'Many Digits' Friendly Competition

Milad Niqui, Freek Wiedijk
Institute for Computing and Information Sciences, Radboud University Nijmegen, Toernooiveld 1, 6525 ED, Nijmegen, The Netherlands.
{M.Niqui, freek}@cs.ru.nl

Abstract

In this article we give an account of the 'Many Digits' friendly competition, a competition between arbitrary precision arithmetic packages that was organised following [1] and was held on October 3–4, 2005 in order to investigate the state of the art in the field of exact and arbitrary precision arithmetic.

The competition consisted of two types of problems, a basic and an intermediate set. The basic problems consisted of simple expressions involving elementary functions. The intermediate problems involved more mathematical construct or slightly more programming. A solution consisted of a sequence of decimal digits, for most problems 10000 digits. The systems were required to calculate the solution within two minutes and depending on their success retry the problem with a higher/lower number of digits. The competition had nine participants, some of them competing remotely.

Contents

1 Background

There are a several packages/libraries that are capable of performing arbitrary precision arithmetic on elementary real functions. They implement different ideas and approaches to *arbitrary precision arithmetic*, *multiple precision arithmetic* or *exact real arithmetic*. A direct comparison between this group of software — to the extent that such comparison is possible— will provide a good understanding of the state of the art in the field of arbitrary precision computing. The idea of 'Many digits' friendly competition was to provide a benchmark platform, albeit limited, for comparing some aspects of such kind of systems. Similar benchmarks already existed, including some of those targeting the same kind of systems as the 'Many digits' competition, eg., *competition at CCA'00 [1]*, *software demonstration at the CCA'02[15]*, the benchmarks published at [11, 18] or the very successful challenge at *SIAM 100-digit challenge* [23, 22, 2, 21]. Inspired by these events, the authors organised the 'Many digits' friendly competition on October 3–4, 2005 in the Institute for Computing and Information Sciences at Radboud University Nijmegen. In this report we present the format, the problems and the results of this competition.

2 Format and Rules of the Competition

2.1 Two Problem Sets

There were two problem sets, the *practice problems* and the *competition problems*. Both sets were very similar.

The first set, the set of practice problems, was intended for the participants to know what to expect of the competition. Also they were to be used by the organisers of the event to make sure that the competition procedure would work well. The results of the

timings for the practice problems would not count in the final rankings of the systems. The second set would be the one used for the competition and was only announced shortly before the competition.

Each set consists of 24 problems. The first twelve of those problems have the form 'evaluate a closed term to K digits'. The second twelve problems are more involved: iterations, solving equations, calculation of integrals, inverting matrices, etc. However, all problems have the form that they have a parameter N which determines how difficult the problem is, and ask for a finite string of decimal digits (e.g. 10^N decimal digits).

The practice problems were numbered **P01**–**P24** (Section 3). Competition problems were numbered **C01**–**C24** (Section 4). The competition problems were striven to be similar to the practice problem, i.e., in designing the problems the intention was that the problem **Cx** be similar to **Px** (the practice problem with the same number). Of course this similarity should be understood in a broad sense: it was tried that the principle behind each practice problem is mimicked in the corresponding competition problem. The reason for this correspondence between the two set of problems was two-folds: by releasing the practice problems before the competition and getting feedback from those systems that tried the practice problems we would get an idea of possible issues in the design of the problems or the competition in general, eg., regarding the rules and syntax on input/output. But more importantly, the participants, by trying their systems on practice problems, would get a realistic view of whether their system is capable of solving the 'corresponding' competition problem in the tight time frame of the competition. If necessary they could improve some aspects of their system before the competition that would help them tackle the competition problems better.

2.2 Procedure During The Competition

There was one single computer on which the competition was run. Each system participating in the competition would get a turn running all 24 problems. Between the different systems the computer would be rebooted, to ensure there will not be lingering effects of the runs of previous systems.

The problems were to be run one by one, so not in parallel. For each problem there was be a default value for the 'parameter', that had been chosen based on experience with the practice problems. The problem was to first run using this parameter. If the solution took longer than two minutes, the calculation was interrupted ('`control C`'), and the problem was be rerun with the parameter decreased by one, until the running time would drop below two minutes. If the solution took shorter than two seconds, the problem would be rerun with the parameter increased by one until the running time would raise above two seconds. (This means that the problems might not be identical between systems. However, it will still be clear which system is better at a problem than another one.)

The timings was to be the 'wall-clock time'. In this regard, prior to the competition we fixed the following in the rules.

> *'They [the timings for each system] will be either measured automatically, or if that is too difficult for some systems, for those systems just by using a stopwatch.'*

Afterwards, all the solutions turned to be executable using a simple command (see Appendix B). Hence we used the real-time output of `/usr/bin/time` to measure the wall-clock time.

2.3 Caching/Memoisation

The systems were allowed to cache the intermediate results. It is possible that the times associated with the final parameter N is affected by caching. Any such effect would be corrected by rerunning the solutions after the competition.

All the answers had to be computed while the systems were connected to our machine. So solutions could not carry precomputed answers in them. This also included non-integer sub terms of problems: for example in problem **P01** the solution was not allowed to contain a fully precomputed version of neither of $sin(1)$,$sin(sin(1))$ or $sin(sin(sin(1)))$ before connecting to our system.

2.4 Syntax of Input/Output

As can be seen from the list of practice problems, there are four different types of output.

First K digits after the decimal point of X

Most of the problems ask for the 'first K digits after the decimal point of' the numerical value of the problem. In this case we are only interested in the fractional part of the numerical value of the problem.

If a problem asks for K digits after the decimal point, then the program should give the digits of the exact answer according to *1ulp* rule. We do not require the sign of the result to be part of the output

For example if we ask for the three digits after the decimal point of $5/3$ or $-5/3$, in both cases the answer that we require is the string '666' or '667' and not '1.666'.

If the answer is exactly 1, and we ask for three digits after the decimal point, then '999', '000' and '001' will all be considered acceptable. (Which variant the system gave is indicated in the final table of how the systems behaved by postfixing the time with '−', no postfix, or '+'.)

If the answer is strictly between 1 and 1.001 then the answers '000' and '001' are both accepted. Again these are indicated in the table of results by no postfix or by the postfix '+' (see Table 3).

We asked the participants that if they can influence their system on this behaviour, we would have a preference for no postfix over '−', and for '−' over '+'. However, this preference would not affect the final ranking of the systems, which is just by speed.

In case a real number has two decimal representations the truncated version of both representations is accepted as correct answer.

If a program gets the last few digits wrong, it would be rerun asking for 10 extra digits, and if then after removal of that 10 extra digits it manages to get the last digit of the original number of digits correct, then that run would count.

For the above reasons, the input to the problems is not the parameter N (which indicates the level of difficulty) but directly the number of digits that we are looking

after. For example in the problem **P01** the initial input to the problem is 10000 (10^N) and not 4 (N). Depending on the outcome the second input might be

1000 : if the problem is not solved after two minutes;

100000 : if solving the problem takes less than two seconds;

10010 : if the problem is solved within two minutes but the 10000th digit is rounded incorrectly.

If the problem is solved within two minutes and all the digits are correct no more actions is needed to be taken and the system should proceed to solve the next problem.

10^N+10 **case** In the case a problem is solved within two minutes but only the last few digits are incorrect, we need the $10^N + 10$ digits of the *original* problem. For example in **P13** with $N = 4$ we need the first 10010 digits after the decimal point of x_{10000}.

In this case the system would be ranked under N, but the time for producing 10^N+10 digits is considered in the final table. The system would be indicated in the final table as '*'. Having '*'s were not to affect the final ranking.

Note that in the diagonal problems (**P13**–**P16**) the answer for $N = 3$ is still not a prefix of the answer for $N = 4$. We do not fix the iteration of the loops to be always 10000 rather it is always 10^N. In other words, it is not important that the solution are not converging by increasing N, but instead we want the problems to become more difficult whenever N increases. This is to minimise the effect of caching.

First K digits starting from the first nonzero digit after the decimal point of X

There is one problem of this type. The explanation is similar to the above case with the exception that the leading zeros after the decimal point should not be included in the output.

Kth partial quotient of the regular continued fraction of X

There is one problem of this type. The problem asks for the Kth partial coefficient of the regular continued fraction of a number X where $0 < X < 1$.

As an example let

$$a = 474200268124105199161441151369724483594818921505749248182939666 50$$

and

$$b = 679589410007729162090655784069715892527056724414013407130379512 51$$

and let $X = a/b$. Then for $0 < K < 50$ the Kth partial quotient of the regular continued fraction of X will be K.

As another example, for $X = \cos(\frac{2\pi}{7})$ (problem **P20**) the 350th , 701st and 5598th partial quotients are respectively '349', '1433' and '340613'.

First K decimal digits of element (m, n) of matrix M

Two of the problems are of this type. We are asking for the K most significant digits of an element of the square matrix X. The output should not contain the decimal point or the sign.

As an example let 2×2 matrix M be defined as

$$M = \begin{bmatrix} 1 & \frac{-1248480910}{75257} \\ \frac{4396347}{4395316} & 1 \end{bmatrix} .$$

Then the output for the first 10 decimal digits of the element $(1, 2)$ of M should be '1658956522'. Similarly the output for the first 10 decimal digits of the element $(2, 1)$ should be '1000234567'.

2.5 Scoring The Systems

For each problem the systems will be ranked from 1 to M, where M is the number of systems that can do the problem. (If a system cannot do a problem or it gives the wrong answer, it gets rank $M + 1$ for that problem.)

First we rank the systems that can solve the problem with highest N. These systems will be ranked among themselves according to the time.

