel) type:

```
    typedef unsigned int Talgn /*~RootType*/;
```

The variant /*~RootType <typeList>*/ is allowed only in adjustment files:

```
    /*~RootType wchar_t, wint_t*/
```

15.35 /*~SameValue*/

avoids warning if an enum constant have the same value that a previous constant of the same enum:

```
    enum _colors {BEG_COLORS, INFRARED=BEG_COLORS /* ~SameValue not
       necessary here, because 'BEG_COLORS' is last defined constant */,
        RED, ORANGE, YELLOW, ..., BEG_VISIBLE_COLORS=RED /*~SameValue*/}
```

15.36 /*~Saved*/

to be used if a formal parameter which is a pointer is saved into permanent storage (static/extern):

```
    void qio(Semaphore * /*~Saved*/);
```

15.37 /*~SideEffectOK*/

tells that a side effect via a macro parameter is OK:

```
    #define DangerousMin(x, y/*~SideEffectOK*/) (x<y)? x : y
```

```
    DangerousMin(oldMin, tab[i++])
```

15.38 /*~SizeOfMemBlk*/

allows to check, for 'malloc'-like functions using 'sizeof' as parameter, whether the type of the (first) sizeof argument is the type pointed by the receiving pointer:

```
    void *malloc(size_t /*~SizeOfMemBlk*/);
```

15.39 /*~SizeOK*/

avoids warning when no width limit is specified in a conversion specification of 'sscanf', or when a large struct/union is passed as parameter:

```
    (void)sscanf(charPtr1, "%s", charPtr2/*~SizeOK*/);
```

15.40 /*~TypeCombination <combList>*/

indicates allowed combinations between (a priori incompatible) parallel types. Syntax:

```
    <combList>::= <combElt> [ , <combElt> ]*
    <combElt> ::= <typeIdent> <operator> <typeIdent> -> <typeIdent>
```

The possible operators are: +, -, *, /, %, ~+ (non-commutative add) and ~* (non-commutative multiply). For the / operator, if one operand is a descendant of the other, the result type can be a native type (for use as a generic coefficient). Besides, some combinations are automatically deduced:

```
- for '+', from Ta + Tb -> Tc are deduced:
                Tb + Ta -> Tc   (1)
                Ta - Tb -> Tc   (2) (deduced only if Tc = Ta)
- for '*', from Ta * Tb -> Tc are deduced:
                Tb * Ta -> Tc   (3)
                Tc / Ta -> Tb   (4)
                Tc % Ta -> Tc   (5)
                Tc / Tb -> Ta   (6)
                Tc % Tb -> Tc   (7)
```

Lines 5 and 7 are deduced only if Ta and Tb are of integer type.

If necessary, type hierarchy is searched to find an applying type combination. D-pragma usable only outside of any block. Once given, cannot be desactivated:

```
    typedef float Volt, Amp, Watt, Ohm;
    typedef int Tgen;   /* 'generic' type */
    typedef Tgen Tspe;  /* specialization of Tgen */
    /*~TypeCombination Volt  * Amp  -> Watt,
                       Amp   * Ohm  -> Volt,
                       Watt  / Watt -> float,
                       Tspe ~+ Tgen -> Tspe */
    typedef int Fruit;
    typedef Fruit Apple, Pear;
    /*~TypeCombination Apple + Pear -> Fruit */
    {
      Volt v; Amp i; Watt p, p1; Ohm r; Tspe spe; Tgen gen;
      typedef Apple MacIntosh;
      MacIntosh macIntosh; Fruit fruit; Pear pear;
```

```
    p = r * i * i;     /* OK */
    p = v * (v / r);   /* OK */
    r *= p/p1;         /* OK */
    spe -= gen;        /* OK; illegal if no ~TypeCombination, because
                            contrary to dcc default hierarchy rules. */
    fruit = pear + macIntosh;  /* idem */
   }
```

15.41 /*~Undef <listIdent>*/ =head2 /*~UndefTag <listIdent>*/

terminates (for dcc) the scope of all indicated identifiers ('general' or 'tag' name space); usable anywhere (outside of any block):

```
    /*~Undef TpermSto, headPermSto*/
```

15.42 /*~Utility*/

to be used for functions returning a representation type that has no reason to be named. Incompatible with /*~Generic*/:

```
    int scanf(const char *, ...) /*~Utility*/;
```

Can also be used for a function formal parameter, to specify that actual parameter cannot be of parallel type.

15.43 /*~VoidToOther*/

autorizes automatic conversion from (non-generic) 'void *' type to any other pointer type, or to read, via scanf, into a 'void *' variable:

```
    objPtr = /*~VoidToOther*/ ptrOnVoid;
    scanf("%i", /*~VoidToOther*/ ptrOnVoid);
```

15.44 /*~Warn*/

to be used in front of dcc-compliant code (default state at beginning of program, except if '-znw' option used); usable anywhere. See also /*~PopWarn*/, /*~NoWarn*/ d-pragmas.

