Gaol 4.2.0

NOT Just Another
Interval Arithmetic Library

Edition 4.4
Last updated 21 May 2015

Frédéric Goualard
Laboratoire d'Informatique de Nantes-Atlantique, France

Part of the work was done while Frédéric Goualard was a postdoctorate at the *Swiss Federal Institute of Technology, Lausanne, Switzerland* supported by the *European Research Consortium for Informatics and Mathematics* fellowship programme.
This is edition 4.4 of the gaol documentation. It is consistent with version 4.2.0 of the `gaol` library.

Recipriversexcluson. n. *A number whose existence can only be defined as being anything other than itself.*

Douglas Adams, 1982
Life, the Universe and Everything

Contents

Copyright

Gaol is distributed under the GNU Lesser General Public License (see Section 20, page 73). The copyright for the initial version—named `cell`—(year 2001) is owned by the *Swiss Federal Institute of Technology*, Lausanne, Switzerland. The copyright for the following versions (from 2002 onward) is owned by the *Laboratoire d'Informatique de Nantes-Atlantique*, France.

For the input of floating-point numbers, gaol uses the `strtord()` function written by David M. Gay, whose copyright notice follows:

```
* The author of this software is David M. Gay.
*
* Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
*
* Permission to use, copy, modify, and distribute this
* software for any purpose  without fee is hereby  granted,
* provided that  this entire notice is included in all
* copies of any software  which is or includes  a copy or
* modification  of this software and  in all copies of the
* supporting documentation for such software.
*
* THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY
* EXPRESS OR IMPLIED WARRANTY.  IN PARTICULAR, NEITHER
* THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR
* WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY
* OF THIS SOFTWARE OR ITS FITNESS  FOR ANY PARTICULAR
* PURPOSE.
```

Gaol relies either on IBM APMathlib or CRlibm mathematical libraries for most floating-point operators. Both libraries are released under the GNU Lesser General Public License (see Section 20, page 73).

1 Introduction

Gaol[†] is a C++ library to perform arithmetic with floating-point intervals. The development of gaol was initiated at the *Swiss Federal Institute of Technology*, Lausanne, Switzerland, while F. Goualard was a post-doctorate fellow supported by the *Swiss National Science Foundation*. It started as a limited version of *Jail* (now *halloween*), a templated C++ interval library developed during Goualard's PhD.

To our knowledge, a unique feature of gaol among all C++ interval arithmetic libraries available is the implementation of *relational arithmetic operators* required by interval constraint arithmetic software (see Section 8.2, page 34). Hence, the game of the name: gaol is *not* JAIL (*Just Another Interval Library*). That situation should change in the near future, as relational arithmetic operators are required by the future IEEE 1788 standard for interval arithmetic.

This document is both a manual and a reference to use gaol. It assumes a prior knowledge of interval arithmetic. Refer to the books and papers by Goldberg, Neumaier, and others [2,4,5,1,6] for a basic presentation of floating-point arithmetic, interval arithmetic and the use thereof.

The main entry point for interval arithmetic on the Web is Vladik Kreinovich's *Interval Computation site* (http://www.cs.utep.edu/interval-comp/).

Classes, methods, functions, macros, constants and variables available in the library but not described in this document are likely to change or to be removed. Consequently, they should be used with caution, if at all.

[†] For those readers who are not native English speakers, "gaol" should be pronounced jāl, like the word "jail", of which it is a, chiefly British, variant.

[2] David Marc Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991.

[4] IEEE. IEEE standard for binary floating-point arithmetic. Technical Report IEEE Std 754-1985, Institute of Electrical and Electronics Engineers, 1985. Reaffirmed 1990.

[5] Ramon Edgar Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N. J., 1966.

[1] Götz Alefeld and Jürgen Herzberger. *Introduction to Interval Computations*. Academic Press Inc., New York, USA, 1983. Traduit par Jon Rokne de l'original Allemand *'Einführung In Die Intervallrechnung'*.

[6] Arnold Neumaier. *Interval methods for systems of equations*, volume 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1990.

2 Installation

The installation procedure differs depending on your platform. The current release of gaol is supported on the following platforms:

- ix86-based computers and compatibles under Linux with GNU gcc/g++. Both 32 bits and 64 bits operating systems are supported;
- ix86_64-based computers under Mac OS X (gaol has been successfully tested on *Darwin*).

Gaol used to be available on UltraSparc-based computers under SUN Solaris 2.[5–8] with GNU gcc/g++. With no such architecture at hand anymore, it is no longer actively developed on it, though its support should not require too much work.

2.1 Getting the software

The official web page for gaol is http://sourceforge.net/projects/gaol/.

The latest versions of gaol come as a source-code tarball only, which should be compiled and installed specifically for the computer you intend to use. There are no precompiled versions.

2.2 Installing gaol from the source tarball on Unix and Linux

Installing gaol from the source archive is done in three steps, in accordance with the spirit of all GNU softwares: *configuration, building, installation*. These steps are described hereunder. In the following, the base directory of the gaol distribution as created by decompressing the archive will be referred as the *root directory of the distribution* (or simply, the root directory).

2.2.1 Prerequisites

In order to build and install gaol, you will need the following tools. Some of them are mandatory, some are only required if you intend to modify the code, and others are only optional (their absence will not prevent you from using gaol though some features might be unavailable).

Mandatory tools and programs

Gaol uses many features provided by the ANSI-standard ISO/IEC FDIS 14882 for the C++ language. As a consequence, you will need a recent C++ compiler in order to compile gaol—e.g. gcc 3.0 or above.

Gaol relies either on IBM APMathlib or CRlibm floating-point arithmetic libraries. They must be properly installed on your system prior to configuring gaol. For your convenience, an archive of APMathlib is available on the gaol web site (`http://sourceforge.net/projects/gaol`). The CRlibm library is available on LIPForge (`http://lipforge.ens-lyon.fr/projects/crlibm/`).

Important note. Some assumptions are made concerning the accuracy of some functions provided by the standard mathematical library (`libm`). In particular, depending on the library chosen (`apmathlib` or `crlibm`), the following functions are considered to be computed to at most one ulp of accuracy:

Functions	**APmathlib**	**CRlibm**
cosh	•	
sinh	•	
tanh	•	•
acosh	•	•
asinh	•	•
atanh	•	•

Tools for maintainers

Gaol uses code produced by GNU Flex and GNU Bison for parsing the expression used to initialize an interval.

Any modification of the files

`gaol_interval_lexer.lpp`

or

`gaol_interval_parser.ypp`

requires the availability of these tools.

Optional tools

- `dot`. This program is used by `doxygen` (see below) to draw dependency graphs in the HTML documentation. It is part of the GraphViz package (`http://www.research.att.com/sw/tools/graphviz/`);
- `doxygen`. Tool similar to SUN Javadoc for the C++ language. It is available at `http://www.stack.nl/~dimitri/doxygen/index.html`. If you do not have it, you will not be able to regenerate the HTML documentation;
- `CppUnit`. This library for unit testing is available on SourceForge (`http://sourceforge.net/projects/cppunit`). It is required to test the proper compilation of gaol.

2.2.2 Configuration

Before actually compiling the library, you have to configure it for your platform by using the `configure` program located at the root of the gaol distribution. It accepts the following options:

- `--help`. Displays a list of all options. Note that only those described hereunder are supported;
- `--prefix=`*`prefix-dir`*. The root directory where the library will be installed. It defaults to `/usr/local`;
- `--libdir=`*`lib-dir`*. The directory where to put the libraries. It defaults to *`prefix-dir`*`/lib`;
- `--includedir=`*`include-dir`*. The directory where to put the header files. It defaults to *`prefix-dir`*`/include`;
- `--infodir=`*`info-dir`*. The directory where to put the documentation in `info` format. It defaults to *`prefix-dir`*`/info`;
- `--enable-shared[=`*`yes/no`*`]`. Creates or not a shared library. This option defaults to `yes` whenever shared libraries are supported by the current platform;
- `--with-mathlib=(`*`apmathlib|crlibm`*`)`. Specifies which mathematical library to use. The following libraries are currently supported.

  `apmathlib`. The IBM APMathlib library. This library must be installed and available before configuring *gaol*;

  `crlibm`. The *Correcty Rounded Mathematical library* available at `http://lipforge.ens-lyon.fr/www/crlibm/`. This library should be installed and available before configuring *gaol*.

  This option defaults to "`apmathlib`."
- `--with-mathlib-include=PATH`. Set the path where are the headers for the mathematical library to PATH;
- `--with-mathlib-lib=PATH`. Set the path where is the library file for the mathematical library to PATH;
- `--with-cppunit-include=PATH`. Set the path where are the headers for cppunit to PATH;
- `--with-cppunit-lib=PATH`. Set the path where is the library file for cppunit to PATH;
- `--enable-debug[=`*`yes/no`*`]`. Adds or not debugging information to the library. Enable the use of debugging macros (see Section 16, page 61). This option defaults to `no`.
- `--enable-preserve-rounding[=`*`yes/no`*`]`. The library assumes that the rounding direction is never modified outside of gaol, which allows to set it once and for all to "upward" at initialization (see Section 3.1, page 10). This option defaults to `no`. You should define this option to `yes` if you use gaol together with libraries or programs that manipulate the rounding direction, or that require the rounding direction to be to the nearest;

- `--enable-optimize[=yes/no]`. Compiles gaol with full optimization turned on. This option defaults to `yes`;
- `--enable-fast-math[=yes/no]`. Compiles gaol with fast but less accurate transcendental and power operators. This option defaults to `yes`;
- `--enable-exceptions[=yes/no]`. If enabled, errors should be reported by throwing an exception (see Section 15.1, page 58). If disabled, errors are reported by calling `gaol_error()`, which prints a message to the standard error channel. This option defaults to `yes`;
- `--enable-asm[=yes/no]`. Allows the use of assembler code in some parts of gaol. For most platforms, assembler code is used only to switch thr rounding direction of the FPU. On ix86, assembler code is also used for many primitives. This option defaults to `yes`. On ix86, disabling the assembler support should be done with caution, depending on the propensity of of your compiler to wrongly optimize code using floating-point instructions (which is high for versions of `gcc` prior to 4.1);
- `--enable-relations=kind`. Defines the kind of relation to use for relational symbols ($=, \leqslant, \ldots$) to be *kind*. Possible values for *kind* are `certainly`, `set`, and `possibly` (see Section 7, page 21). The default is `certainly`;
- `--verbose-mode[=yes/no]`. Allows information messages to be sent to the standard output (such as messages to report automatic initialization and cleanup). The default is `yes`.
-

Note that, as usual, `--disable-xxx` is equivalent to `--enable-xxx=no`. Moreover, `--enable-xxx` is equivalent to `--enable-xxx=yes`.