Inevitably for some problems a system may take more than two minutes for $N + 1$ and less than two seconds for N to produce the solution. In such case the system will be ranked as $N/(N + 1)$.

In the final scoring the systems ranked $N/(N + 1)$ are considered to be between the systems ranked N and those ranked $N + 1$. The systems ranked $N/(N + 1)$ will not be ordered among themselves because there is no timing in the selected window to compare with. For the practice problems the timings for both N and $N + 1$ were announced but for the final table of the competition we only put 0 as the timing for N without the actual (negligible) timing having any effect on the ranking (Table 3).

The score of the system will be the sum of its ranks over all the problems. This will determine the overall ranking.

2.6 Specification of the Environment Used

The machine used was a mono-processor AMD Opteron 144 machine (1.8GHz, ∼1250 SPECint2000) running Debian GNU/Linux '*sid*' unstable i386 (32 bit mode), with 4GB of RAM. The system ran the Linux kernel version 2.6.12.

Both during the competition and in the course of post-competition tests, between any two consecutive participants the system would be rebooted in order to assign an equal amount of memory to all participants.

Each participating team were given a login on the machine four weeks before the competition. The participants were asked to install the necessary software for their system on the competition machine. Moreover, they were encouraged to try their solutions for the practice problems on the competition machine. This way we could get an idea of the feasibility of our procedure. Most of the systems tried to solve the practice problems and their timings were publicly announced.

The questions for the competition were announced three days before the competition. On the competition day each participant was given a one hour slot to login to the system and run all the solutions. After logging in, the participants were required to run `ttyrec` command to record the session on the competition in a log file that could be replayed later. These log files were used merely for preparing the provisional rankings and keeping track of the command line arguments for various problems. The final rankings were announced two weeks after the end of the competition, based on rerunning all the solutions by the organisers in the same environment as during the competition.

3 Practice Problems

We initially contacted some of the potential participants regarding the idea of having this competition, including the authors of some of the packages/libraries that are capable of performing arbitrary precision arithmetic on elementary real functions. In the email, after putting forward the initial idea and a rough sketch of the competition set-up (to the extent it was known at that point), it was written:

> *'We would very much appreciate suggestions for problems that people working in the field of exact arithmetic consider reasonable and interesting for this competition. Even if you decide not to participate in this event, then suggestions for problems are appreciated.'*

The purpose was that, in addition to the basic computation tasks (Section 3.1), to inspect whether there is enough interest in the potential participants for slightly more complicated tasks. In response we received some interesting suggestions for competitive problems, many of which were included in the practice problem set. Furthermore, it was decided the competition problems will be divided into two sets:

1. a set that all systems are likely able to tackle albeit not within two minutes. These were mainly simple expressions involving elementary functions and were called the *basic problem set*.

2. problems that either required a bit of coding (but not complicated coding, eg., computing sums and iterations) on part of the participants or dealt with matrices and advanced mathematical functions. These were called the *intermediate problem set*.

The intermediate problem set intended to assess the ability of the systems beyond the expressions in the basic problem sets. It was not expected of all the systems to be able to deal with all of a them as some targeted very specialised class of functions or features that only few systems had implemented.

After this, following the received suggestions, the list of practice problems were chosen and announced publicly. Before the competition and as the systems were trying the practice problems, many ambiguities in the original proposed version was discovered by the participants. This resulted in several improvements, mainly in the type of required output in some of the problems as well as clarification and modifications in the rules (Section 2). Below is the final list of practice problems after all the modifications during the pre-competition period.

3.1 Basic Set of Practice Problems

The required output in all cases is the string containing first 10^N decimal digits after the decimal point. The first 10 problems were given in the CCA 2000 competition [1].

Problem P01. $[N = 4]$ $\sin(\sin(\sin(1)))$

Problem P02. $[N = 4]$ $\sqrt{\pi}$

Problem P03. $[N = 4]$ $\sin(e)$

Problem P04. $[N = 4]$ $e^{\pi\sqrt{163}}$

Problem P05. $[N = 4]$ e^{e^e}

Problem P06. $[N = 4]$ $\log(1+(\log(1+\log(1+\log(1+\pi)))))$

Problem P07. $[N = 4]$ e^{1000}

Problem P08. $[N = 4]$ $\cos(10^{50})$

Problem P09. $[N = 4]$ $\sin(\frac{3\log(640320)}{\sqrt{163}})$

Problem P10. $[N = 4]$ $\sqrt[3]{\sqrt[5]{\frac{32}{5}} - \sqrt[5]{\frac{27}{5}}} - \frac{1+\sqrt[5]{3}-\sqrt[5]{9}}{\sqrt[5]{25}}$

Problem P11. $[N = 4]$ $\tan(e)+\arctan(e)+\tanh(e)+\operatorname{arctanh}(\frac{1}{e})$

Problem P12. $[N = 4]$ $\arcsin(\frac{1}{e}) + \cosh(e) + \operatorname{arcsinh}(e)$

3.2 Intermediate Set of Practice Problems

The initial value for parameter N is given between the square brackets. Note that the required output in Problems **P16**, **P20**, **P23** and **P24** is different from the other problems.

Together with each problem we mention the origin of the problem (either the person who suggested it to the organisers or the work in the literature that inspired the problem).

Problem P13. $[N = 4]$ The logistic map: $x_0 = 0.5, x_{n+1} = 3.75 \times x_n(1 - x_n)$

Output: The first 10^N digits after the decimal point of x_{10^N} .

Origin: This problem was suggested by Norbert Müller. It is also one of the problems in [1], albeit with a different starting value.

Problem P14. $[N = 2]$ $a_0 = \frac{11}{2}, a_1 = \frac{61}{11}, a_{n+1} = 111 - \frac{1130-\frac{3000}{a_{n-1}}}{a_n}$

Output: The first 10^N digits after the decimal point of $a_{10^{N+2}}$.

Origin: This problem is presented by Jean-Michel Muller in [14].

Problem P15. $[N = 4]$ $h(i) = \sum_{k=1}^{i} \frac{1}{k}$

Output: The first 10^N digits after the decimal point of $h(10^{N+1})$.

Origin: This problem was suggested by Norbert Müller. The corresponding competition problem **C15** is inspired by [5].

Problem P16. $[N = 4]$ $f(i) = \pi - \Big(3 + \frac{1\times1}{3\times4\times5} \times \big(8 + \frac{2\times3}{3\times7\times8} \times (\ldots (5i-2) + \frac{i(2i-1)}{3(3i+1)(3i+2)})\big)\Big)$

Output: The first 10^N digits starting from the first nonzero digit after the decimal point of $f(10^N)$.

Origin: The series are due to Bill Gosper [5], also studied by Jeremy Gibbons [4]. The corresponding competition problem **C16** can be found in [5] as well.

Problem P17. $[N = 4]$ (Riemann's ζ) $S = \zeta(2)\zeta(3) + \zeta(5)$

Output: The first 10^N decimal digits after the decimal point of S .

Origin: The corresponding competition problem (**C17**, Section 4.2) was inspired by [19].

Problem P18. $[N = 4]$ (Euler's γ) $\gamma = \lim_{n\to\infty} (\sum_{k=1}^{n} \frac{1}{k} - \log(n))$

Output: The first 10^N decimal digits after the decimal point of γ .

Origin: Some systems have predefined routines for such mathematical constants, while others rely on user's implementation. The purpose of this problem was to assess this aspect of the systems. Note that according to the rules in Section 2.3, the digits were not allowed to be precomputed. Incidentally, both here and the corresponding competition problem the irrationality of the constant in question was unknown at the time of the competition.

Problem P19. $[N = 4]$ $a_n = n^2, L = \sum_{n=1}^{\infty} (\frac{1}{7})^{a_n}$

Output: The first 10^N digits after the decimal point of L .

Origin: This problem was suggested by Norbert Müller.

Problem P20. $[N = 4]$ $X = \cos(\frac{2\pi}{7})$

Output: The 10^Nth partial quotient of the regular continued fraction of X .

Origin: This problem was suggested by Keith Briggs.

Problem P21. $[N = 4]$ Equation $e^{\sin(x)} = x$

Output: The first 10^N digits after the decimal point of x .

Origin: This problem was suggested by Paul Zimmermann.

Problem P22. $[N = 3]$ $J = \int_0^1 \sin(\sin(x))dx$

Output: The first 10^N digits after the decimal point of J .

Origin: This problem was suggested by Paul Zimmermann.

Problem P23. $[N = 2]$ M_1=Inverse of H_{10^N} (the $10^N \times 10^N$ Hilbert matrix)

Output: The first 10 decimal digits of the element $(10^N-1, 10^N-3)$ of M_1 .

Origin: This problem was suggested by Norbert Müller. Based on this, and inspired by [20] the corresponding competition problems (**C23**–**C24**, Section 4.2) were devised.

Problem P24. $[N = 2]$ M_2=Inverse of $I_{10^N} + H_{10^N}$ (sum of the identity and the $10^N \times 10^N$ Hilbert matrix)

Output: The first 10 decimal digits of the element $(10^N-1, 10^N)$ of M_2 .

Origin: This problem was suggested by Norbert Müller.

4 Competition Problems

As it was mentioned before, in designing the competition problems we decided to be as faithful as possible to the practice problems. This meant that problem **Cx** is intended to be similar to **Px**. This concern was especially the case for intermediate set: the basic set were, as a whole, indicative of systems' capability on dealing with elementary functions. While each problem in the intermediate set targeted a specific feature. However, even in the basic set the 'principles' behind the problems was tried to remain intact. In the intermediate set, sometimes only the parameters were altered (eg., **C13**) while in other cases the resemblance is more general than that. In particular, **C23**–**C24** were inspired by [20] as well as the practice problems **P23**–**P24**.