15.45 /*~zif <boolExp> <stringCst> */

causes emission on stderr (at dcc execution time) of <stringCst> if compile-time evaluable <boolExp> is true; can make use of special 'zif' functions (see "Compile-time checking tool" paragraph in the paper on dcc), and be used anywhere (also inside macros):

```
    /*~ zif (sizeof(typArray)/sizeof(typArray[0]) !=
                __extent(TtypName)+1) "Array 'typArray': bad length" */

    typedef struct{int a, b, c;} AnyStruct;
    static AnyStruct x = {
      0  /*~zif !__member(a) "misplaced init for field 'a'" */,
      3  /*~zif !__member(b) "misplaced init for field 'b'" */,
      -2 /*~zif !__member(c) "misplaced init for field 'c'" */,
      /*~zif !__member() "Structure 'x' not fully initialized" */};
```

15.46 Notes

- for dcc, a d-pragma is made up of tokens, the inner ones being perfectly macro-substitutable; for instance, the /*~LocalAdr*/ d-pragma consists of the three following tokens:
  ```
  /*~          (special token)
  LocalAdr     (identifier)
  */           (special token)
  ```
- an empty d-pragma name is legal; the d-pragma is then ignored (except in the reach of a '#' preprocessing operator, in which case it translates into a space),
- d-pragmas are visible in the list generated by the '+zlt' option,
- numerous examples of use of d-pragmas can be found in dcc source files.

---

16 - 9 - Indentation rules

Indentation is checked only on lines beginning a declaration or a statement.

General case :

Indentation level is increased on entering a block (just after its opening brace), for the duration of the block, or on entering a sub-statement (statement controlled by 'if', 'else', 'do', 'while', 'for'), for the duration of the substatement.

Special cases :

- no change of indentation level for construct "else if", both tokens being on same line;
- block as substatement may be lined up with controlling statement;
- 'case/default's may be lined up with corresponding 'switch'.

---

17 - 10 - Options

Their list can be obtained by executing a dcc command alone (no argument); the version/release number is also given. If the prefix '+' is replaced by '-', or vice versa, the option effect is reversed.

A given option is valid for all files met subsequently in the command line, unless reverted or cancelled (cf. option '+zrst'); options given after last file name apply only to last file (VMS).

Available options :
```
 -zac  no missing 'const' qualifier check,
 -zbo  no 'bool' type check,
 -zcc  do not call compiler,
 -zcw  call compiler only if no warning/error,
 -zefi no check of inefficient constructs,
 -zfci do not check first character of identifiers,
 -zfrt no unnamed function return type check,
```

```
 -zgpr no check of "good programming practices",
 -zinc no '#include' position check,
 -zind no indentation check,
 -zlvlX call compiler only if there is no warning of level >= X,
 -znui no check of unused identifiers,
 -znup no check of unused function parameters,
 -znw  start in "no warning" mode,
 -zpe  no check of various possible errors,
 -zpo  no portability check,
 -zrd  no readability check,
 -zsam no stopping after each message,
 -ztr  no trailer,
 -zuc  no check of unnamed constants,
 -zwa  no warnings,
 -zwr  no forced newline at 80 characters,
 +zae  report all  errors/warnings (default is, after a few  messages,
       to report only first error of each statement or declaration),
 +zafn always display  current file  name  in  error/warning messages,
 +zctn check first character  of  (created)  type  names (if '-zfci'),
 +zdol allows '$' in identifiers,
 +zepl warn on empty formal parameter list,
 +zkwm keywords redefinable (via macro),
 +zlt/+zltX list last tokens  processed before error/warning  (X =
       buffer length),
 +zmcc check all pointers for missing 'const' qualifier,
 +zmic more index type checking,
 +zmsgX  X: maximum number of emitted messages (default is 20),
 +zpnt check that function parameters are of named type,
 +zrhw report only errors or else highest level warnings,
 +zrst reset all previous options,
 +zsy  print each block's symbol table,
 +ztabX  X: tab  spacing (for indentation purposes; 8 is default),
 +zudt warn on tags declared (but not defined) in header files,
 +zusg give 'usage',
 +zve  output more informative error/warning messages,
 +zvve idem '+zve', plus output type descriptions in full.

Options not  beginning with `+z'/`-z'  are transmitted to compiler, but
dcc interprets  `-I'/`-D'/`-U' cc options (or their VMS  counterparts),
besides  passing them to the compiler. For VMS, only one  macro can  be
defined by a  given /DEF; /DEF, /INCL, /UNDEF options have  to be named
that way, and separated by spaces.

The environment variable DCCDFLTS can  contain any part of the command
line;  its  content  is  logically  added  (just  after  'dcc') to the
beginning  of  all following  dcc  command  lines.  The  value  of the
'PrefixDCCDFLTS' symbol (file 'dccFiles. mng') is also added.
Examples of setting of DCCDFLTS:
- UNIX          setenv DCCDFLTS '-zsam +zvve +zlt'
- VMS           $ DCCDFLTS = "-zsam +zvve +zlt"
```

18 - 11 - Environment variables

There are two of them: DCCFILES (see paragraph 4), and DCCDFLTS (see paragraph 10).