Configuration examples

First, go to the root directory. If you simply type

```
% ./configure
```

you will create a shared library with full optimization, which will be installed in `/usr/local`.
By issuing

```
% ./configure --enable-debug \
              --prefix=/usr1/local --infodir=/export/info
```

you will create a library including debugging information that will be installed in the root directory `/usr1/local` except for the info files, which will be installed in the `/export/info` directory.

Missing optional tools are reported during the configuration process, though they do not prevent you from building the library. The configuration is aborted if some important tool or library is missing.

2.2.3 Building

After having configured gaol, you can now type

```
% make
```

in the root directory to build the library and its documentation (`pdf` and `html` files).

The targets for the `Makefile` in the root directory are:

- `all`. Similar to calling `make` without any argument;
- `doc`. Create the manual in both `pdf` and `html` formats;
- `html` Create only the `html` reference;
- `check`. Test the library by compiling some benchmarks and checking their output against the expected one;
- `clean`, `distclean`, `maintainer-clean`. These are standard options for a GNU standard compliant `Makefile`. The `clean` option erases all files created during the building process; the `distclean` erases also the files created during the configuration process; `maintainer-clean` is meant to be used by maintainers only since it might erase files needing special tools to be re-created as well;
- `install`. Install the library in the directories specified at configuration time.

2.2.4 Installation

To install gaol on your system, just type

```
% make install
```

Remember that the directories you have chosen to install the libraries into, must be accessible to your compiler, i.e. they must appear in the paths contained in the relevant environment variables:

- `LIBRARY_PATH` for static libraries,
- `LD_LIBRARY_PATH` for dynamic libraries,
- ...

to be able to use the library once installed.

3 An overview of gaol

In this chapter, we will assume that gaol has already been properly installed, and that the libraries and header files are accessible to your compiler.

Let us consider the following program to compute the range of the function

$$f(x,y) = \big(1 + (x+y)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)\big) \\ \big(30 + (2x-3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)\big)$$

for $x \in [-2, 2]$ and $y \in [-2, 2]$.

Example

```
1  #include <iostream>
2  #include <gaol/gaol.h>
3
4  int main(void)
5  {
6    gaol::init();
7
8    interval
9      x(-2,2),
10     y(-2,2), z;
11
12   z=(1+sqr(x+y)*(19-14*x+3*sqr(x)-14*y+6*x*y+3*sqr(y)))*
13     (30+sqr(2*x-3*y)*(18-32*x+12*sqr(x)+
14        48*y-36*x*y+27*sqr(y)));
15
16   std::cout << "z = " << z << std::endl;
17
18   gaol::cleanup();
19   return 0;
20 }
```

First, note that we have to include the `gaol/gaol.h` header file in order to use all the facilities provided by gaol. All the functions, classes, constants and types defined in gaol are embedded into the `gaol` namespace. The `gaol/gaol.h` header imports the whole namespace such that it is not necessary to use the `gaol` prefix. Alternatively, you may include the `gaol/gaol` header instead of `gaol/gaol.h` and add `using` directives to only import what you actually need.

The call to `gaol::init()` on Line 6 is related to the use of the so-called *trust rounding mode* (see next section): it switches the rounding mode of the floating-point unit towards $+\infty$ (if the library was compiled with `--preserve-rounding=no`) and calls some initialization code.

The `sqr(x)` function stands for *square of x* and is equivalent to `pow(x,2)`.

Let `f.cpp` be the name of the file containing the program above. To compile it with `g++`, we have to type the following command:

```
% g++ -o f f.cpp -lm -lultim -lgaol
```

were `ultim` corresponds to the APMathlib library.

We thus create the executable file `f`, using the `gaol` and `APMathlib` libraries—`APMathlib` is the *Accurate Portable Mathematical library* developed by IBM; it provides us with correctly rounded mathematical functions if they are not directly available on the given platform.

Executing `f`, we obtain:

Output

```
z = [−56254330, 94177270]
```

We then know that $f(x, y)$ ranges over $[-56254330, 94177270]$ when x and y range over $[-2, 2]$ independently.

3.1 The trust rounding mode

Floating-point interval arithmetic requires *outward rounding* in order to fulfill the *containment property*: for example, to add intervals $[a, b]$ and $[c, d]$, we compute $[\downarrow a + c \downarrow, \uparrow b + d \uparrow]$, where $\downarrow r \downarrow$ and $\uparrow r \uparrow$ return the greatest (resp. smallest) floating-point number smaller (resp. greater) than the real result of r. These two operations are performed by switching the rounding direction of the FPU towards, respectively, $-\infty$ and $+\infty$.

Definition 1 (Rounding down/up) *Given $\mathbb{R}$ the set of real numbers and $\mathbb{F}$ the set of floating-point numbers (`double`), we have:*

$$\forall x \in \mathbb{R}: \begin{cases} \downarrow x \downarrow = \max\{y \in \mathbb{F} \mid y \leqslant x\} \\ \uparrow x \uparrow = \min\{y \in \mathbb{F} \mid y \geqslant x\} \end{cases}$$

On most platforms, switching the rounding direction is costly. However, it is possible to cut down the number of switches by relying on the property that $\downarrow -r \downarrow = -\uparrow r \uparrow$. Consequently, one can replace nearly all downward rounding operations by upward rounding ones by negating appropriately twice the operations performed. The next step is then to only switch once and for all the rounding direction towards $+\infty$ at the beginning of a computation. This strategy reduces drastically the number of rounding direction switches at the cost of putting on the user the burden to ensure that the rounding direction be always set towards $+\infty$ before any computation involving intervals. This mode is called the *trust rounding mode* since we trust the user for ensuring that the rounding direction is always properly set.

Basically, all the user has to do when using the trust rounding mode is to switch the rounding direction towards $+\infty$ at the beginning of his program (this is performed automatically by `gaol::init()`), and then ensuring that it always remains set to that direction before performing any interval operation. This condition is never violated by any function or method of the library itself.

When gaol is used in a larger application that also relies on libraries that assume the rounding direction to be to the nearest, it is easier and safer to configure it with the trust rounding mode disabled (see Section 2.2.2, page 5).

3.2 Common errors

In this section, we will review common errors made when using gaol in the—forlorn?—hope that it will help prevent users from making them.

3.2.1 Floating-point arithmetic and rounding

Programming with floating-point numbers is one of the few activities where one must always consider ones compiler defiantly. For example, let us consider the following piece of code:

Example

```
#include <gaol/gaol>  // We do not import the gaol namespace

using gaol::interval;

int main(void) {
  gaol::init();
  interval one_tenth(0.1); BEWARE: wrong !

  [some code using one_tenth]
  gaol::cleanup();
}
```

Though a rational perfectly representable in decimal, `0.1` is not representable in binary (at least, not with a finite number of bits) and thus requires rounding. Obviously, the purpose of the user was to define an interval containing this value. However, `0.1` will be rounded *at compile time*, most certainly to the nearest representable floating-point number $\updownarrow 0.1 \updownarrow$. As a consequence, `one_tenth` will be a degenerate interval containing only $\updownarrow 0.1 \updownarrow$, and the *containment property* will be violated.

The right way to deal with rational constants that might not be perfectly representable as floating-point numbers is to stringigy them, such that they can be correctly rounded downward and upward at *runtime*:

Example

```
#include <gaol/gaol>

using gaol::interval;

int main(void)
{
  gaol::init();
  interval one_tenth("0.1"), // OK: this is the right way
           one_tenth2("1/10"); // Another possible way

  [some code using one_tenth]
  gaol::cleanup();
  return 0;
}
```

Now, `one_tenth` will be the smallest floating-point interval enclosing `0.1`. An interval like this one, containing at most two consecutive floating-point numbers, is called a *canonical interval*.

Definition 2 (Rounding to the nearest) *Given* $\mathbb{R}$ *the set of real numbers and* $\mathbb{F}$ *the set of floating-point numbers (`double`), we have:*

$$\forall x \in \mathbb{R}\colon \updownarrow x \updownarrow = y \in \mathbb{F} \text{ s.t. } x - y = \min\{|\delta| \in \mathbb{R} \mid \delta = x - z, \forall z \in \mathbb{F}\}$$

Definition 3 (canonical interval) *A non-empty interval* $I = [a, b]$ *is* canonical *if and only if* $a \geqslant b^{-}$.

4 Initialization and cleanup

The following functions have to be called before using any functionality of the library and just after having used it for the last time.

Since Release 1.0 of gaol, there is an automatic initialization/cleanup feature that ensures that no problem will arise if the user forgets to explicitly call these functions.

bool **init** (int *dbg_lvl* = 0)

Initializes the variable `debug_level` (see Section 16, page 61) to the value of *dbg_lvl*.
If the library was not compiled with the `--enable-preserve-rounding` option, it sets the FPU control word according to the requirements of the `APMathlib` library and enforces rounding towards $+\infty$. In addition, it sets the number of digits to display for interval bounds to 16.
Returns `true` if the library was not already initialized and false otherwise.
After its first call, the only effect of this function is to—possibly—set the debugging level to a new value.

bool **cleanup** (void)

Restores the state of the FPU to its value prior to the initialization of the gaol library.
In the current version, returns `true` the first time it is called and false afterwards.

Example

```
1  #include <gaol/gaol>
2
3  int main(void)
4  {
5    init(1); // First level of debugging requested
6
7    [Some code using interval arithmetic]
8
9    cleanup();
10   return 0;
11 }
```

5 Interval creation and assignment

The methods for creating an interval and assigning a new value to an already existing one are described in the following.

5.1 Constructors

One can create an interval in five different ways:

- by providing its left and right bounds:

Example
```
interval x(l,r);
```

  where *l* and *r* are `double`s or of a type that is castable into a `double`;

- by providing only one bound, for degenerate point intervals:

Example
```
interval x(v);
```

  This is equivalent to: `interval x(v,v);`

- without providing any bound:

Example
```
interval x;
```

  This is equivalent to:

Example
```
interval x(-GAOL_INFINITY,+GAOL_INFINITY);
```

  where `GAOL_INFINITY` represents the infinity value of the `double` format (see Section 11.1, page 49);

- by copying an already existing interval (*copy constructor*):

Example
```
interval x(-12,12), y=x, z(x);
```

- by using a string representing an interval in the same format as the one used for input (see Section 10.1.1, page 43)

Example

```
interval x("[-23, inf]"),
         y("[5*0.1+dmin, 89*sinh(2.1)]");
```

If the input string does not comply with the expected format, an empty interval is returned. The exception `gaol::input_format_error` is raised if the library was compiled with exception support; in the absence of exception support, the `gaol_error()` function is called to print an error message, and the `errno` variable is set to `-1` (`errno` is *not* modified when no error occurs).