4.1 Basic Set of Problems

The required output in Problems **C01**–**C12** is the string containing the first 10^N decimal digits after the decimal point of the constant in question. The initial value for parameter N is given between the square brackets.

Problem C01. $[N = 4]$ $\sin(\tan(\cos(1))$

Problem C02. $[N = 5]$ $\sqrt{\frac{e}{\pi}}$

Problem C03. $[N = 4]$ $\sin\left((e+1)^3\right)$

Problem C04. $[N = 4]$ $e^{\pi\sqrt{2011}}$

Problem C05. $[N = 4]$ $e^{e^{\sqrt{e}}}$

Problem C06. $[N = 4]$ $\operatorname{arctanh}\Big(1 - \operatorname{arctanh}\big(1 - \operatorname{arctanh}(1 - \operatorname{arctanh}(\frac{1}{\pi}))\big)\Big)$

Problem C07. $[N = 4]$ π^{1000}

Problem C08. $[N = 4]$ $\sin(6^{6^6})$

Problem C09. $[N = 4]$ $\sin\Big(10 \times \arctan\big(\tanh(\frac{\pi\sqrt{2011}}{3})\big)\Big)$

Problem C10. $[N = 4]$ $\sqrt[3]{7 + \sqrt[5]{2} - 5\sqrt[5]{8}} + \sqrt[5]{4} - \sqrt[5]{2}$

Problem C11. $[N = 4]$ $\tan(\sqrt{2}) + \operatorname{arctanh}(\sin(1))$

Problem C12. $[N = 4]$ $\arcsin(\frac{1}{e^2}) + \operatorname{arcsinh}(e^2)$

4.2 Intermediate Set of Problems

The initial value for parameter N is given between the square brackets. Note that the required output in Problems **C16**, **C20**, **C23** and **C24** is different from the other problems.

Problem C13. $[N = 4]$ The logistic map: $x_0 = 0.5,\ \ x_{n+1} = 3.999 \times x_n(1 - x_n)$.

Output: The first 10^N digits after the decimal point of x_{10^N} .

Problem C14. $[N = 2]$ $a_0 = \frac{14}{3},\ \ a_1 = 5,\ \ a_2 = \frac{184}{35},\ \ a_{n+2} = 114 - \frac{1463 - \frac{6390 - \frac{9000}{a_{n-1}}}{a_n}}{a_{n+1}}$.

Output: The first 10^N digits after the decimal point of $a_{10^{N+2}}$.

Problem C15. $[N = 4]$ $h(n) = \sum_{k=n}^{n^2} \frac{1}{k}$.

Output: The first 10^N digits after the decimal point of $h(10^{N+1})$.

Problem C16. $[N = 4]$ $f(i) = \frac{\pi^2}{6} - \Big(\frac{13}{8} + \frac{1}{8\times 27} \times \big(\frac{34}{8} + \frac{8}{8\times 125} \times (\ldots (\frac{21i-8}{8} + \frac{i^3}{8(2i+1)^3}))\big)\Big)$.

Output: The first 10^N digits starting from the first nonzero digit after the decimal point of $f(10^N)$.

Problem C17. $[N = 4]$ (Riemann's ζ) $S = -4\zeta(2) - 2\zeta(3) + 4\zeta(2)\zeta(3) + 2\zeta(5)$.

Output: The first 10^N decimal digits after the decimal point of S .

Problem C18. $[N = 4]$ (Catalan's G) $G = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)^2}$.

Output: The first 10^N decimal digits after the decimal point of G .

Problem C19. $[N = 4]$ $a_n = n^3 + 1,\ \ L = \sum_{n=1}^{\infty} (\frac{1}{7})^{a_n}$.

Output: The first 10^N digits after the decimal point of L .

Problem C20. $[N = 4]$ $X = \sin(\frac{2\pi}{17})$.

Output: The 10^Nth partial quotient of the regular continued fraction of X .

Problem C21. $[N = 4]$ Equation $e^{\cos(x)} = x$.

Output: The first 10^N digits after the decimal point of x .

Problem C22. $[N = 3]$ $J = \int_0^1 \sin(\sin(\sin(x)))dx$

Output: The first 10^N digits after the decimal point of J .

Problem C23. $[N = 2]$ M_1=Inverse of the $10^N \times 10^N$ Hankel matrix X where $X_{ij} = \frac{1}{\mathsf{fib}(i+j-1)} \qquad (i, j > 0)$, $\mathsf{fib}(k) = \frac{(1+\sqrt{5})^k-(1-\sqrt{5})^k}{2^k\sqrt{5}}$.

Output: The first 10 decimal digits of the element $(10^N-1, 10^N-3)$ of M_1 .

Problem C24. $[N = 2]$ M_2=Inverse of $I_{10^N}+X$ (sum of the identity and the $10^N \times 10^N$ Hankel matrix X) where X is defined as in C23.

Output: The first 10 decimal digits of the element $(10^N-1, 10^N)$ of M_2 .

4.3 Problem C14

After the problems were announced, during the competition problem **C14** turned out to be poorly formulated by the organisers. The original problem **P14** as formulated in [14] is intended to highlight the effect of initial fluctuations introduced by early roundings leading to the convergence towards the wrong attractor. But by asking for the values *after* the decimal point this effect was neutralised: the wrong attractor and the correct solution would have the same digits after decimal point. This discrepancy was noticed by several of the participants but it was too late to change the formulation.

4.4 Answers for $N = 1$

As an initial assessment facility for the participants, together with the competition problems we also provided the set of answers for parameter when parameter N is taken to be 1 (Table 1).

5 Participating Systems

A call for participation was sent in July 2005. The participants and the system they used are shown in Table 2. To get a better impression of each system, the code of the solutions provided by each system is given in Appendix A.

#	N=1	#	N=1
C01	5645109298	**C13**	9452215206
C02	9301913671	**C14**	9999999999
C03	9094952410	**C15**	6102285184
C04	0891129268	**C16**	4673949577
C05	3313036085	**C17**	9992228377
C06	1237676104	**C18**	9159655941
C07	9679087443	**C19**	0204081880
C08	9539537434	**C20**	8
C09	9999999999	**C21**	3029640012
C10	0000000000	**C22**	4078390263
C11	5603103379	**C23**	9433912411
C12	8334468080	**C24**	1014362014

Table 1: **Answers for $N = 1$.**

#	Team/System	Contact person / Members
S1	Bignum[1] [6]	Martin Guy
S2	COMP	Yann Kieffer
S3	Few Digits [17]	Russell O'Connor
S4	GiNaC/CLN [8]	Richard B. Kreckel
S5	iRRAM [16]	Norbert Müller
S6	Maple[1] [10]	Jacques Carette
S7	mpfr team [7]	Paul Zimmermann, Vincent Lefèvre, Patrick Pélissier, Laurent Fousse, Guillaume Hanrot, Jim White
S8	RealLib [9]	Branimir Lambov
S9	Wolfram Research Team[1] [24]	Ed Pegg Jr

Table 2: **List of participating systems.**

6 Results of the Competition

Immediately after the end of the competition the initial *provisional* results and rankings were announced. The provisional results included the timings that were extracted from the `ttyrec` log files.

In the weeks following the competition the we reran all the solutions. This led to the final results and rankings of the competition. The timing were obtained by applying `/usr/bin/time` to the executable files for each solution and taken the real time (wall-clock time). The precise command and command-line parameters for each system and each problem can be found in Appendix B. The machine was reboot for each system and the remote connections were prohibited. Having a unified timing method inevitably resulted in slightly different timings compared to the provisional timings; the final rankings however, were the same as the provisional ranking.

[1]Remote participation.

The source code of each solution for each system can be found in Appendix A. Of course, the source code pertains to the solutions, i.e., they do not include the general purpose code of each system and used libraries and they only include parts *directly* contributing to the solution. In other words, the source code are intended to indicate what a *user* of the system in question should program to solve the problems in this competition.

The results of this final timing and ranking of each system for individual problems are presented in Table 3. The rankings are given according to the rules of Section 2. In particular, is a system could solve a problem with parameter N within two seconds but could not solve that problem with parameter $N + 1$ within two minutes, then in this table it receives a timing of 0 for parameter N. The symbols $+, *$ and $-$ after some of the timings are explained in Section 2.4.

After ranking each problem according to the procedure in Section 2.5 the scores are determined. The scores for the basic set of problems, intermediate problems and the final scores and ranking are presented in Table 4.

The overall winners were the mpfr team. The top 3 in the overall ranking and for each problem set were the following.

Overall Ranking (all 24 problems)

1. mpfr team
2. GiNaC/CLN
3. iRRAM

Basic Set

1. Wolfram Research Team
2. GiNaC/CLN
3. mpfr team

Intermediate Set

1. mpfr team
2. iRRAM
3. GiNaC/CLN

Full ranking can be found in Table 4.

The overall winners, the mpfr team published a document explaining their solutions for the competition [12]. Earlier they had provided a document explaining their solutions for the practice problems [13].