---

19 - 12 - Exit statuses

dcc provides six different exit statuses, whose values depend on local platform (see file 'configdc.th'): EXIT_SUCCESS, two EXIT_WARNINGS, two EXIT_ERRORS, and EXIT_FAILURE (returned when dcc has to stop before end).

---

20 - 13 - Badly implemented features

- floating constants are recognized lexically, but ill converted if they have a fractional part or a non-nul exponent (only noticeable in fancy array declaration, e.g. "char arr[(int)(2.5+6.5)]" => arr[4], not [9]),
- not all cases of non-portability checked.

Note: 'sizeof' does not take into account alignment holes, but this is done on purpose, to render portable some checks via d-pragma /*~zif*/.

---

21 - 14 - Unimplemented features

- check for non-ambiguity of external identifiers,
- size of objects not given ('+zsy' option),
- computation of floating-point constant expressions.

---

22 - 15 - Known bugs

None

---

23 - 16 - Main changes with respect to version 2.6

- d-pragma /*~NotUsedAfter*/ removed (not useful anymore),
- big improvement in management of adjustment files (cf. modifications to paragraph 5),
- revision of all adjustment files,
- cases of '16-bits int' platforms, and of platforms where NULL is not all-zeros, taken care of (but not tested).

24 - 17 - About 16 bits int platforms

```
Portage completed (but not tested).
```

dcc - Disciplined C Checker (version 2.7X)

Table of Contents

1 NAME 2
2 SYNOPSIS 3
3 DESCRIPTION 3
4 Copyright 3
5 Agreement 3
6 Version&Author 4
7 Contents 4
8 - 1 - Generalities 4
9 - 2 - In the event of bug 5
10 - 3 - Distribution files 5
11 - 4 - Configuration, compilation and installation of dcc 6
11.1 UNIX OS 7
11.2 VMS OS 8
11.3 WindowsXX OS 8
11.4 Other 8
12 - 5 - Execution files 8
12.1 Starter file 8
12.2 Adjustment files 9
13 - 6 - Test files 10
14 - 7 - Article 10
15 - 8 - Available d-pragmas 11
15.1 /*~AddedObj*/ 11
15.2 /*~BackBranch*/ 11
15.3 /*~CastTo <type> */ 11
15.4 /*~ComposingHdr*/ 11
15.5 /*~DccCompliant*/ 12
15.6 /*~DollarSign*/ 12
15.7 /*~DynInit*/ 12
15.8 /*~EndLongjmpCallable*/ 12
15.9 /*~ExactCmp*/ 12
15.10 /*~FullEnum*/ 12
15.11 /*~Generic*/ 12
15.12 /*~Ignore*/ 13
15.13 /*~IndexType <type> */ 13
15.14 /*~Init <listVar>*/ 13
15.15 /*~LiteralCst*/ 13
15.16 /*~LocalAdr*/ 13
15.17 /*~Masking*/ 13
15.18 /*~MayModify*/ 13
15.19 /*~NeverReturns*/ 14
15.20 /*~NoBreak*/ 14
15.21 /*~NoDefault*/ 14

15.22 /*~NonConstExp*/ 14
15.23 /*~NotUsed*/ 14
15.24 /*~NoWarn*/ 14
15.25 /*~OddCast*/ 15
15.26 /*~PopWarn*/ 15
15.27 /*~PortableQM*/ 15
15.28 /*~Private*/ 15
15.29 /*~PrivateTo "<fileName>" [, "<fileName>"]* */ 15
15.30 /*~PseudoVoid*/ 15
15.31 /*~Public*/ 15
15.32 /*~ResultPtr*/ 16
15.33 /*~ResultType*/ 16
15.34 /*~RootType*/ 16
15.35 /*~SameValue*/ 16
15.36 /*~Saved*/ 16
15.37 /*~SideEffectOK*/ 16
15.38 /*~SizeOfMemBlk*/ 17
15.39 /*~SizeOK*/ 17
15.40 /*~TypeCombination <combList>*/ 17
15.41 /*~Undef <listIdent>*/ =head2 /*~UndefTag <listIdent>*/ 18
15.42 /*~Utility*/ 18
15.43 /*~VoidToOther*/ 18
15.44 /*~Warn*/ 18
15.45 /*~zif <boolExp> <stringCst> */ 18
15.46 Notes 19
16 - 9 - Indentation rules 19
17 - 10 - Options 19
18 - 11 - Environment variables 21
19 - 12 - Exit statuses 21
20 - 13 - Badly implemented features 21
21 - 14 - Unimplemented features 21
22 - 15 - Known bugs 21
23 - 16 - Main changes with respect to version 2.6 21
24 - 17 - About 16 bits int platforms 22