- by using a string representing an interval in the same format as the one used for input (see Section 10.1.1, page 43) for each bound:

Example

```
// constructs x=[-4,2]
interval x("[-5,4]+1","[4,6]-[3,2]");
```

Caution. you have to be very careful when creating an interval from floating-point constants. Remember that a rational number that is perfectly representable in the decimal base may require rounding in the binary base. For example, if you write the following statement:

Example

```
interval x(0.1);
```

you will *not* have created an interval containing 0.1 since this number has an infinite expansion in the binary base (i.e. it is impossible to represent it perfectly whatever the size of the mantissa may be). As a consequence, the constant 0.1 has very likely been rounded to the nearest floating-point number at compile-time. In such a case, you have to use a string instead:

Example

```
interval x("0.1");
```

Note. For performance reasons, interval constructors do not forbid to create intervals whose bounds are infinities of the same sign:

Example

```
interval I(GAOL_INFINITY,GAOL_INFINITY);
cout << I << endl;
<inf, inf>
```

However, such intervals are neither allowed nor supported in interval computation. They are not considered empty intervals and their use with interval operators leads to inconsistent results.

As an exception, creating an interval from the `inf` string constant leads to a legitimate interval:

Example

```
interval I("[-inf, -inf]");
interval J("<-inf, -inf>");
interval K("inf");
cout << I << endl;
```

```
cout << J << endl;
cout << K << endl;
[-inf, -1.797693134862316e+308]
[-inf, -1.797693134862316e+308]
[1.797693134862316e+308, inf]
```

5.2 Straight assignment

It is possible to assign a new value to an already existing interval in three different ways:

- by copying another interval:

Example

```
    interval x(-12,12),
             y; // Here, y is [−∞ + ∞, ]
    y = x; // Now, y is [−12, 12]
```

- by using a string whose format follows the one expected for input (see Section 10.1, page 43).

Example

```
interval x;
x = "[-inf, 123]";
```

- by using a `double`:

Example

```
interval x;
x = 1234.5; // Now, x is [1234.5, 1234.5]
```

Note that there is no method for modifying a bound of an interval since intervals must be considered as an atomic concept.

5.3 Assignment combined with an operation

The following assignment operators combine the value of the interval pointed to by `self` and the value of the right-hand side interval.

In this manual, we note `self` the object to which `this` is a pointer in C++.

interval& interval::**operator&=** (const interval& *I*)

■ self ← self ∩ *I*

Assigns to `self` the interval resulting from the intersection of `self` and *I*.

Example

```
interval x(-12,12);
x &= interval(-6,23); // Now, x is [−6, 12]
```

interval& interval::**operator|=** (const interval& *I*)

■ self ← self ∪ *I*

Assigns to `self` the interval resulting from the union of `self` and *I*.

Example

```
  interval x(-12,12);
  x |= interval(-6,23); // Now, x is [−12, 23]
```

interval& interval::**operator+=** (const interval& *I*)
interval& interval::**operator+=** (double *d*)

- `self` $\leftarrow$ `self` $+ I$
- `self` $\leftarrow$ `self` $+ d$

Assigns to `self` the interval resulting from adding `self` and I (resp. d).

Example

```
interval x(-12,12);
x += interval(-6,23); // Now, x is [−18, 35]
```

interval& interval::**operator-=** (const interval& *I*)
interval& interval::**operator-=** (double *d*)

- `self` $\leftarrow$ `self` $- I$
- `self` $\leftarrow$ `self` $- d$

Assigns to `self` the interval resulting from subtracting I (resp. d) from `self`.

Example

```
interval x(-12,12);
x -= interval(-6,23); // Now, x is [−35, 18]
```

interval& interval::**operator*=** (const interval& *I*)
interval& interval::**operator*=** (double *d*)

- `self` $\leftarrow$ `self` $\times I$
- `self` $\leftarrow$ `self` $\times d$

Assigns to `self` the interval resulting from multiplying `self` and I (resp. d).

Example

```
interval x(-12,12);
x *= interval(-6,23); // Now, x is [−276, 276]
```

interval& interval::**operator/=** (const interval& *I*)
interval& interval::**operator/=** (double *d*)

- `self` $\leftarrow$ `self`$/I$
- `self` $\leftarrow$ `self`$/d$

Assigns to `self` the interval resulting from dividing `self` by I (resp. d).

Example

```
interval x(-12,12);
x /= interval(-6,23); // Now, x is [−∞, +∞]
x /= interval::zero;  // Now, x is ∅
```

interval& interval::**operator%=** (const interval& *I*)
interval& interval::**operator%=** (double *d*)

Assigns to `self` the interval resulting from dividing `self` by I (resp. d), using a *relational division* (see Section 8.2, page 34).

Example

```
interval x(-12,12);
x %= interval(-6,23); // Now, x is [−∞, +∞]
x %= interval::zero;  // Now, x is [−∞, +∞]
```

6 Interval constants

For convenience, some useful intervals and some canonical intervals enclosing real constants are defined as static functions of the `interval` class:

Function	Value
`interval::emptyset()`	$\varnothing$
`interval::half_pi()`	$[\downarrow \frac{\pi}{2} \downarrow, \uparrow \frac{\pi}{2} \uparrow]$
`interval::minus_one_plus_one()`	$[-1, 1]$
`interval::negative()`	$[-\infty, 0]$
`interval::one()`	1
`interval::one_plus_infinity()`	$[1, +\infty]$
`interval::pi()`	$[\downarrow \pi \downarrow, \uparrow \pi \uparrow]$
`interval::positive()`	$[0, +\infty]$
`interval::two_pi()`	$[\downarrow 2\pi \downarrow, \uparrow 2\pi \uparrow]$
`interval::universe()`	$[-\infty, +\infty]$
`interval::zero()`	0

Example

```
cout << interval::emptyset;
[empty]
```

7 Interval relations

Interval relations may be divided into three groups. Given I and J two intervals, we have:

1. *set relations*: intervals I and J are considered as sets of reals. For example:

   $$I = J \Leftrightarrow (\forall x \in I, \exists y \in J : x = y) \wedge (\forall y \in J, \exists x \in I : x = y)$$

   Basically, two intervals are equal in that mode if they have the same bounds;

2. *certainly relations*: the relations must be true for any tuple of values in the intervals. For example:

   $$I = J \Leftrightarrow (\forall x \in I, \forall y \in J : x = y)$$

   Then, two intervals are equal in that mode if they are both reduced to the same value;

3. *possibly relations*: the relations are true if it exists at least one tuple verifying the corresponding real relation. For example:

   $$I = J \Leftrightarrow (\exists x \in I, \exists y \in J : x = y)$$

   Then, two intervals are equal in that mode whenever their intersection is not empty.

The kind of relation to associate to relation symbols such as $==$ and $\leqslant$ is chosen when configuring the library (see Section 2.2.2, page 5). The other possible definitions are always available through the methods described hereunder.

7.1 Set relations

bool interval::**set_contains** (const interval& *I*) const
bool interval::**set_contains** (double *d*) const

- $I \subseteq$ self
- $d \in$ self

Returns `true` if I (resp. $\{d\}$) is included in `self`.

Example

```
interval x(-12,34), y(-12,5);

cout << x.set_contains(y) << endl;
cout << x.set_contains(interval::emptyset) << ' '
     << y.set_contains(x) << ' '
     << interval::emptyset.set_contains(x) << ' '
     << interval::emptyset.set_contains(interval::emptyset)
     << endl;
true true false false true
```

bool interval::**set_strictly_contains** (const interval *I*) const
bool interval::**set_strictly_contains** (double *d*) const

- $I \subset$ self
- $d \in$ self

Returns `true` if I (resp. $\{d\}$) is strictly included in `self`

Example

```
interval x(-10,12), y(-10, 11), z, t, u(10.5);

cout << boolalpha
    << x.set_strictly_contains(y) << ' '
    << z.strictly_contains(t)
    << ' ' << x.set_strictly_contains(u) << ' '
    << interval::emptyset.set_strictly_contains(
                                    interval::emptyset)
    << ' ' << u.set_strictly_contains(
                           interval::emptyset) << endl;
false false true true true
```

bool interval::**set_disjoint** (const interval& *I*) const

- $*this \cap I = \varnothing$

Returns `true` if the intersection of `self` and I is empty.

Example

```
interval a(2,4), b(6,dmax);
cout << a.set_disjoint(b) << " "
     << interval::emptyset.set_disjoint(interval::emptyset);
true true
```

bool interval::**set_eq** (const interval& *I*) const

- $\forall x \in$ self$\exists y \in I : x = y \wedge \forall y \in I, \exists x \in$ self$: y = x$

Returns `true` if intervals `self` and I are equal when considered as sets of reals.

Example

```
cout << interval(4,dmax).set_eq(interval(4,dmax)) << " "
     << interval::emptyset.set_eq(interval::empty_set) << endl;
true true
```

bool interval::**set_neq** (const interval& *I*) const

■ $\exists x \in \texttt{self} \forall y \in I : x \neq y \vee \exists y \in I \forall x \in \texttt{self} : y \neq x$

Returns `true` if `self` and I are not equal when considered as sets of reals.

Example

```
cout << interval::universe.set_neq(interval::emptyset);
🖶 true
```

bool interval::**set_le** (const interval& *I*) const

■ $\forall x \in \texttt{self}, \exists y \in I : x < y \wedge \forall y \in I, \exists x \in \texttt{self} : y > x$

Returns `true` if the real set defined by `self` is strictly included in `I`.

Example

```
cout << interval(-4.5,3).set_le(interval(-10,10))
     << interval::emptyset.set_le(interval(5,6))
     << interval::emptyset.set_le(interval::emptyset);
🖶 true false
```

bool interval::**set_leq** (const interval& *I*) const

■ $\forall x \in \texttt{self}, \exists y \in I : x \leqslant y \wedge \forall y \in I, \exists x \in \texttt{self} : y \geqslant x$

Returns `true` if the real set defined by `self` is included in `I`.

Example

```
cout << interval(4.5,6).set_leq(interval(4.5,6))
     << interval(3.5,9).set_leq(interval(2,6))
     << interval::emptyset.set_leq(interval::emptyset);
🖶 true false true
```

bool interval::**set_ge** (const interval& *I*) const

■ $\forall x \in \texttt{self}, \exists y \in I : x > y \wedge \forall y \in I, \exists x \in \texttt{self} : y < x$

Returns `true` if the real set defined by `self` strictly contains `I`.

bool interval::**set_geq** (const interval& *I*) const

■ $\forall x \in \texttt{self}, \exists y \in I : x \geqslant y \wedge \forall y \in I, \exists x \in \texttt{self} : y \leqslant x$

Returns `true` if the real set defined by `self` contains `I`.

7.2 Certainly relations

bool interval::**certainly_eq** (const interval& *I*) const

■ $\forall x \in \texttt{self}, \forall y \in I : x = y$

Returns `true` if `self` is certainly equal to I, which is true only when both intervals are degenerate and contain the same floating-point number.