Table 3: **Results after rerunning the solutions** (Sx: team/system as in Table 2, CxxN: Parameter N, CxxT: time in seconds, CxxR: Ranking for that problem)

Problem		**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**
C01	C01N	1		3	5	5	5	5	4	5
	C01T	45.0		23.5+	17.4	26.4	96.9	18.1	2.1	10.9
	C01R	8	9	7	2	4	5	3	6	**1**
C02	C02N	2	5	3	6	5	5	6	4	5
	C02T	5.2	112.2+	2.3	13.6	2.9	7.5	32.8	11.6	0
	C02R	9	6	8	**1**	4	5	2	7	3
C03	C03N	2		3	5	5	5	5	4	5
	C03T	3.6		5.1+	6.5	8.9	45.7+	9.2	14.8	3.7
	C03R	8	9	7	2	3	5	4	6	**1**
C04	C04N	2	4	3	5	5	5	5	4	5
	C04T	34.5	4.9+	28.8	4.4	53.7	21.2+	11.8	12.0+	2.3
	C04R	9	6	8	2	5	4	3	7	**1**
C05	C05N	2	4	3	5	5	5	5	4	5
	C05T	99.0	13.3+	44.0+	6.7	11.1	33.8	5.2	18.2+	3.7
	C05R	9	6	8	3	4	5	2	7	**1**
C06	C06N			1	5	5	4	5	4	5
	C06T			0	12.3	21.3	3.5	8.3	31.2	7.0
	C06R	8	8	7	3	4	5	2	6	**1**
C07	C07N	2	4		6	6	5	6	4	6
	C07T	20.1	3.9		12.4	75.2	4.1	25.5	17.6+	11.8
	C07R	8	6	9	2	4	5	3	7	**1**
C08	C08N				5	5	5	5	4	5
	C08T				6.8	9.7	49.9	8.8	56.8	4.2
	C08R	7	7	7	2	4	5	3	6	**1**
C09	C09N			3	5	5	5	5	4	5
	C09T			88.2	13.8	19.6	67.2	33.6	20.4	7.1
	C09R	8	8	7	2	3	5	4	6	**1**
C10	C10N			2	5	6	4	5	4	6
	C10T			8.9	9.0−	36.2	0−	4.2	26.7	22.6
	C10R	8	8	7	4	2	5	3	6	**1**
C11	C11N			2	5	5	5	5	4	5
	C11T			21.2	15.6	29.5	99.4	24.5	20.8+	9.0
	C11R	8	8	7	2	4	5	3	6	**1**
C12	C12N			2	5	5	5	5	4	5
	C12T			14.3	9.4	25.1	54.6	8.6	20.7+	5.7
	C12R	8	8	7	3	4	5	2	6	**1**

Problem		**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**
C13	C13N		4		4	4	2	4	4	
	C13T		107.2		21.7	6.1	0+	9.2	84.0	
	C13R	7	5	7	3	**1**	6	2	4	7
C14	C14N				2	2	2	2	1	
	C14T				57.5+	30.4	36.0	45.4+	5.9+	
	C14R	6	6	6	4	**1**	2	3	5	6
C15	C15N		1		2	4		4	2	
	C15T		0		0	5.1		7.3	0	
	C15R	6	5	6	3	**1**	6	2	3	6
C16	C16N				4	4		4	3	
	C16T				2.5	0		0	6.5	
	C16R	5	5	5	3	**1**	5	**1**	4	5
C17	C17N				4	4	3	5	3	
	C17T				2.4	8.4	10.3	116.9	18.4	
	C17R	6	6	6	2	3	4	**1**	5	6
C18	C18N				5	5	4	5	3	
	C18T				9.6	48.6	3.0+	8.3	2.1	
	C18R	6	6	6	2	3	4	**1**	5	6
C19	C19N				6	6		6	5	
	C19T				67.5	38.1		3.5	47.5	
	C19R	5	5	5	3	2	5	**1**	4	5
C20	C20N				4	4		5		
	C20T				25.2	0		4.3		
	C20R	4	4	4	3	2	4	**1**	4	4
C21	C21N				4	5	3	5		
	C21T				9.8	18.9	0*	15.0		
	C21R	5	5	5	3	2	4	**1**	5	5
C22	C22N				2	3	2	3		
	C22T				79.4	20.0	0	26.6		
	C22R	5	5	5	4	**1**	3	2	5	5
C23	C23N				2	2		7	1	
	C23T				43.5	57.9		18.7	0	
	C23R	5	5	5	2	3	5	**1**	4	5
C24	C24N				2	2		2	2	
	C24T				0	0		0	0	
	C24R	5	5	5	**1**	**1**	5	**1**	**1**	5
Total Rank		9	7	8	2	3	5	**1**	6	4

Table 4: **Final Rankings** (Sx: team/system as in Table 2)

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**
C01R	8	9	7	2	4	5	3	6	**1**
C02R	9	6	8	**1**	4	5	2	7	3
C03R	8	9	7	2	3	5	4	6	**1**
C04R	9	6	8	2	5	4	3	7	**1**
C05R	9	6	8	3	4	5	2	7	**1**
C06R	8	8	7	3	4	5	2	6	**1**
C07R	8	6	9	2	4	5	3	7	**1**
C08R	7	7	7	2	4	5	3	6	**1**
C09R	8	8	7	2	3	5	4	6	**1**
C10R	8	8	7	4	2	5	3	6	**1**
C11R	8	8	7	2	4	5	3	6	**1**
C12R	8	8	7	3	4	5	2	6	**1**
C13R	7	5	7	3	**1**	6	2	4	7
C14R	6	6	6	4	**1**	2	3	5	6
C15R	6	5	6	3	**1**	6	2	3	6
C16R	5	5	5	3	**1**	5	**1**	4	5
C17R	6	6	6	2	3	4	**1**	5	6
C18R	6	6	6	2	3	4	**1**	5	6
C19R	5	5	5	3	2	5	**1**	4	5
C20R	4	4	4	3	2	4	**1**	4	4
C21R	5	5	5	3	2	4	**1**	5	5
C22R	5	5	5	4	**1**	3	2	5	5
C23R	5	5	5	2	3	5	**1**	4	5
C24R	5	5	5	**1**	**1**	5	**1**	**1**	5
Score Basic Set	98	89	89	28	45	59	34	76	14
Score Intermediate Set	65	62	65	33	21	53	17	49	65
Score Total	163	151	154	61	66	112	51	125	79
# 1st places	0	0	0	2	6	0	8	1	11
Ranking Basic Set	9	7	7	2	4	5	3	6	**1**
Ranking Intermediate Set	7	6	7	3	2	5	**1**	4	7
Final Ranking	9	7	8	2	3	5	**1**	6	4

7 Post-competition Enhanced Solutions

In the few weeks after the competition, the machine used during the competition was not immediately upgraded; hence it provided a brief opportunity to try new possible solutions on an environment identical to the one used during the competition. Some of the systems used this opportunity and submitted enhanced solutions leading to better timing for some of the problems.

The mpfr team posted enhanced solutions for all of the problems (Table 5).

#	N	time	#	N	time
C01	5	11.2	**C13**	4	5.6
C02	6	21.3	**C14**	2	44.7
C03	5	6.1	**C15**	4	7.1
C04	5	2.2	**C16**	4	0
C05	5	3.1	**C17**	5	116.1
C06	5	6.6	**C18**	5	7.0
C07	6	18.3	**C19**	6	2.9
C08	5	6.4	**C20**	6	16.6
C09	5	12.3	**C21**	5	9.8
C10	6	7.4	**C22**	3	19.6
C11	5	8.8	**C23**	19716	2.0
C12	5	5.7	**C24**	2	0

Table 5: **Post-competition solutions of mpfr team.**

Regarding the extraordinary value of N for **C23**, note that the mpfr team used a logarithmic method by Guillaume Hanrot [12] that could handle very large values of N within two minutes. While at $N = 19717$ it began to take slightly longer than two seconds. Hence according to the rules of the competition it was assigned this score. For this problem, the organisers had no other means of producing the answers apart from for those values that were scored during the competition ($N < 8$).

Three other teams had enhanced solution for one problem each (Table 6).

System	#	N	time
COMP	**C15**	2	106.7
iRRAM	**C20**	6	43.1
Maple	**C13**	4	30.4

Table 6: **Post-competition solutions of some problems.**

Apart from these two systems that did not participate in the competition submitted solutions for some of the problems *after* the competition (Table 7). The first set of solutions was submitted by Jean-Christophe Filliâtre using Creal library for OCaml [3]. The second set of solutions was by Jim White' DMXLIB system[2] based on GMP's floating point library.

[2]Student in B. Computational Science program of Stephen Roberts at the Australian National University.

System	#	N	time
Creal	**C01**	4	8.9
Creal	**C02**	4	3.7
Creal	**C03**	4	7.3
Creal	**C04**	4	15.3
Creal	**C05**	4	34.5
Creal	**C06**	4	85.1
Creal	**C07**	4	14.6
Creal	**C09**	4	41.7
Creal	**C10**	4	68.9
Creal	**C11**	4	29.0
DMXLIB	**C01**	5	5.4
DMXLIB	**C03**	5	2.5
DMXLIB	**C08**	5	3.3

Table 7: **Post-competition solutions of some problems.**

Acknowledgements. We would like to thank all the members of the participating teams and systems who made the competition possible. Their valuable feedback and suggestions helped enhance the quality of problems, correct several errors or ambiguities in our original formulation and test our system. They also helped us in finding the correct solutions. In this regard we thank Jacques Carette, Laurent Fousse, Martin Guy, Guillaume Hanrot, Yann Kieffer, Richard B. Kreckel, Branimir Lambov, Vincent Lefèvre, Norbert Müller, Russell O'Connor, Patrick Pélissier, Ed Pegg Jr, Jim White and Paul Zimmermann.