Example

```
cout << interval(3,4).certainly_eq(interval(3,4))
     << interval(-6).certainly_eq(interval(-6,-6))
     << interval::universe.certainly_eq(interval::universe)
     << interval::emptyset.certainly_eq(interval::emptyset)
🖶 false true false true
```

bool interval::**certainly_neq** (const interval& *I*) const

- $\forall x \in \texttt{self}, \forall y \in I\text{: } x \neq y$

Returns `true` if `self` is certainly not equal to I.

bool interval::**certainly_le** (const interval& *I*) const

- $\forall x \in \texttt{self}, \forall y \in I\text{: } x < y$

Returns `true` if `self` is certainly strictly less than I.

Example

```
cout << interval(4,5).certainly_le(interval(6,9))
     << interval(4,5).certainly_le(interval(5,9))
     << interval::emptyset.certainly_le(interval(4,6));
true false true
```

bool interval::**certainly_leq** (const interval& *I*) const

- $\forall x \in \texttt{self}, \forall y \in I\text{: } x \leqslant y$

Returns `true` if `self` is certainly less or equal to I

Example

```
cout << interval(4,5).certainly_leq(interval(6,9))
     << interval(4,5).certainly_leq(interval(5,9))
     << interval(5,9).certainly_leq(interval(4,5))
     << interval(4,8).certainly_leq(interval(5,9))
     << interval::emptyset.certainly_leq(interval(4,6));
true true false false true
```

bool interval::**certainly_ge** (const interval& *I*) const

- $\forall x \in \texttt{self}, \forall y \in I\text{: } x > y$

Returns `true` if `self` is certainly strictly greater than I

Example

```
cout << interval(8,10).certainly_ge(interval(4,8))
     << interval::emptyset.certainly_ge(interval::emptyset);
false true
```

bool interval::**certainly_geq** (const interval& *I*) const

- $\forall x \in \texttt{self}, \forall y \in I\text{: } x \geqslant y$

Returns `true` if `self` is certainly greater or equal to I.

Example

```
cout << interval(8,10).certainly_geq(interval(4,8))
     << interval::emptyset.certainly_geq(interval::emptyset);
true true
```

bool interval::**certainly_positive** (void) const

- $\texttt{self} \subseteq [0, +\infty]$

Returns `true` if `self` lower bound is greater or equal to zero.

Example

```
cout << interval::emptyset.certainly_positive()
     << interval(4,5).certainly_positive()
     << interval(-0.0,6).certainly_positive()
     << interval(-6,0).certainly_positive();
true true true false
```

bool interval::**certainly_strictly_positive** (void) const

- self $\subset [0, +\infty]$

Returns `true` if `self` lower bound is strictly greater than zero.

Example

```
cout << interval::emptyset.certainly_strictly_positive()
     << interval(4,5).certainly_strictly_positive()
     << interval(-0.0,6).certainly_strictly_positive()
     << interval(-6,0).certainly_strictly_positive();
🖶 true true false false
```

bool interval::**certainly_negative** (void) const

- self $\subseteq [-\infty, 0]$

Returns `true` if `self` lower bound is lower or equal to zero.

Example

```
cout << interval::emptyset.certainly_negative()
     << interval(4,5).certainly_negative()
     << interval(-6,0).certainly_negative()
     << interval(-6,-5).certainly_negative();
🖶 true false true true
```

bool interval::**certainly_strictly_negative** (void) const

- self $\subset [-\infty, 0]$

Returns `true` if `self` lower bound is strictly lower than zero.

Example

```
cout << interval::emptyset.certainly_strictly_negative()
     << interval(4,5).certainly_strictly_negative()
     << interval(-6,0).certainly_strictly_negative()
     << interval(-6,-5).certainly_strictly_negative();
🖶 true false false true
```

7.3 Possibly relations

bool interval::**possibly_eq** (const interval& *I*) const

- $\exists x \in$ self, $\exists y \in I$: $x = y$

Returns `true` if `self` is possibly equal to I.

Example

```
cout << interval(5,10).possibly_eq(interval(6,100))
     << interval::emptyset.possibly_eq(interval::emptyset);
🖶 true false
```

bool interval::**possibly_neq** (const interval& *I*) const

- $\exists x \in$ self, $\exists y \in I$: $x \neq y$

Returns `true` if `self` is possibly not equal to I

Example

```
cout << interval(4,5).possibly_neq(interval(4,5))
     << interval(4,4).possibly_eq(interval(4,4))
     << interval::emptyset.possibly_eq(interval::emptyset);
🖶 true false false
```

bool interval::**possibly_le** (const interval& *I*) const

■ $\exists x \in$ `self`, $\exists y \in I$: $x < y$

Returns `true` if `self` is possibly strictly less than *I*.

Example

```
cout << interval(4,5).possibly_le(interval(3,7))
     << interval(4,5).possibly_le(interval(2,4))
     << ;interval(4,5).possibly_le(interval::emptyset)
true false false
```

bool interval::**possibly_leq** (const interval& *I*) const

■ $\exists x \in$ `self`, $\exists y \in I$: $x \leqslant y$

Returns `true` if `self` is possibly less or equal to *I*.

Example

```
cout << interval(4,5).possibly_leq(interval(3,7))
     << interval(4,5).possibly_leq(interval(2,4))
     << ;interval(4,5).possibly_leq(interval::emptyset)
true true false
```

bool interval::**possibly_ge** (const interval& *I*) const

■ $\exists x \in$ `self`, $\exists y \in I$: $x > y$

Returns `true` if `self` is possibly strictly greater than *I*

Example

```
cout << interval(4,5).possibly_ge(interval(3,6))
     << interval(4,5).possibly_ge(interval(5,6))
     << interval(4,5).possibly_ge(interval(6,7))
     << interval(4,5).possibly_ge(interval::emptyset);
true false false false false
```

bool interval::**possibly_geq** (const interval& *I*) const

■ $\exists x \in$ `self`, $\exists y \in I$: $x \geqslant y$

Returns `true` if `self` is possibly greater or equal to *I*.

Example

```
cout << interval(4,5).possibly_geq(interval(3,6))
     << interval(4,5).possibly_geq(interval(5,6))
     << interval(4,5).possibly_geq(interval(6,7))
     << interval(4,5).possibly_geq(interval::emptyset);
true true false false false
```

7.4 Relational Symbols

bool **operator==** (const interval& *I1*, const interval& *I2*)

Returns `I1.set_eq(I2)`, `I1.certainly_eq(I2)`, or `I1.possibly_eq(I2)` depending on the default kind of relation chosen when configuring gaol (see Section 2.2.2, page 5).

bool **operator!=** (const interval& *I1*, const interval& *I2*)

Returns `I1.set_neq(I2)`, `I1.certainly_neq(I2)`, or `I1.possibly_neq(I2)` depending on the default kind of relation chosen when configuring gaol (see Section 2.2.2, page 5).

bool **operator<** (const interval& *I1*, const interval& *I2*)

Returns `I1.set_le(I2)`, `I1.certainly_le(I2)`, or `I1.possibly_le(I2)` depending on the default kind of relation chosen when configuring gaol (see Section 2.2.2, page 5).

bool **operator<=** (const interval& *I1*, const interval& *I2*)

Returns `I1.set_leq(I2)`, `I1.certainly_leq(I2)`, or `I1.possibly_leq(I2)` depending on the default kind of relation chosen when configuring gaol (see Section 2.2.2, page 5).

bool **operator>** (const interval& *I1*, const interval& *I2*)

Returns `I1.set_ge(I2)`, `I1.certainly_ge(I2)`, or `I1.possibly_ge(I2)` depending on the default kind of relation chosen when configuring gaol (see Section 2.2.2, page 5).

bool **operator>=** (const interval& *I1*, const interval& *I2*)

Returns `I1.set_geq(I2)`, `I1.certainly_geq(I2)`, or `I1.possibly_geq(I2)` depending on the default kind of relation chosen when configuring gaol (see Section 2.2.2, page 5).

7.5 Interval-specific relations

bool interval::**straddles_zero** (void) const

■ $0 \in$ `self`

Returns `true` if `self` contains zero.

Example

```
1 interval x(0,4), y, z(-12,-5);
2
3 cout << boolalpha << x.straddles_zero() << ' '
4      << y.straddles_zero() << ' '
5      << z.straddles_zero() << endl;
6 🖶 true true false
```

Note. `I.straddles_zero()` ≡ `I.set_contains(interval::Real(0))`

bool interval::**strictly_straddles_zero** (void) const

■ $\{0\} \subset$ self

Returns `true` if zero is included in the interior of `self`.

Example

```
interval x(0,4), y, z(-12,-5);

cout << boolalpha << x.strictly_straddles_zero() << ' '
     << y.strictly_straddles_zero() << ' '
     << z.strictly_straddles_zero() << endl;
```

```
false true false
```

Note. The method call to `I.strictly_straddles_zero()` is equivalent to `I.set_strictly_contains(interval::Real(0))`

bool interval::**is_a_double** (void) const

Returns `true` whenever the left and right bounds of the interval are equal.

Example

```
interval x(-12.5), y;

cout << boolalpha << x.is_a_double() << ' '
     << y.is_a_double()
     << interval::emptyset.is_a_double() << endl;
```

```
true false false
```

bool interval::**is_an_int** (void) const

Returns `true` whenever the left and right bounds of the interval are equal and castable into an integer (type `int`).

Example

```
interval x(-12.0), y;

cout << boolalpha << x.is_an_int() << ' ' << y.is_an_int()
     << interval::emptyset.is_an_int() << endl;
```

```
true false false
```

bool interval::**is_canonical** (void) const

Returns `true` if `self` contains at most two floating-point numbers.

Example

```
interval x(0), y(-14,9);

cout << boolalpha << x.is_canonical()      << ' ' <<
                     y.is_canonical()      << ' ' <<
                     interval::emptyset << ' ' <<
                     interval::pi;
```

```
true false false true
```

bool interval::**is_empty** (void) const

■ self = ∅

Returns `true` if `self` is an empty interval.

Example

```
cout << interval(4,5).is_empty()
     << interval(5,4).is_empty()
     << interval::emptyset.is_empty();
🖶 false true true
```

bool interval::**is_zero** (void) const

■ self = [0, 0]

Returns `true` if `self` is equal to the interval containing only 0.

Example

```
cout << interval::zero.is_zero()
     << interval(0.0,0.0).is_zero()
     << interval(-0.0,+0.0).is_zero()
     << interval(0,5).is_zero()
     << interval::emptyset.is_zero();
🖶 true true true false false
```

bool interval::**is_symmetric** (void) const

Returns `true` if the left bound of `self` is the opposite of the right bound. An empty interval is not symmetric.

Example

```
cout << interval(-5,5).is_symmetric()
     << interval::emptyset.is_symmetric() << endl;
🖶 true false
```

bool interval::**is_finite** (void) const

Returns `true` if both bounds are finite.

Example

```
cout << interval("[4,inf]").is_finite()
     << interval::emptyset.is_finite()
     << interval(5,80).is_finite()
🖶 false true true
```

8 Interval Arithmetic

The *containment principle* of (floating-point) interval arithmetic imposes that for any operation "$\circ$", and any intervals I and J, the following does hold:

$$I \circ J = \square\{i \circ j \mid i \in I, j \in J\}$$

where $\square$ is a function mapping any real set to the smallest floating-point interval containing it.

For example, if we consider the interval square root, we have:

$$\sqrt{I} = \square\{\sqrt{i} \mid x \in I\}$$

From monotonicity considerations, the square root of $[1,\ 2]$ is then $\sqrt{[1,\ 2]} = [1,\ \sqrt{2}]$. Now, another interpretation of the square root function is as follows:

$$\sqrt{}_r I = \square\{j \in \mathbb{R} \mid \exists i \in I\colon j^2 = i\}$$

This last definition stands for the *relational square root* and permits obtaining both negative and positive values. Hence, we have:

$$\sqrt{}_r[1,\ 2] = [-\sqrt{2},\ \sqrt{2}]$$

This operator arises when we consider the relation

$$x^2 = y$$

which can alternatively be written

$$x = \sqrt{}_r y$$

Here, the *functional square root* is not suitable since it would induce the intersection of the domain of y with $[0,\ +\infty]$.

Some applications (mainly in the area of *constraint programming*) require the availability of such operators. As a consequence, gaol offers both functional and relational versions of the main arithmetic operators.

8.1 Functional Arithmetic

interval interval::**operator+** (void) const
interval **operator+** (const interval& *I*, double *d*)
interval **operator+** (double *d*, const interval& *I*)
interval **operator+** (const interval& *I1*, const interval& *I2*)

Addition of two intervals, or of one interval and a double.

interval interval::**operator-** (void) const
interval **operator-** (const interval& *I*, double *d*)
interval **operator-** (double *d*, const interval& *I*)
interval **operator-** (const interval& *I1*, const interval& *I2*)

Negation, or subtraction of two intervals, or of one interval and a double.

interval **operator*** (const interval& *I*, double *d*)
interval **operator*** (double *d*, const interval& *I*)
interval **operator*** (const interval& *I1*, const interval& *I2*)

Multiplication of two intervals, or of one interval and a double.

interval **operator/** (const interval& *I*, double *d*)
interval **operator/** (double *d*, const interval& *I*)
interval **operator/** (const interval& *I1*, const interval& *I2*)

Functional division of two intervals, or of one interval and a double.

interval **operator%** (const interval& *I*, double *d*)
interval **operator%** (double *d*, const interval& *I*)
interval **operator%** (const interval& *I1*, const interval& *I2*)

Relational division of two intervals, or of one interval and a double.

interval **sqrt** (const interval& *I*)

- $\sqrt{I}$

Functional square root of I.

interval **sqr** (const interval& *I*)

- I^2

Square of I

interval **pow** (const interval& *I*, int *b*)
interval **pow** (const interval& *I1*, const interval& *I2*)

- I^b
- $I_1^{I_2}$

Power function. The former computes I to the integral power b, while the latter raises *I1* to the interval power *I2*. If *I2* is an `int` in disguise, the first function is used to improve accuracy.
The power function for interval exponents is not yet fully tested and should be used with care.

interval **nth_root** (const interval& *I*, int *b*)

- $\sqrt[b]{I}$

Computes the bth functional root of I.

interval **exp** (const interval& *I*)

Exponential of *I*.

interval **log** (const interval& *I*)

Natural logarithm of *I*.

8.1.1 Trigonometric functions

interval **cos** (const interval& *I*)

Returns the cosine of *I*.

interval **acos** (const interval& *I*)

Returns the arccosine of *I*.

interval **sin** (const interval& *I*)

Returns the sine of *I*.

interval **asin** (const interval& *I*)

Returns the arcsine of *I*.

interval **tan** (const interval& *I*)

Returns the tangent of *I*.

interval **atan** (const interval& *I*)

Returns the arctangent of *I*.

8.1.2 Hyperbolic functions

interval **cosh** (const interval& *I*)

Returns the hyperbolic cosine of *I*.

interval **acosh** (const interval& *I*)

Returns the hyperbolic arccosine of *I*.

interval **sinh** (const interval& *I*)

Returns the hyperbolic sine of *I*.

interval **asinh** (const interval& *I*)

Returns the hyperbolic arcsine of *I*.

interval **tanh** (const interval& *I*)

Returns the hyperbolic tangent of *I*.

interval **atanh** (const interval& *I*)

Returns the hyperbolic arctangent of *I*.

8.2 Relational Arithmetic

interval interval::**operator%** (double *d*) const
interval interval::**operator%** (const interval& *I*) const
interval **operator%** (double *d*, const interval& *I*)

- $*this/I = \{z \in \mathbb{R} \mid \exists x \in *this, \exists y \in I : x = yz\}$

  Relational division.

8.2.1 $(n+1)$-ary relational functions

Consider the relation $y = \cos x$ where x and y are interval variables. One would like to be able to express this relation in the equivalent way: $x = acos\, y$. However, one cannot use the *acos* function because its result is always included into the interval $[0, \pi]$. What we need here is a relational version of the *acos* function. But, since for any value x there are infinitely many values y verifying $x = acos\, y$, we have to take into account the domain of y. As a consequence, we define a new binary operator `acos_rel` whose definition is as follows:

$$\texttt{acos\_rel}(Y, X) = \square\{x \in X \mid \exists y \in Y : y = \cos x\}$$

This is to be contrasted with the previous definition of the *acos* function:

$$\texttt{acos}(Y) = \square\{x \in \mathbb{R} \mid \exists y \in Y : x = acos\, y\}$$

Figure 8.1 presents the different results obtained when computing either $\texttt{acos}(J)$ or $\texttt{acos\_rel}(J, I)$.

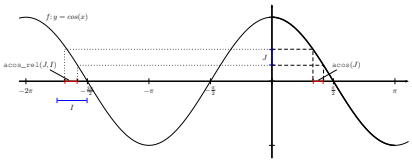

Figure 8.1: Relational cosine

interval **acos_rel** (const interval& J, const interval& I)

- $\texttt{acos\_rel}(J, I) = \square\{x \in I \mid \exists y \in J : y = \cos x\}$

  Returns the *relational arccosine* of J w.r.t. I.

interval **asin_rel** (const interval& J, const interval& I)

- $\texttt{asin\_rel}(J, I) = \square\{x \in I \mid \exists y \in J : y = \sin x\}$

  Returns the *relational arcsine* of J w.r.t. I.

interval **atan_rel** (const interval& J, const interval& I)

- $\texttt{atan\_rel}(J, I) = \square\{x \in I \mid \exists y \in J : y = \tan x\}$

  Returns the *relational arctangent* of J w.r.t. I.

interval **acosh_rel** (const interval& J, const interval& I)

- $\texttt{acosh\_rel}(J, I) = \square\{x \in I \mid \exists y \in J : y = \cosh x\}$

  Returns the *relational hyperbolic arccosine* of J w.r.t. I.

interval **sqrt_rel** (const interval& J, const interval& I)

- $\texttt{sqrt\_rel}(J, I) = \square\{x \in I \mid \exists y \in J : y = x^2\}$

  Returns the *relational square root* of J w.r.t. I.

interval **nth_root_rel** (const interval& *J*, unsigned int *n*,
const interval& *I*)

■ `nth_root_rel`$(J, n, I) = \square\{x \in I \mid \exists y \in J \colon y = x^n\}$

Returns the *relational inverse b-th root* of J w.r.t. I.

interval **invabs_rel** (const interval& J, const interval& I)

■ `invabs_rel`$(J, I) = \square\{x \in I \mid |x| \in J\}$

Returns the *relational inverse absolute value* of J w.r.t. I.

interval **div_rel** (const interval& K, const interval& J, const interval& I)

■ `div_rel`$(K, J, I) = \square\{x \in I \mid \exists z \in K, \exists y \in J \colon z = xy\}$

Returns the *ternary relational division* of K by J w.r.t. I.

Relational function that do not appear here are identical to their inverse function (e.g., the relational hyperbolic arcsine is identical to the hyperbolic arcsine, which should be used instead).

9 Interval functions

double interval::**width** (void) const

- `width`$([a, b]) =\uparrow b - a \uparrow$

Returns the width of `self`. Returns `-1.0` whenever the interval is empty.

Example

```
1 cout << interval(4,6).width()
2      << (interval(1,next_float(1)).width()
3          == std::numeric_limits<double>::epsilon())
4      << interval::emptyset.width();
5 🖶 2 true -1
```

double interval::**mig** (void) const

Returns the *mignitude* of `self`. See the book by Hansen [3, chap. 3]. The mignitude of an interval $[a, b]$ is the smallest absolute value of the numbers in the interval, that is: 0 if the interval straddles `0`, a if the interval is strictly positive, and $-b$ otherwise.

Note. The mignitude of the empty interval is a NaN.

Example

```
1 cout << interval(4,5).mig()
2      << interval(-6,-3).mig()
3      << interval(-3,8).mig();
4 🖶 4 3 0
```

double interval::**smig** (void) const

[3] Eldon Robert Hansen. *Global Optimization Using Interval Analysis.* Pure and Applied Mathematics. Marcel Dekker Inc., 1992.

[8] Volker Stahl. *Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations.* Phd. thesis, Johannes Kepler Universität, Linz, September 1995.

Returns the *signed magnitude* of `self`. See Stahl's thesis [8, def. 1.3.28]. The signed mignitude of an interval $[a, b]$ is 0 if the interval straddles `0`, a if the interval is strictly positive, and b otherwise.

Note. The signed mignitude of the empty interval is an NaN.

Example

```
cout << interval(4,5).mig()
     << interval(-6,-3).mig()
     << interval(-3,8).mig();
4 -3 0
```

double interval::**mag** (void) const

Returns the *magnitude* of `self`. the magnitude of an interval $[a, b]$ is the greatest absolute value of the numbers in the interval.

Note. The magnitude of the empty interval is a NaN.