We also thank members of the community for their response to our initial call for suggesting problems. Keith Briggs, Norbert Müller and Paul Zimmermann were involved form the very beginning by giving valuable advise on the problems and assisted us by independently checking some of the answers. In addition we thank Richard Fateman, Jean-Christophe Filliâtre, Todd Rowland, Chee Yap and Roland Zumkeller.

We thank Vincent Lefèvre for several corrections regarding the material on the website. Special thanks are due to Lionel Elie Mamane for help and patience in preparing and maintaining the machine used as the competition environment. We also thank the members of Nijmegen Foundation group: Henk Barendregt, Herman Geuvers, Bas Spitters and Dan Synek. The first author was supported by the Netherlands Organisation for Scientific Research (NWO).

References

[1] J. Blanck. Exact real arithmetic systems: Results of competition. In J. Blanck, V. Brattka, and P. Hertling, editors, *Computability and Complexity in Analysis: 4th International Workshop, CCA 2000, Swansea, UK, September 17–19, 2000, Selected Papers*, volume 2064 of *Lecture Notes in Computer Science*, pages 389–393. Springer-Verlag, 2001.

[2] F. Bornemann, D. Laurie, S. Wagon, and J. Waldvogel. *The SIAM 100-digit challenge*. SIAM, Philadelphia, PA, 2004. A study in high-accuracy numerical computing, With a foreword by David H. Bailey.

[3] J.-C. Filliâtre. Creal library, 2005. `http://www.lri.fr/~filliatr/creal.en.html`.

[4] J. Gibbons. Unbounded spigot algorithms for the digits of pi. *Amer. Math. Monthly*, 113(4):318–328, 2006.

[5] R. W. Gosper. Acceleration of series. Memo 304, MIT AI Laboratory, Mar. 1974. available at `ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-304.pdf`.

[6] M. Guy. bignum / BigFloat, 2005. `http://bignum.sourceforge.net/`.

[7] G. Hanrot, V. Lefèvre, P. Pélissier, P. Zimmermann, S. Boldo, D. Daney, M. Dutour, E. Jeandel, L. Fousse, F. Rouillier, and K. Ryde. The MPFR Library, 2005. `http://www.mpfr.org/`.

[8] R. Kreckel. CLN - Class Library for Numbers, 2005. `http://www.ginac.de/CLN/`.

[9] B. Lambov. The RealLib Project, 2005. `http://www.brics.dk/~barnie/RealLib/`.

[10] M. Monagan, K. Geddes, K. Heal, G. Labahn, and S. Vorkoetter. *Maple V Programming Guide for Release 5*. Springer-Verlag, Berlin/Heidelberg, 1997.

[11] The MPFR Developers. Comparison of MPFR with other software. `http://www.mpfr.org/timings-mpfr1.0-gmp3.1.html`.

[12] The MPFR team. Solutions for the "many digits" friendly competition, Oct. 2005. `http://www.loria.fr/~zimmerma/mpfr/manydigits/compet.pdf`.

[13] The MPFR team. Solutions to the practice problems, Sept. 2005. `http://www.loria.fr/~zimmerma/mpfr/manydigits/practice.pdf`.

[14] J. M. Muller. *Arithmétique des Ordinateurs*. Masson, Paris, 1989.

[15] N. Müller. Software demonstration at the fifth workshop on computability and complexity in analysis (cca 2002), 2002. `http://www.informatik.uni-trier.de/~mueller/cca2002/`.

[16] N. Müller. iRRAM – Exact Arithmetic in C++, 2005. `http://www.informatik.uni-trier.de/iRRAM/`.

[17] R. O'Connor. Few Digits 0.4.0, 2005. `http://r6.ca/FewDigits/`.

[18] The PARI Group. Comparison of other software to MPFR. `http://pari.math.u-bordeaux.fr/benchs/timings-mpfr.html`.

[19] R. Pemantle and C. Schneider. When is 0.999 . . . equal to 1? *Amer. Math. Monthly*, 114(4):344–350, 2007.

[20] T. M. Richardson. The Filbert matrix. *Fibonacci Quart.*, 39(3):268–275, 2001.

[21] G. Strang. Learning from 100 numbers. *Science*, 307(5709):521–522, Jan. 2005.

[22] L. N. Trefethen. Chastened challenge sponsor: "I misjudged" (the $100, 100-digit challenge). *SIAM news*, 35(6), July/August 2002.

[23] L. N. Trefethen. A hundred-dollar, hundred-digit challenge. *SIAM news*, 35(1):65, January/February 2002.

[24] S. Wolfram. *The Mathematica book*. Cambridge University Press, Cambridge, 1996.

A Source Codes of the Solutions

A.1 Bignum

The Bignum solution is written in Haskell. There is one source file called `compete.hs` which holds all the solutions. It is compiled using the commands

```
ghc -O2 -fvia-C -c compete.hs
ghc -o compete compete.o BigFloat.o
```

and then invoked as

```
export GHCRTS=-H512m
time ./compete 1 10
```

This then prints something like

```
5645109298

real    0m44.891s
user    0m43.639s
sys     0m0.898s
```

The program `compete.hs` defines a main function `main`, a list of expressions of type `BigFloat` called `fns`, some abbreviations and a function `remove_up_to_dot`. The function `remove_up_to_dot` is defined as:

```
-- remove all characters from a string up to and including the first '.'
remove_up_to_dot :: [Char]->[Char]
remove_up_to_dot [] = []
remove_up_to_dot ('.':x) = x
remove_up_to_dot (a:x)
        | a == '.'      = x
        | otherwise     = remove_up_to_dot x
```

the `main` function is

```
-- program takes two parameters: the problem number and the precision
-- (precision n gives the first 10**n digits after the decimal point)
main :: IO ()
main = do
        hSetBuffering stdout NoBuffering
        [prob,n] <- getArgs
        -- problems are numbered from 1, arrays start at 0.
        putStrLn (take (read n) (remove_up_to_dot (show (fns!!(read prob - 1)))))
```

and the only abbreviations that are used are

```
arcsin=asin
arcsinh=asinh
arctanh=atanh
```

The expressions in the list `fns` are given as the solutions below.

Solution to Problem C01

```
sin(tan(cos(1)))
```

Solution to Problem C02

```
sqrt(e/pi)
```

Solution to Problem C03

```
sin((e+1)^3)
```

Solution to Problem C04

```
exp(pi*(sqrt(2011)))
```

Solution to Problem C05

```
exp(exp(exp(1/2)))
```

Solution to Problem C06

```
arctanh(1-arctanh(1-arctanh(1-arctanh(1/pi))))
```

Solution to Problem C07

```
pi**1000
```

Solution to Problem C08

```
sin(6**(6**6))
```

Solution to Problem C09

```
sin(10*atan(tanh(pi*(2011**(1/2))/3)))
```

Solution to Problem C10

```
(7+2**(1/5)-5*(8**(1/5)))**(1/3) + 4**(1/5)-2**(1/5)
```

Solution to Problem C11

```
tan(2**(1/2))+arctanh(sin(1))
```

Solution to Problem C12

```
arcsin(1/(e*e)) + arcsinh(e*e)
```

A.2 Comp

The Comp solution consists of a collection of Haskell programs, one for every problem. For instance the solution to Problem C02 is in the file `C02.hs`, which is reproduced in full as subsection A.2 below. This is the source of a program `C02`, which is compiled with the command

```
ghc --make C02 -o C02 -prof -auto-all
```

and then run as

```
time ./C02 10000 > C02.out
```

Solution to Problem C02

```
module Main(main) where

import IO
import System(getArgs)
import Reals1
import MDF

main = do args <- getArgs
          let prec = read (head args)
          print (c02 prec)
            where
               c02 k = mdFormat (srn (realSqrt (NNR 1 (realPartBR
                    (realDiv (BR 2 (realPartNNR (realExp (BR 0 one))))
                             (NNR (-1) realPi) ) ))) k)
```

Solution to Problem C04

```
module Main(main) where

import IO
import System(getArgs)
import Reals1
import MDF

main = do args <- getArgs
          let prec = read (head args)
          print (c04 prec)
            where
c04 k = mdFormat (srn (realPartNNR (realExp (realProd
                      (BR 2 realPi)
                      (BR 6 (realSqrt (NNR (-10) (realFromInteger 2011) ))) )))
                k)
```

Solution to Problem C05

```
module Main(main) where

import IO
import System(getArgs)
import Reals1
import MDF

main = do args <- getArgs
          let prec = read (head args)
          print (c05 prec)
            where
c05 k = mdFormat (srn
           (realPartNNR (realExp (BR 3 (realPartNNR (
              realExp (BR 1 (realPartNNR (realExp (BR (-1) half))))))))) k)
```

Solution to Problem C07

```
module Main(main) where

import IO
import System(getArgs)
import Reals1
import MDF

main = do args <- getArgs
          let prec = read (head args)
          print (c07 prec)
            where
c07 k = mdFormat (srn (realPartBR p1000) k)
  where p1000 = realSq p500
        p500 = realSq p250
        p250 = realSq p125
        p125 = realProd p100 p25
        p100 = realSq p50
        p50 = realSq p25
        p25 = realProd p20 p5
        p20 = realSq p10
        p10 = realSq p5
        p5 = realProd p4 p1
        p4 = realSq p2
        p2 = realSq p1
        p1 = BR 2 (approx realPi (k+20))
```

Solution to Problem C13

```
module Main(main) where

import IO
import System(getArgs)
import Reals1
import MDF

main = do args <- getArgs
          let prec = read (head args)
          let m = read (head (tail args))
          print (c13 m prec)
            where
```

```
newLogMapApprox :: RealN -> Index -> RealN
newLogMapApprox x k = realPartBR (
                  realProd (realProd (BR 2 (rfr (3999%1000)))  (BR 0 x1))
                                   (realToLogBoundedReal (realDiff one x1)) )
                                  where x1 = approx x (k+2)

newLogMap :: RealN -> RealN
newLogMap = realExactFunction newLogMapApprox

c13 n k = mdFormat (srn (iterateN newLogMap half n) k)
```