Example

```
cout << interval(4,5).mag()
     << interval(-6,-3).mag()
     << interval(-10,5).mag();
5 6 10
```

double **hausdorff** (const interval& *I1*, const interval& *I2*)

Returns the Hausdorff distance between the two sets defined by intervals *I1* and *I2*, that is:

$$\texttt{hausdorff}(I_1, I_2) = \max(|\underline{I_1} - \underline{I_2}|, |\overline{I_1} - \overline{I_2}|)$$

Example

```
cout << hausdorff(interval(4,8),interval(5,10))
2
```

double interval::**midpoint** (void) const

Returns the midpoint of `self`. Given a and b two finite floating-point numbers and `std::numeric_limits<double>::max()` the largest positive floating-point number of type double, we have the following cases:

$$\begin{cases} \texttt{midpoint}(\varnothing) & = \text{NaN} \\ \texttt{midpoint}([-\infty, +\infty]) & = 0 \\ \texttt{midpoint}([-\infty, b]) & = -\texttt{std::numeric\_limits<double>::max()} \\ \texttt{midpoint}([a, +\infty]) & = \texttt{std::numeric\_limits<double>::max()} \\ \texttt{midpoint}([a, b]) & = \updownarrow (a+b)/2 \updownarrow \end{cases}$$

interval interval::**mid** (void) const

Returns an interval enclosing the midpoint of `self`. The result is not guaranteed to be canonical though it is always included in `self`. With the same notations as for `midpoint()`, we have the cases:

$$\begin{cases} \texttt{mid}(\varnothing) & = \varnothing \\ \texttt{mid}([-\infty, +\infty]) & = [0, 0] \\ \texttt{mid}([-\infty, b]) & = [-\texttt{std::numeric\_limits<double>::max()}] \\ \texttt{mid}([a, +\infty]) & = [\texttt{std::numeric\_limits<double>::max()}] \\ \texttt{mid}([a, b]) & = [\downarrow (a+b)/2 \downarrow, \uparrow (a+b)/2 \uparrow] \end{cases}$$

double interval::**left** (void) const

- $\texttt{left}([x, y]) = x$

Returns the left bound of `self`. Note that this method may return a finite floating-point number (i.e. neither a NaN, nor an infinity) even when the interval itself is empty.

double interval::**right** (void) const

- $\texttt{right}([x, y]) = y$

Returns the right bound of `self`. Note that this method may return a finite floating-point number (i.e. neither a NaN, nor an infinity) even when the interval itself is empty.

interval **abs** (const interval& *I*)

- $\text{abs}(I) = \{|x| \in \mathbb{R}^+ \mid x \in I\}$

Returns the absolute value of *I*.

Example

```
1 cout << abs(interval(-5,6))
2      << abs(interval(-4,-2));
3 🖶 [0, 6] [2, 4]
```

double **chi** (const interval &*I*)

This function, introduced by Ratscheck and Rokne [7], characterizes the degree of symmetry of intervals. Its definition is as follows:

$$\text{For} I = [a, b], \quad \text{chi}(I) = \begin{cases} -1 & \text{if } I = 0 \\ a/b & \text{if } |a| \leqslant |b| \\ b/a & \text{otherwise} \end{cases}$$

Example

```
1 cout << chi(interval(3,6))
2      << chi(interval(-6,3))
3      << chi(interval::emptyset)
4      << chi(interval::universe)
5      << chi(interval("[-5,inf]"));
6 🖶 0.5 -0.5 NaN 1 0
```

interval **min** (const interval &I, const interval &J)

[7] Helmut Ratschek and Jon Rokne. Interval methods. In *Handbook of Global Optimization*, pages 751–828. Kluwer Academic, 1995.

- $\min([a,\ b],[c,\ d]) = [\min(a,c),\ \min(b,d)]$

Returns the minimum of two intervals.

Example

```
cout << min(interval(5,6),interval(3,9))
     << min(interval::emptyset,interval(3,8));
```

[3, 6] *[empty]*

interval **max** (const interval &*I*, const interval &*J*)

- $\max([a,\ b],[c,\ d]) = [\max(a,c),\ \max(b,d)]$

Returns the maximum of two intervals.

Example

```
cout << max(interval(5,6),interval(3,9))
     << max(interval::emptyset,interval(3,8));
```

[5, 9] *[empty]*

interval **floor** (const interval &*I*)

- [floor(I.left()), floor(I.right())]

Example

```
cout << floor(interval(4.5,6.5))
     << floor(interval("[-10.4,3.5]"));
```

[4, 6] [−11, 3]

interval **ceil** (const interval &*I*)

- [ceil(I.left()), ceil(I.right())]

Example

```
cout << ceil(interval(4.5,6.5))
     << ceil(interval("[-10.4,3.5]"));
```

[5, 7] [−10, 4]

interval **integer** (const interval &*I*)

- [ceil(I.left()), floor(I.right())]

Narrows down the bounds to the closest integers. Note that the resulting bounds are still `double` numbers, and may therefore not be representable with integral types.

Example

```
cout << integer(interval(4.5,6.5));
```

[5, 6]

9.1 Splitting methods

void interval::**split** (interval& *I1*, interval& *I2*) const

Splits `self` into two parts using `midpoint()`; returns the left part in *I1* and the right part in *I2*.
I1 or *I2* may be equal to `self`.

Example

```
interval I1a, I2a,
         I3(1.0,next_float(1.0)),
         I1b, I2b;
interval(4,5).split(I1a,I2a);
I3.split(I1b,I2b);
cout << I1a << " " << I2a << " "
     << (I1b==1.0) << " " << (I2b==I3) << endl;
[4, 4.5] [4.5, 5] true true
```

interval interval::**split_left** (void) const

- `split_left`$([a, b]) = [a, \uparrow (a + b)/2 \uparrow]$

Splits `self` into two parts using `midpoint()` and returns the left part.

Example

```
cout << interval(4,5).split_left();
[4, 4.5]
```

interval interval::**split_right** (void) const

- `split_right`$([a, b]) = [\uparrow (a + b)/2 \uparrow, b]$

Splits `self` into two parts using `midpoint()` and returns the right part.

Note. The left bound of the result is rounded up such that there is the least overlap possible with the interval returned by `split_left()`.

Example

```
cout << interval(4,5).split_left();
[4.5, 5]
```

9.2 Union and intersection

interval **operator&** (const interval& I1, const interval& I2)

- $I1 \cap I2$

Returns the interval resulting from the intersection of *I1* and *I2*.

Example

```
cout << interval(4,6) & interval(5,9);
[5, 6]
```

interval **operator|** (const interval& I1, const interval& I2)

- $I1 \cup I2$

Returns the interval resulting from the union of *I1* and *I2*.

Example

```
cout << interval(3,6) | interval(9,12);
[3, 12]
```

10 Input/output

10.1 Reading intervals

istream& **operator**>> (ostream& *in*, interval& I)

> Reads an interval from the input stream *in* and assigns it to *I*. If the string read is syntactically ill-formed, an `input_format_error` exception is thrown (see Section 15.1, page 58) if the library was compiled with exceptions enabled (see Section 2.2.2, page 5); alternatively, it prints an error message to `cerr` and aborts if exceptions were disabled.

10.1.1 Input format

A string to be translated into an interval must have the following syntax (with terminals in lower case and non-terminals in slanted upper case):

```
ITV_EXPR
        : PARSED_INTERVAL
        | ITV_EXPR + ITV_EXPR
        | ITV_EXPR - ITV_EXPR
        | ITV_EXPR * ITV_EXPR
        | ITV_EXPR / ITV_EXPR
        | - ITV_EXPR
        | + ITV_EXPR
        | ITV_FUNCTION_CALL
        | ( ITV_EXPR )
        ;

ITV_FUNCTION_CALL
        : cos ( ITV_EXPR )
        | sin ( ITV_EXPRT )
        | tan ( ITV_EXPR )
        | atan2 ( ITV_EXPR , ITV_EXPR )
        | acos ( ITV_EXPR )
        | asin ( ITV_EXPR )
        | atan ( ITV_EXPR )
        | cosh ( ITV_EXPR )
```

```
        | sinh ( ITV_EXPR )
        | tanh ( ITV_EXPR )
        | acosh ( ITV_EXPR )
        | asinh ( ITV_EXPR )
        | atanh ( ITV_EXPR )
        | exp ( ITV_EXPR )
        | log ( ITV_EXPR )
        | pow ( ITV_EXPR , ITV_EXPR )
        | sqrt ( ITV_EXPR )
        | nth_root ( ITV_EXPR , ITV_EXPR )
        ;

PARSED_INTERVAL
        : EXPRESSION
        | empty                                  // Empty interval
        | [ EXPRESSION ]
        | [ EXPRESSION , EXPRESSION ]
        | < EXPRESSION , EXPRESSION >
        | [ empty ]                              // Empty interval
        ;

EXPRESSION
        : NUMBER
        | dmin          // Smallest positive floating-point number
        | dmax           // Largest positive floating-point number
        | pi
        | inf              // Floating-point positive ``infinity''
        | EXPRESSION + EXPRESSION
        | EXPRESSION - EXPRESSION
        | EXPRESSION * EXPRESSION
        | EXPRESSION / EXPRESSION
        | - EXPRESSION
        | + EXPRESSION
        | FUNCTION_CALL
        | ( EXPRESSION )
        ;

FUNCTION_CALL
        : cos ( EXPRESSION )
        | sin ( EXPRESSION )
        | tan ( EXPRESSION )
        | atan2 ( EXPRESSION , EXPRESSION )
        | acos ( EXPRESSION )
        | asin ( EXPRESSION )
        | atan ( EXPRESSION )
        | cosh ( EXPRESSION )
        | sinh ( EXPRESSION )
        | tanh ( EXPRESSION )
        | acosh ( EXPRESSION )
        | asinh ( EXPRESSION )
        | atanh ( EXPRESSION )
        | exp ( EXPRESSION )
        | log ( EXPRESSION )
        | pow ( EXPRESSION , EXPRESSION )
```

```
| sqrt ( EXPRESSION )
| nth_root ( EXPRESSION , EXPRESSION )
;
```

Spaces are not significant except in numbers. The "+" sign before numbers and `inf` is optional. Note that the second argument of `nth_root` shall be a point interval that can be evaluated as an integer. Otherwise, an `invalid_action_error` exception is thrown (or an error is reported, depending on whether exceptions were enabled or not at configuration time—see Section 2.2.2, page 5).

If a rational number is not representable in the floating-point format, it is replaced by the smallest floating-point interval containing it. The notations "n" and "$[n]$" are equivalent.

The two bounds in the string "`<a, b>`" must be expressions that evaluate to the same value even for different rounding directions. An `input_format_error` exception is raised otherwise.

Example

```
interval x("[4, 6*7]");
interval y("[-inf, dmax]");
interval z("[3.14,3.15]/8", "[3.14,3.15]/7");
interval t("[3.14,3.15]/[7,8]");
```

Caution 1: Case is significant for all the operators.

Caution 2: Numbers appearing in the string shall not have more than 15 digits, otherwise their translation to floating-point numbers is not guaranteed to be correct.

Expressions in bounds are evaluated using interval arithmetic; the left (resp. right) bound is then used, depending on the side it appeared in.

Note that, as of version 4.2.0, the `atan2` operator is not yet implemented for interval expressions.

Contrary to constructors that take numeric constants as parameters, expressions such as `"inf"` or `"<-inf, -inf>"` lead to legitimate intervals (see the note Page 16).

10.2 Writing intervals

Intervals may be printed into a stream like any other C++ primitive type by using the "<<" operator.

ostream& **operator**<< (ostream& *out*, const interval& I)

> Prints the interval I to the output stream *out*. The way the intervals are actually displayed depends on the active format (see next section). However, whatever the format, an empty interval is always displayed as `[empty]`

10.2.1 Converting intervals to strings

For convenience, the `interval` class provides a conversion operator into the standard C++ type `string`.

Example

```
interval I(3,4);
string s = "test line embedding " + string(I) + " as a string";
// Now, s is "test line embedding [3, 4] as a string"
```

10.2.2 Output format

Intervals may be displayed following four different formats:

`agreeing`. By printing all the digits that are the same in the left and right bounds followed by an interval containing the remaining digits:

"`3.141~[5926, 6001]`" stands for "[3.1415926, 3.1416001]"

Note that if the bounds do not have any agreeing digit, there will still be a tilde before the bracketed part:

Example

```
interval::format(interval_format::agreeing);
std::cout << interval(4,6) << std::endl;
⏏ ~[4., 6.]
```

`bounds`. By printing the bounds between square brackets. Degenerate intervals whose left and right bound are equal are printed with angles:

Example

```
interval::format(interval_format::bounds);
std::cout << interval(3,5) << std::endl;
⏏ [3, 5]
std::cout << interval(4) << std::endl;
⏏ <4, 4>
```

`width`. by printing their midpoint and their width:

Example

```
interval::format(interval_format::width);
std::cout << interval(4,5) << std::endl;
⏏ 4.5 (+/- 0.5)
```

`hexa`. by printing the hexadecimal representation of their left and right bounds (useful when one wants to know the precise value of the bound without being affected by the round-off error due to binary-to-decimal conversion):

Example

```
interval::format(interval_format::hexa);
std::cout << interval("0.1") << std::endl;
⏏ [3fb9999999999999, 3fb999999999999a]
```

Note that *the `bounds` format is the only one recognized at present as an input* (see previous section).

The choice of the format to use is made through the following static methods:

void interval::**format** (interval_format::format_t *f*) static
interval_format::format_t interval::**format** (void) static

The first form of the method allows modifying the format to use in subsequent printing of intervals. The second form reports what is the current form in use. It returns a value of type `interval_format::format_t` (see below and 10.2.4 for an example of use).

interval_format struct

Structure type used to choose the ouput format for intervals. It has four possible values of type `interval_format::format_t`:

- `interval_format::agreeing`.
- `interval_format::bounds`.
- `interval_format::width`.
- `interval_format::center`.
- `interval_format::hexa`.

Example

```
interval I(interval::pi);

interval::format(interval_format::agreeing);
cout << I << "\n";
// Prints 3.14159265358979~[3, 4]
// The ~[] part is dropped if the bounds agree on all digits

interval::format(interval_format::bounds);
cout << I << "\n";
// Prints [3.141592653589793, 3.141592653589794]

interval::format(interval_format::width);
cout << I << "\n";
// Prints 3.141592653589793 (+/- 2.220446049250313e-16)

interval::format(interval_format::center);
cout << I << "\n";
// Prints 3.141592653589793

interval::format(interval_format::hexa);
cout << I << "\n";
// Prints [400921fb54442d18, 400921fb54442d19]
```

10.2.3 Choosing the number of digits to display

You can manipulate the number of digits to print by using the `precision()` static methods of the interval class:

std::streamsize interval::**precision** (void)

Returns the current number of digits used for printing bounds of intervals. See example below.

std::streamsize interval::**precision** (std::streamsize *n*)

Set the number of digits to use for printing bounds to *n*. In addition, returns the number of digits previously used.
See example below.

10.2.4 Example

Example

```
#include <iostream>
#include <gaol/gaol.h>

using std::cout;
using std::endl;

int main(void)
{
  gaol::init();
  interval::precision(4);
  interval::format(interval_format::bounds);
  cout << interval::pi << endl;

  if (interval::format() != interval_format::bounds) {
    cout << interval::pi << endl;
  } else {
    int old_prec = interval::precision(16);
    interval::format(interval_format::width);
    cout << interval::pi << endl;
  }
  gaol::cleanup();
}
```

On a Pentium-based PC, the previous program has the following output:

Output

```
[ 3.142, 3.142 ]
3.141592653589793 (+/- 2.220446049250313e-16)
```

The first call to `interval::format()` is unnecessary since the default format is `interval_format::bounds`.

Note. Translating an interval into a string and then reading it back as an interval is likely to produce an inaccurate or plain wrong result if you choose a precision different from `17`. It is however useless to specify a precision greater than `17` for the `double` format since the extra digits would be garbage.

The `interval::pi` constant is a predefined *canonical* interval containing π (see Section 6, page 19). Here, the width of the interval is equal to the ϵ of the format.

The `interval_format::width` format may be useful whenever the number of digits displayed is insufficient to know whether the result is a single floating-point number or an interval whose size is very small (consider for example the first result above), because *we have the guarantee that if the actual width of an interval is greater than zero, the width displayed will also be different from zero.* Another indication is that a degenerate interval is displayed as a floating-point number.

11 Floating-point numbers

11.1 Floating-point constants

In addition to the constants available through `numeric_limits<double>`, gaol defines the following `double` constants:

Constant (`double`)	Value
`two_pi`	$\updownarrow 2\pi \updownarrow$
`pi`	$\updownarrow \pi \updownarrow$
`half_pi`	$\updownarrow \frac{\pi}{2} \updownarrow$
`pi_dn`	$\downarrow \pi \downarrow$
`pi_up`	$\uparrow \pi \uparrow$
`half_pi_dn`	$\downarrow \frac{\pi}{2} \downarrow$
`half_pi_up`	$\uparrow \frac{\pi}{2} \uparrow$
`ln2_dn`	$\downarrow \ln 2 \downarrow$
`ln2_up`	$\uparrow \ln 2 \uparrow$
`two_power_53`	2^{53}
`GAOL_NAN`	NaN (quiet)
`GAOL_INFINITY`	$+\infty$

11.2 Floating-point functions

bool **feven** (const double& *x*)

Returns true whenever x is even.
This function should not be used with infinity and NaN arguments.

Example

```
1    assert( feven(3.0) );                          // false
2    assert( feven(3.5) );                          // false
3    assert( feven(4.0) );                           // true
4    assert( feven(4.5) );                          // false
5    assert( feven(GAOL_INFINITY) );              // always true
6    assert( feven(GAOL_NAN) );                   // always false
```

double **next_float** (double *x*)

Returns the smallest `double` greater than x.

double **previous_float** (double *x*)

Returns the greatest `double` smaller than x.

bool **is_signed** (double *x*)

Returns true whenever x is signed. No provision is made concerning the fact that x is a NaN. If you only want to test for negative numbers (and -0), you will have to test also whether x is a NaN by using the `isnan()` predicate in `math.h`.

double **minimum** (double *x*, double *y*)

Returns the minimum `double` value of x and y. This function is commutative and returns -0 when comparing -0 and $+0$, i.e.:

$$\begin{cases} \min(x,y) & = \min(y,x), & \forall x \neq \mathrm{NaN}, \forall y \neq \mathrm{NaN} \\ \min(x,\mathrm{NaN}) & = \min(\mathrm{NaN},x) = \mathrm{NaN}, & \forall x \\ \min(-0,0) & = \min(0,-0) = -0 \end{cases}$$

double **maximum** (double *x*,double *y*)

Returns the maximum `double` value of x and y. This function is commutative and returns $+0$ when comparing -0 and $+0$, i.e.:

$$\begin{cases} \max(x,y) & = \max(y,x), & \forall x \neq \mathrm{NaN}, \forall y \neq \mathrm{NaN} \\ \max(x,\mathrm{NaN}) & = \max(\mathrm{NaN},x) = \mathrm{NaN}, & \forall x \\ \max(-0,0) & = \max(0,-0) = 0 \end{cases}$$

ULONGLONGINT macro

Macro standing for an unsigned integral data type with a size equal to 8 bytes (usually `unsigned long long int`).

ULONGLONGINT **nb_fp_numbers** (double *a*, double *b*)

Returns the number of floating-point numbers in the interval $[a, b]$. In particular, we have:

- `nb_fp_numbers(a,next_float(a)) == 2`
- `nb_fp_numbers(a,a) == 1`

Note. As a precondition, a shall be lower or equal to b.
Returns `numeric_limits<ULONGLONGINT>::max()` if either a or b is a NaN or an infinity. In addition, raises an `invalid_action_error` exception (see Section 15.1, page 58) or calls `gaol_error` depending on the way the library was configured.

12 Manipulating the FPU

The gaol library provides functions to manipulate the FPU and its flags. The main functions are the one described in the next section for modifying the rounding direction. As for now, gaol provides these facilities for the following platforms:

- ix86 and compatibles under Linux
- SPARC under Solaris
- ISO C99-compliant platforms

Whenever possible, inline assembler versions are used.

12.1 Rounding functions

void **round_downward** (void)

Sets the rounding direction mode towards $-\infty$.

void **round_nearest** (void)

Sets the rounding direction mode to the nearest/even.

void **round_zero** (void)

Sets the rounding direction mode to zero.

void **round_upward** (void)

Sets the rounding direction mode to $+\infty$.

12.2 Manipulating the FPU flags

The following functions allow to manipulate the FPU flags. See the documentation of the FPU for your machine for a description of these flags.

void **clear_inexact** (void)

Clears the `inexact` flag of the FPU.

Warning: This function is currently unavailable on some platforms. For these platforms, a warning is issued when the function is called.

int **get_inexact** (void)

Returns a non-zero value whenever the last floating-point operation was performed with rounding. The associated FPU flag is a persistent one. As a consequence, you should always clear it by calling `clear_inexact()` *before* performing the operation you want to test.

unsigned short **get_fpu_cw** (void)

Returns the value of the FPU control word.

Warning: This function is currently unavailable on some platforms. For these platforms, a warning is issued when the function is called.

unsigned short **get_fpu_sw** (void)

Returns the value of the FPU status word.

Warning: This function is currently unavailable on some platforms. For these platforms, a warning is issued when the function is called.

13 Version information

The library provides four constants to allow programs to determine at runtime with which version they are dynamically linked with. The versioning scheme adopted is the one used by the Apache Software Foundation described at http://apr.apache.org/versioning.html.

unsigned int **version_major** const

Major version of the library.

unsigned int **version_minor** const

Minor version of the library.

unsigned int **version_patch** const

Patch version of the library.

const char *const **version** const

Version of the library as a string.

Example

```
1 const char *const version = "1.0.3";
```

14 Additional functions

The following functions are utility functions not necessarily related to intervals or floating-point numbers.

template <typename T>
bool **odd** (const *T*& *x*)

> Returns `true` if x is odd and false otherwise. The T type may be any type providing the `&` ("bitwise `and`") operator with the same semantics as the one for `int`s.

template <typename T>
bool **even** (const *T*& *x*)

> Returns `true` if x is even and false otherwise. The T type may be any type providing the `&` ("bitwise `and`") operator with the same semantics as the one for `int`s.

15 Error handling

A program that uses gaol may report errors in two different ways:

- by throwing an exception;
- or by setting the `errno` variable.

The mecanism in use depends on the way the library is configured. If you use the option `--enable-exceptions=yes`, all errors are reported through exception throwing; otherwise, the `errno` variable is used. Relying on exceptions is more in the C++ spirit, though it may incur some overhead.