Solution to Problem C15

```
module Main(main) where

import IO
import System(getArgs)
import Reals1
import MDF

main = do args <- getArgs
          let prec = read (head args)
          let m = read (head (tail args))
          print (c15 m prec)
            where
c15 n k = mdFormat (srn (realSumList2 (n * (n-1) + 1)
           (map (\i -> realPartNNR (realInv (NNR (binaryLogCeil i) (rfi i))))
                [n..n*n])) k)
```

A.3 Few Digits

The Few Digits solution is written in Haskell. It consists of a main file called `CCA.hs`, that holds the solutions. Also for every problem there is a 'stub' file. For instance for Problem C01 there also is a file called `P01.hs` that holds the text

```
module Main where
import CCA
main = putStrLn $ ans 1000 (problems!!0)
```

This file `P01.hs` is compiled by

```
ghc --make -O2 P01.hs
```

and then run as

```
time ./a.out > P01.out
```

The file `CCA.hs` defines a list called `problems` which consists of expressions of type `CReal`. These expressions are reproduced as the solutions below. Furthermore it defines the function `ans`, which is defined as

```
ans n x = (replicate (n-(length post)) '0')++post
 where
  pre = show $ intApprox (realScale (10^n) x)
  post = drop ((length pre)-n) pre
```

Solution to Problem C01

```
sin (tan (cos 1))
```

Solution to Problem C02

```
sqrt (e/pi)
```

Solution to Problem C03

```
sin (realPowerInt (e+1) 3)
```

Solution to Problem C04

```
exp (pi * sqrt 2011)
```

Solution to Problem C05

```
exp (exp (sqrt e))
```

Solution to Problem C06

```
atanh $ realTranslate 1
    $ - (atanh $ realTranslate 1
          $ - (atanh $ realTranslate 1
                $ - (atanh $ recip pi)))
```

Solution to Problem C07

```
pi**1000
```

Solution to Problem C08

```
sin (fromRational (6^(6^6)))
```

Solution to Problem C09

```
sin (realScale 10 (atan(tanh(realScale (1/3) (pi*sqrt(2011))))))
```

Solution to Problem C10

```
(realTranslate 7 ((2**(1/5))-(realScale 5 ((8)**(1/5)))))**(1/3) + (4**(1/5)) - (2**(1/5))
```

Solution to Problem C11

```
tan (sqrt 2) + atanh(sin(1))
```

Solution to Problem C12

```
asin (1/(exp 2)) + asinh(exp(2))
```

A.4 GiNaC

The GiNaC solution has one C++ file per problem, which are reproduced in full in the subsections below. For instance the file `c01.cc` is reproduced in subsection A.4. This program is compiled by the commands

```
c++ -I../project/include -O2 c01.cc libclnginac.a -o c01
echo "stripping c01..."; strip c01 && strip -R .note c01 && strip -R .comment c01
```

and run as

```
time ./c01 > c01.result
```

which gives output like

```
./c01 with 10^N==100000 at 100000 digits precision.
17.387s real, 16.666s user, 0.371 system
```

Solution to Problem C01

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cout << fractional_part(sin(tan(cos(cl_float(1)))),nn) << endl;
}
```

Solution to Problem C02

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
```

```
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 1000000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cout << fractional_part(sqrt(exp1()/pi()),nn) << endl;
}
```

Solution to Problem C03

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
```

```
         << " at " << Digits << " digits precision." << endl;

    cout << fractional_part(sin(expt(exp1()+1,3)),nn) << endl;
}
```

Solution to Problem C04

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn+60;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cout << fractional_part(exp(pi()*sqrt(cl_float(2011))),nn) << endl;
}
```

Solution to Problem C05

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
```

```
            << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cout << fractional_part(exp(exp(exp(recip(cl_float(2))))),nn) << endl;
}
```

Solution to Problem C06

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cout << fractional_part(the<cl_R>(atanh(1-atanh(1-atanh(1-atanh(recip(pi()))))),nn) << endl;
}
```

Solution to Problem C07

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
```

```
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 1000000;
    Digits = nn+500;  // Pi^1000 is approx. 10^498
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cout << fractional_part(expt(pi(),1000),nn) << endl;
}
```

Solution to Problem C08

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    // That argument is pervertly large.  We need to crank up the working precision
    // by as much as there are decimal digits in the argument:
    cl_I argument = expt_pos(6,expt_pos(6,6));
    unsigned nn = 100000;
    Digits = unsigned(nn + 1 + 0.43429448190325*double_approx(the<cl_F>(log(cl_float(argument)))))/10*10;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
```

```
         << " at " << Digits << " digits precision." << endl;

    cout << fractional_part(sin(argument),nn) << endl;
}
```

Solution to Problem C09

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    //sin(10*arctan(tanh(pi*(2011^(1/2))/3))) ~= 1, but not quite
    cout << fractional_part(sin(10*atan(tanh(pi()*sqrt(cl_float(2011))/3))),nn) << endl;
}
```

Solution to Problem C10

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/complex.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
```

```
            << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    //(7+2^(1/5)-5*(8^(1/5)))^(1/3) + 4^(1/5)-2^(1/5)
    const cl_N fifthrt2 = expt(2,recip(cl_float(5)));
    const cl_N fifthrt4 = square(fifthrt2);
    cout << fractional_part(expt(7+fifthrt2-5*fifthrt2*fifthrt4,recip(cl_float(3))) + fifthrt4 - fifthrt2,nn
}
```

Solution to Problem C11

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    //tan(2^(1/2))+arctanh(sin(1))
    cout << fractional_part(tan(sqrt(cl_float(2)))+atanh(sin(1)),nn) << endl;
}
```

Solution to Problem C12

```
#include "manydigits.cc"
#include <iostream>
```

```
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    //arcsin(1/e^2) + arcsinh(e^2)
    cl_F e2 = square(exp1());
    cout << fractional_part(asin(recip(e2))+asinh(e2),nn) << endl;
}
```

Solution to Problem C13

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/rational.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    // 1000 needs 1290 digits
    // 10000 needs 12940 digits
    unsigned nn = 10000;
    Digits = unsigned(1.295*nn+1)/10*10;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
```

```
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cl_F x = recip(cl_float(2));
    cl_F y = cl_float(3999)*recip(cl_float(1000));
    for (unsigned i=0; i<nn; ++i) {
        x = (x-square(x))*y;
    }
    cout << fractional_part(x,nn) << endl;
}
```

Solution to Problem C14

```
#include "manydigits.cc"
#include <cmath>
#include <cassert>
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/rational.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    // n==10  -> Digits==1230  (*123)
    // n==20  -> Digits==2460  (*123)
    // n==30  -> Digits==3690  (*123)
    // n==40  -> Digits==4930  (*123.25)
    // n==50  -> Digits==6160  (*123.2)
    // n==100 -> Digits==12310 (*123.1)
    unsigned nn = 100;
    Digits = unsigned(nn*123.3+1)/10*10;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cl_F a = cl_float(14/cl_RA(3));
    cl_F b = cl_float(5);
```

```
    cl_F c = cl_float(184/cl_RA(35));
    cl_F ainv = recip(a);
    cl_F binv = recip(b);
    cl_F cinv = recip(c);
    const cl_I A = 114;
    const cl_I B = 1463;
    const cl_I C = 6390;
    const cl_I D = 9000;
    for (unsigned i=0; i<nn*100; ++i) {
        cl_F x = c, xinv = cinv;
        c = the<cl_F>(A-(B-(C-D*ainv)*binv)*cinv);
        cinv = recip(c);
        a = b;  ainv = binv;
        b = x;  binv = xinv;
    }

    assert(nearbyint(float_approx(a))==6);
    cout << fractional_part(a,nn) << endl;
}
```

Solution to Problem C15

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/rational.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    // In GiNaC notation:
    // sum(1/k,k={m,m^2})==psi(1+m^2)-psi(m)
    // with m==10*nn: psi(1+nn*nn*100)-psi(nn*10).  But that is not any faster
    // because psi internally falls back to just that sum! Oh, dear...
    cl_F x = cl_float(0);
    for (unsigned i=nn*10; i<=nn*nn*100; ++i) {
```

```
        x = x + recip(cl_float(i));
    }
    cout << fractional_part(x,nn) << endl;
}
```

Solution to Problem C16

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

static const cl_F f(int ii)
{
    cl_F x = cl_float(1);
    for (int i=ii; i>0; --i) {
        x = the<cl_F>(x*expt_pos(i,3)/8/expt_pos(2*i+1,3));
        x = x+cl_float(21*i-8)/8;
    }
    return square(pi())/6-x;
}

int main(int argc, const char* argv[])
{
    unsigned nn = 10000;
    Digits = unsigned(2.81*nn+1)/10*10;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cout << nonzero_digits(f(nn),nn) << endl;
}
```

Solution to Problem C17

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/rational.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
```

```
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 10000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    // -4*Zeta(2) - 2*Zeta(3) + 4*Zeta(2)*Zeta(3) + 2*Zeta(5)
    cl_F z2 = zeta(2);
    cl_F z3 = zeta(3);
    cout << fractional_part(-4*z2-2*z3+4*z2*z3+2*zeta(5),nn) << endl;
}
```

Solution to Problem C18

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/rational.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 100000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
```

```
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cout << fractional_part(catalanconst(),nn) << endl;
}
```