It is up to the user to comply with this mecanism when adding error reporting code to ones program. Gaol defines the following macro to be used whenever one wants to report an error.

gaol_ERROR (*excep*,*msg*) macro

> The behavior of the macro depends on the value chosen for the option `--enable-exceptions`: if exceptions are enabled, exception *excep* is raised with the message *msg*; otherwise, the program aborts with message *msg*.

Example

```
interval x;

[Code manipulating x]

if (x.is_empty()) {
   gaol_ERROR(failure_error,"Emptyness of one interval");
}
```

The `gaol_error()` function is defined as follows:

void **gaol_error** (const char *const *err*)
void **gaol_error** (const char **file*, int *line*, const char **err*)

> Displays a message on the standard error output. The ternary version should be called with the `GAOL_FILE_POS` macro for the first two parameters.

The `GAOL_FILE_POS` macro is described in the next section.

15.1 Exceptions

The library defines `gaol_exception` as a class to be used as a base class for all gaol exceptions. All of them provide at least the name of the file and the line number from where the exception has been thrown. As a facility, gaol defines the following macro:

GAOL_FILE_POS macro

Expands itself into the first two arguments of any constructor for `gaol_exception` or one of its derived classes:

Example

```
1 if ([some condition]) {
2   throw gaol_exception(GAOL_FILE_POS,
3                        "No additional information");
4 }
```

All gaol exceptions can be sent to an output stream through the "<<" operator.

15.1.1 The `gaol_exception` exception

The `gaol_exception` class is the base class from which derive all gaol exceptions. It inherits from the C++ standard class `exception`.

Every exception class deriving from it must at least provide the name of the file and the line where the corresponding exception was thrown. As a consequence, the constructors for `gaol_exception` are as follows:

gaol_exception::**gaol_exception** (const char* *f*, unsigned *l*)
gaol_exception::**gaol_exception** (const char* *f*, unsigned *l*, const char* *e*)

Constructs a `gaol_exception` being thrown from file *f* at line *l*. The second form permits adding some explanatory string *e*.

The class offers the following accessors:

const char* gaol_exception::**file** (void) const

Return the name of the file from where the exception was thrown.

unsigned int gaol_exception::**line** (void) const

Returns the line number in the file from where the exception was thrown.

const char* const gaol_exception::**explanation** (void) const

Returns a string explaining why the exception was thrown. Returns an empty string if no additional information was provided.

15.1.2 The `input_format_error` exception

The `input_format_error` exception is thrown whenever one attempts to create an interval from an invalid string. This situation may occur when reading an interval from a stream with the >> operator, or when creating an interval from a string.

This class, as all gaol exceptions, derives from `gaol_exception` (see Section 15.1.1, page 58). Its constructors have the same format than the ones for `gaol_exception`, namely:

input_format_error::**input_format_error** (const char* *f*, unsigned *l*)
input_format_error::**input_format_error** (const char* *f*, unsigned *l*, const char* *e*)

> Constructs an `input_format_error` being thrown from file *f* at line *l*. The second form permits adding some explanatory string *e*.

The methods of the class are inherited from `gaol_exception` (see Section 15.1.1, page 58).

15.1.3 The `unavailable_feature_error` exception

This exception is thrown whenever an unavailable feature is requested.

This class, as all gaol exceptions, derives from `gaol_exception`. (see Section 15.1.1, page 58). Its constructors have the same format than the ones for `gaol_exception`, namely:

unavailable_feature_error::**unavailable_feature_error** (const char* *f*, unsigned *l*)
unavailable_feature_error::**unavailable_feature_error** (const char* *f*, unsigned *l*, const char* *e*)

> Constructs an `unavailable_feature_error` being thrown from file *f* at line *l*. The second form permits adding some explanatory string *e*.

The methods of the class are inherited from `gaol_exception` (see Section 15.1.1, page 58).

15.1.4 The `invalid_action_error` exception

This exception is thrown whenever a function is called with invalid arguments (e.g. calling `nb_fp_numbers()` with NaNs as parameters).

This class, as all gaol exceptions, derives from `gaol_exception` (see Section 15.1.1, page 58). Its constructors have the same format than the ones for `gaol_exception`, namely:

invalid_action_error::**invalid_action_error** (const char* *f*, unsigned *l*)
invalid_action_error::**invalid_action_error** (const char* *f*, unsigned *l*, const char* *e*)

> Constructs an `invalid_action_error` being thrown from file *f* at line *l*. The second form permits adding some explanatory string *e*.

The methods of the class are inherited from `gaol_exception` (see Section 15.1.1, page 58).

15.2 Warnings

void **gaol_warning** (const char **warn*)
void **gaol_warning** (const char **file*, int *line*, const char **warn*)

Prints the message *warn* on the standard error output. The second form should be called with the `GAOL_FILE_POS` macro for the first two parameters.

16 Debugging facilities

The debugging facilities described hereunder are available only if gaol has been configured with the debugging facilities enabled (see the `--enable-debug` option, Section 2.2.2, p. 5).

int **debug_level**

Global variable used to remember the current value of the debugging level. This variable is set when initializing the library. The variable is declared in the `gaol` namespace.

GAOL_DEBUG (*lvl*,*cmd*) macro

Executes *cmd* if *lvl* is lower or equal to the current debugging level (see the variable `debug_level` above).
This macro defaults to nothing if the library was not configured with the `--enable-debug` option.

A possible use for this macro is as follows:

Example

```
1 interval x(-10,10);
2
3 [Some code]
4
5 GAOL_DEBUG(1,cout << "The value of x is " << x);
6 x += double_interval(3.5,4.5);
7 GAOL_DEBUG(2,cout << "Now the value of x is " << x);
```

The first message will be displayed whenever gaol has been configured with debugging facilities enabled (see the `--enable-debug` option). The second message will be displayed only if the debugging level is greater or equal to 2.

GAOL_ASSERT macro

(*cond*) Tests whether *cond* holds. Aborts with an error message if it is not the case.
This macro defaults to nothing if the library was not configured with the `--enable-debug` option.

A possible use for this macro is as follows:

Example

```
1 int x;
2 cout << "Give an integer no greater than 5: ";
3 cin >> x;
4 GAOL_ASSERT(x <= 5);
```

17 Profiling

The following functions permit computing the time used for a computation. The returned times are *user* times, meaning that delays induced by input/output operations and freezing during CPU switches in multi-programming environments are not taken into account.

If you need to keep track of several events, consider using an object of the `timepiece` class (see Section 17.1, page 64) instead of calling directly the functions below.

Warning. The precision of the timing functions depends on the platform used. For example, the precision on ix86-based machines is usually no better than 10 ms. What is more, despite the fact that the reported times are user times, they may vary from an execution to another, and can get larger on heavily loaded machines.

long **get_time** (void)

Returns the time in milliseconds since a certain unspecified moment. This function should only be used to compute differences between two calls since the starting point may vary depending on the availability of `clock()` or `getrusage()` on the system.

Warning. if the function in use is the standard `clock()`, the time returned will wrap approximately every 72 minutes. Consequently, it is not safe to use `get_time()` in that case for processes requiring more than 72 minutes to execute.

void **reset_time** (void)

Resets the time counter. To be called just before executing some code to be profiled.

long **elapsed_time** (void)

Returns the time in milliseconds elapsed between now and the last call to `reset_time()`.

long **intermediate_elapsed_time** (void)

Returns the time in milliseconds elapsed between now and the last call to `reset_time()` or to `intermediate_elapsed_time()`.

Here is a typical example of use of the timing functions:

Example

```
int main(void)
{
  reset_time();
  for (unsigned int i=0;i<1000;++i) {
     [Some time consuming operations]
  }
  cout << "Elapsed time: " << elapsed_time() << " ms." << endl;
  return 0;
}
```

17.1 The `timepiece` class

A `timepiece` object allows to keep track of the time spent to perform a particular task. Since the counter used is local to the object, it is possible to monitor more than one such process.

17.1.1 Methods of the `timepiece` class

void timepiece::**start** (void)

Starts the timepiece.

void timepiece::**stop** (void)

Stops the timepiece and accumulates the time spent since the last call to `start()`.

void timepiece::**reset** (void)

Resets to zero the counter keeping track of the total time the timepiece was running.

long timepiece::**get_total_time** (void) const

Returns the total amount of time the timepiece was running (time between calls to `start()` and `stops`. The timepiece shall have been stopped by calling the `stop()` method before calling this one.

long timepiece::**get_intermediate_time** (void) const

Returns the amount of time spent since the last call to `start()`.

Example

```
int main(void)
{
  timepiece t;
  t.start();
  for (unsigned int i=0;i<1000;++i) {
     [Some time consuming operations]
     cout << "Intermediate time: "
          << t.get_intermediate_time() << " ms." << endl;
  }
  t.stop();
  cout << "Elapsed time: " << t.get_total_time() << " ms." << endl;
  return 0;
}
```

18 Additional Documentation

18.1 Documentation on gaol

The primary reference is this manual. There is also an html reference for the code itself, which might be of interest only to developers seeking to understand/modify gaol.

18.2 References

The following articles and books have inspired in some way or another the devising of the gaol library and/or the writing of this manual.

- *Interval Arithmetic Specification.* Dmitri Chiriaev and G. William Walster. Draft revised May 1998.
- *The Extended Real Interval System.* G. William Walster. April 1998.
- *C++ Interval Arithmetic Programming Reference.* Sun Microsystems, Inc. October 2000, revision A.
- *Interval Arithmetic: From Principles to Implementation.* T. Hickey, Q. Ju, and M. H. van Emden. Tech. Rep. CS-99-202, CS Dept. Brandeis U, July 1999.

19 Reporting bugs

All bugs and suggestions for improvement shall be submitted through the appropriate form available on the web site:

http://sourceforge.net/projects/gaol/

20 Contributors

The main implementor and lead designer for the gaol library is Frédéric Goualard (goualard@users.sourceforge.net).

The `interval pow(const interval&, const interval&)` function was designed by Marc Christie (christie@users.sourceforge.net).

The code for the multiplication and the division is largely inspired from the one presented by Tim Hickey, Qun Ju and Maarten Van Emden in *Interval Arithmetic: from Principles to Implementation* Journal of the ACM 48(5):1038–1068, september 2001.

Library Copying

GNU LIBRARY GENERAL PUBLIC LICENSE

Version 2, June 1991

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that

what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

Terms and Conditions for Copying, Distribution and Modification

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

   The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either

verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. The modified work must itself be a software library.

   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

      (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you

distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the

Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

   b. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

   c. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

   d. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

   For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

    a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

    b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE

OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> *one line to give the library's name and a brief idea of what it does.*
> Copyright (C) *year name of author*
>
> This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.
>
> You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.
>
> *signature of Ty Coon*, 1 April 1990
> Ty Coon, President of Vice

That's all there is to it!

Index