Solution to Problem C19

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/rational.h>
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 1000000;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cl_F result = cl_float(0);
    cl_RA b = 1/cl_RA(7);
    cl_F x = cl_float(b);
    int n = 1;
    while (-float_exponent(x)<nn*3.3219281) {
        x = cl_float(expt_pos(b,n*n*n+1));
        result = result + x;
        ++n;
    }
    cout << fractional_part(result,nn) << endl;
}
```

Solution to Problem C20

```
#include "manydigits.cc"
#include <iostream>
#include <cln/float.h>
#include <cln/integer.h>
#include <cln/rational.h>
```

```
#include <cln/float_io.h>
#include <ginac/numeric.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 10000;
    Digits = unsigned(1.031*nn+1)/10*10;  // gambling...
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    cl_R x = sin(pi()*2/17);

    // Check sign.
    if (minusp(x)) {
        x = -x;
    }
    cl_R_div_t x_split;
    for (unsigned r=0; r<=nn; ++r) {
        // Split x into integral and fractional part.
        x_split = floor2(x);
        x = x_split.remainder;
        if (zerop(x)){
            break;
        }
        x = recip(x);
    }
    // ..., 1, 2, 17, *1*, 3, ...
    cout << x_split.quotient << endl;
}
```

Solution to Problem C21

```
#include "manydigits.cc"
#include <iostream>
#include <ginac/ginac.h>
using namespace std;
using namespace GiNaC;

int main(int argc, const char* argv[])
{
    unsigned nn = 10000;
    Digits = nn;
    if (argc>3) {
```

```
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    symbol x("x");
    relational r = exp(cos(x))==x;  // 1.302964...
    cout << fractional_part(fsolve(r,x,1,2),nn) << endl;
}
```

Solution to Problem C22

```
#include "manydigits.cc"
#include <iostream>
#include <vector>
#include <cln/cln.h>
#include <ginac/ginac.h>
using namespace std;
using namespace cln;
using namespace GiNaC;

// Okay, we don't know about numerical integration. We lose. :-)
// Let's just copy Romberg's method from Numerical Recipes in C and see
// that it converges at all.

void
polint(cl_R xa[], cl_R ya[], int n, const cl_R& x, cl_R* y, cl_R* dy)
{
    int i, m, ns=1;
    cl_R den, dift, ho, hp, w;
    cl_R dif = abs(x-xa[1]);
    vector<cl_R> c; c.resize(n+1);
    vector<cl_R> d; d.resize(n+1);
    for (i=1;i<=n;++i) {
        if ((dift=abs(x-xa[i]))<dif){
            ns = i;
            dif = dift;
        }
        c[i] = ya[i];
        d[i] = ya[i];
    }
    *y = ya[ns--];
    for (m=1;m<n;m++) {
        for (i=1;i<=n-m;i++) {
            ho = xa[i]-x;
            hp = xa[i+m]-x;
            w = c[i+1]-d[i];
            if (zerop(den=ho-hp)) {
                cerr << "Two identical input xa's?\n";
```

```
                abort();
            }
            den = w/den;
            d[i] = hp*den;
            c[i] = ho*den;
        }
        *dy = (2*ns < (n-m) ? c[ns+1] : d[ns--]);
        *y = *y + *dy;
    }
}

const cl_R
trapzd(const ex& func, const symbol& var,
       const cl_R& a, const cl_R& b, int n)
{
    cl_R x, tnm, sum, del;
    static cl_R s = cl_float(0);
    int it, j;
    if (n==1) {
        s = (b-a)/2
            *the<cl_R>((ex_to<numeric>(func.subs(var==numeric(a)).evalf()
                                       +func.subs(var==numeric(b)).evalf())).to_cl_N());
        return s;
    } else {
        for (it=1,j=1;j<n-1;j++)
            it <<= 1;
        tnm = cl_float(it);
        del = (b-a)/tnm;
        x = a+del/2;
        for (sum=cl_float(0),j=1;j<=it;j++,x=x+del)
            sum = sum + the<cl_R>(ex_to<numeric>(func.subs(var==numeric(x)).evalf()).to_cl_N());
        s = (s+(b-a)*sum/tnm)/2;
        return s;
    }
}

const numeric
qromb(const ex& func, const symbol& var,
      const numeric& a_in, const numeric& b_in)
{
    const cl_R a = the<cl_R>(ex_to<numeric>(a_in.evalf()).to_cl_N());
    const cl_R b = the<cl_R>(ex_to<numeric>(b_in.evalf()).to_cl_N());
    // JMAX and K must be made a function of Digits.
    const unsigned K = 20;
    const unsigned JMAX = 22;  // Well, that must be obviously a function of K.
    cl_R ss, dss;
    cl_R s[JMAX+2], h[JMAX+2];
    h[1] = cl_float(1);
    for (int j=1;j<=JMAX;++j) {
        s[j] = trapzd(func,var,a,b,j);
        if (j>=K) {
            polint(&h[j-K],&s[j-K],K,cl_float(0),&ss,&dss);
            if (abs(dss)<abs(ss)/1000000) {
                return ss;
            }
        }
        s[j+1] = s[j];
        h[j+1] = h[j]/4;
```

```
    }
    abort();  // notreached
}

int main(int argc, const char* argv[])
{
    unsigned nn = 100;
    Digits = nn;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    symbol x("x");
    ex f = sin(sin(sin(x)));
    cout << fractional_part(qromb(f,x,0,1),nn) << endl;
}
```

Solution to Problem C23

```
#include "manydigits.cc"
#include <iostream>
#include <ginac/ginac.h>
using namespace std;
using namespace GiNaC;

const matrix
hankel_matrix(unsigned rank)
{
    matrix m(rank,rank);
    for(unsigned d=0; d<2*rank-1; ++d) {
        ex f = 1/fibonacci(d+1);
        int ro = min(d,rank-1);
        int co = d-ro;
        for(;ro>-1&&co<rank;++co,--ro) {
            m(ro,co) = f;
        }
    }
    return m;
}

const ex
inverse_element(const matrix& m, unsigned r, unsigned c)
{
    // return inverse(m)(r,c) would be slower.
    return ((r+c)%2?-1:1) * determinant(ex_to<matrix>(reduced_matrix(m,r,c)))
                          / determinant(m);
}
```

```
int main(int argc, const char* argv[])
{
    unsigned nn = 100;
    Digits = 4110;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    matrix h = hankel_matrix(nn);

    cout << first_digits(inverse_element(h,nn-2,nn-4),10) << endl;
}
```

Solution to Problem C24

```
#include "manydigits.cc"
#include <iostream>
#include <ginac/ginac.h>
using namespace std;
using namespace GiNaC;

const matrix
hankel_plus_id_matrix(unsigned rank)
{
    matrix m(rank,rank);
    for (unsigned d=0; d<2*rank-1; ++d) {
        ex f = evalf(ex(1/fibonacci(d+1)));
        int ro = min(d,rank-1);
        int co = d-ro;
        for (;ro>-1&&co<rank;++co,--ro) {
            m(ro,co) = f;
        }
    }
    for (unsigned i=0; i<rank; ++i) {
        m(i,i) += 1;
    }
    return m;
}

const ex
inverse_element(const matrix& m, unsigned r, unsigned c)
{
    // return inverse(m)(r,c) would be slower.
    return ((r+c)%2?-1:1) * determinant(ex_to<matrix>(reduced_matrix(m,r,c)))
                          / determinant(m);
}

int main(int argc, const char* argv[])
```

```
{
    unsigned nn = 100;
    Digits = 10;
    if (argc>3) {
        clog << "usage: " << argv[0]
             << " [prec (" << Digits << ")"
             << " [10^N (" << nn << ")]]\n";
        return 1;
    }
    if (argc>2) {
        nn = atoi(argv[2]);
    }
    if (argc>1) {
        Digits = atoi(argv[1]);
    }
    clog << argv[0] << " with 10^N==" << nn
         << " at " << Digits << " digits precision." << endl;

    matrix h = hankel_plus_id_matrix(nn);

    // result is 2445157027
    cout << first_digits(inverse_element(h,nn-2,nn-1),10) << endl;
}
```

A.5 iRRAM

The iRRAM solution consist of one C++ program per problem, which are reproduced in full as the solutions below. For instance the solution to Problem C01 is given by the program `C01.cc`, which is listed in subsection A.5. This program is compiled (using a `Makefile`) with the command

```
g++ -g -O2 -DGMP -I/home/irram/iRRAM_current/installed/include -Xlinker -rpath \
  -Xlinker /home/irram/iRRAM_current/installed/lib C01.cc  -liRRAM-GMP -lmpfr  \
  -L/home/irram/iRRAM_current/installed/lib -lgmp -lm -o C01
```

and then run from a `bash` script called `run.rc` as

```
try 01 100000
```

where the function `try` is defined as

```
try () {
echo Problem $1 with Parameter $2
echo ."$1"."$2"."$3".
echo  "Problem $1,  $2 " >>timings
  export pf=1.05
  if [ "x$3" != "x" ]; then export pf=$3; fi
  echo $2 | time ./C$1 -prec_factor=$pf 2>> timings |tee  C$1-$2.result
}
```

This then prints

```
Problem 01 with Parameter 100000
.01.100000..
Problem C01: sin(tan(cos(1)))
Please enter parameter N:
many many digits
```

It also creates a file called `C01-100000.result` that also holds the problem statement and those 100,000 digits.

Solution to Problem C01

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C01: sin(tan(cos(1)))\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

nij_print_a( sin(tan(cos(REAL(1)))) , N , 1);

};
```

Solution to Problem C02

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C02: sqrt(e/pi)\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

nij_print_a( sqrt(my_euler2()/pi()) , N , 1);

};
```

Solution to Problem C03

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C03: sin((e+1)^3)\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

nij_print_a( sin(power(my_euler2()+1,3))  , N , 1);

};
```

Solution to Problem C04

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C04: exp (pi * sqrt(2011))\n";
cout << "Please enter parameter N: ";
```

```
cin  >> N ;
cout << "\n";

nij_print_a( exp (pi() * sqrt(REAL(2011)))  , N ,20);

};
```

Solution to Problem C05

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem P05: exp(exp(sqrt(e)))\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

nij_print_a( my_exp(my_exp(sqrt(my_euler2()) )  )  , N , 7);

};
```

Solution to Problem C06

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem P06: atan (1-(atan(1-atan(1-atan(1/pi())))))\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

nij_print_a( atanh (1-(atanh(1-atanh(1-atanh(1/pi())))))) , N , 1);

};
```

Solution to Problem C07

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C07: pi^1000\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

nij_print_a( power(pi(), 1000)  , N , 500);

};
```

Solution to Problem C08

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C08: sin(6^6^6)\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

hint(N);
REAL x=INTEGER(6)^46656;
nij_print_a( sin(x) , N , 1);

};
```

Solution to Problem C09

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C09: sin(10*atan(tanh(pi*sqrt(2011)/3))) \n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

nij_print_a( sin(10*my_atan(my_tanh(pi()*sqrt(REAL(2011))/3))) , N , 1);

};
```

Solution to Problem C10

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C10: mixed roots\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

REAL a7   = 7;
REAL r5_2 = n_root(REAL(2),5);
REAL r5_4 = square(r5_2);
REAL r5_8 = r5_4*r5_2;
REAL t1   = n_root( a7 + r5_2 -5*r5_8, 3 );
REAL x    = t1+ r5_4 - r5_2;

nij_print_c( x , N , 1);

};
```

Solution to Problem C11

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C11: tan(sqrt(2)) + arctanh (sin(1))\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

REAL e=my_euler2();
REAL x= my_tan(sqrt(REAL(2))) + atanh(sin(REAL(1)));

nij_print_a( x , N , 1);

};
```

Solution to Problem C12

```
#include "iRRAM.h"
#include "nij_print.h"
#include "iRRAM_limit_templates.h"

void compute(){

int N;

cout << "Problem C12: arcsin (1/e^2) + arcsinh (e^2)\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

int DP=10;
for (int i=2;i<=N;i++) DP=DP*10;

REAL e2 = square( my_euler2() );
REAL x  = my_asin (1/e2) +  asinh (e2);

nij_print_a( x , N , 1);

};
```

Solution to Problem C13

```
#include "iRRAM.h"
#include "nij_print.h"

int c_1;
int c_2;

REAL iteration(const REAL& x) {REAL diff=x-square(x); return scale(diff,c_1) -diff/c_2;}
bool iteration_dom(const REAL& x) {return true;}
REAL iteration_lip(const REAL& x) {REAL y=1-scale(x,1); return scale(y,c_1)-y/c_2;}

void compute(){

int N;

cout << "Problem C13: logistic map\n";
cout << "Please enter parameter N: ";
cin  >> N ;
```

```
cout << "\n";

hint(int(1.25*N));

  c_1=2;
  c_2=1000;

  REAL x = 0.5;
  REAL diff;
  for ( int i=1; i<=N; i++ ) {
   x= lipschitz(iteration,iteration_lip,iteration_dom,x);
  }

nij_print_a( x , N , 1);

};
```

Solution to Problem C14

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){

int N;

cout << "Problem C14: iteration\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

int count=N*100;

hint(int(N*N*1.5));

REAL a=REAL(14)/3, b(5), c=REAL(184)/35, d;

for (long i=2;i<=count;i++ ) {
    d=114-(1463-(6390-(9000/a))/b)/c;
    a=b; b=c; c=d;
  }

nij_print_a( a , N ,1);

}
```

Solution to Problem C15

```
#include "iRRAM.h"
#include "nij_print.h"

const int bernoulli_max=20000;
RATIONAL bernoulli_saved[bernoulli_max/2];
INTEGER binomial_saved[bernoulli_max];
unsigned int bernoulli_min_unknown=0;

RATIONAL bernoulli(const unsigned int index)
{
if (index >= bernoulli_max ){ fprintf(stderr,"not yet implemented\n"); exit(1); }

  if ( index   == 1 ) return RATIONAL(-1,2);
  if ( index%2 == 1 ) return RATIONAL(0);
  if (bernoulli_min_unknown == 0){
     bernoulli_saved[0]=RATIONAL(1);
```

```
    binomial_saved[0]=1;
    binomial_saved[1]=2;
    binomial_saved[2]=1;
    bernoulli_min_unknown=2;
  }
  while ( bernoulli_min_unknown <= index ) {
    INTEGER saved_value=1;
    INTEGER old_saved_value;
    for ( int i = 1; i <= bernoulli_min_unknown; i++ ){
      old_saved_value=saved_value;
      saved_value=binomial_saved[i];
      binomial_saved[i] += old_saved_value;
    }

    binomial_saved[bernoulli_min_unknown+1]=1;

    if ( bernoulli_min_unknown%2==0 ){
      RATIONAL summe=bernoulli_saved[0]+binomial_saved[1]*RATIONAL(-1,2);
      for ( int i = 2; i < bernoulli_min_unknown; i+=2 ){
        summe=summe + bernoulli_saved[i/2]*binomial_saved[i];
      }
      summe= -1*summe/binomial_saved[bernoulli_min_unknown];
      bernoulli_saved[bernoulli_min_unknown/2]= summe;
    }
    bernoulli_min_unknown++;
  }
  return bernoulli_saved[index/2];
}

REAL E_M_gamma_approx(int p)
{
  if (p > 0 )return REAL(0.57721566490);

  int n=(2-p)/11;
  int k=0;
  int beta_n;
  REAL k2;
  REAL n2;

  beta_n=5*n;
  REAL s0=0;
  REAL i0=1;
  REAL b=1;
  REAL a=0;

  n2=REAL(n)*n;
  while ( k < beta_n  ) {
     k=k+1;
     if (k<30000) {
         b=(b*n2)/(k*k);
         a=(a*n2+b*k)/(k*k);
     } else {
          k2=REAL(k)*k;
          b=(b*n2)/k2;
          a=(a*n2+b*k)/k2;
    }
     i0+=b;
     s0+=a;
  }
```

```
  k=0;
  REAL k0=1;
  REAL y=1;
  REAL nom;
  while ( k < 4*n ) {
     k=k+1;
     if (k<15000) {
       y=y*((2*k-1)*(2*k-1));
     } else {
       nom=(2*k-1);
       nom*=(2*k-1);
       y=y*nom;
     }
     y= y  / (-16*k) / n;
     k0+=y;
  }
  k0=sqrt(pi()/REAL(4*n))*exp(REAL(-2*n))*k0;

  return (s0-k0)/i0-log(REAL(n));
};

REAL E_M_gamma(){  return limit(E_M_gamma_approx);  };

REAL B_sum_approx(int p,const REAL& x){
stiff_begin();
REAL x2=1/x/x;
REAL y=x2;
REAL term;
REAL sum=0;
int n=1;
while (true) {
  term=REAL(bernoulli(2*n))/2/n*y;
  sum=sum+term;
  if (bound(term,p)) break;
  n++;
  y=y*x2;
}
stiff_end();
return sum;
}

REAL digamma_test(const REAL & x ){ return log(x)- 1/x/2-
limit(B_sum_approx,x);
};

void compute(){

int N;

cout << "Problem P15: harmonic series\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

INTEGER count1=N*10;
INTEGER count2=count1*count1;

REAL dlg2=digamma_test(REAL(count2)+1);
REAL E_M_g=E_M_gamma();

REAL h1=0;
```

```
REAL one = 1;

for (int i=count1-1;i>=1;i-- ) {
    h1+=one/i;
   }
REAL result=E_M_g + dlg2- h1;
nij_print_a(result , N , 1);

return;

REAL h2= 0;

for (int i=count2;i>=1;i-- ) {
    h2+=one/i;
   }
cout << setRwidth(N+10)<< h2-h1 <<"\n";

}
```

Solution to Problem C16

```
#include "iRRAM.h"
#include "nij_print.h"

REAL f(int n){
REAL prod=1;
REAL sum =0;
int  p1=1;
REAL p2=1,p3=1;

REAL sum_part=0;
int  i_part=10;

for (int i=1;i<=n;i++){
  p1 = 21*i-8;
  sum_part = sum_part + p1*prod/8;
  if (i==i_part){ sum=sum+sum_part;sum_part=0;i_part=i_part+10;}
  if ( i<250) {
          p3=8*(2*i+1)*(2*i+1)*(2*i+1);
          p2=i*i*i;
  } else if ( i < 20000 ){
          p2 = REAL(i)*(i*i);
          p3 = REAL(8*(2*i+1))*((2*i+1)*(2*i+1));
  } else {
          p2 = (REAL(i)*i)*i;
          p3 = (REAL(8*(2*i+1))*(2*i+1))*(2*i+1);
  }
  prod=prod*p2/p3;
}
sum=sum+sum_part;
sum=sum+prod;
return square(pi())/6-sum;
}

void compute(){

int N;

cout << "Problem C16: pi^2/6 as iterated product\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";
```

```
hint(int(N*2.90));

REAL x=f(N);
nij_print_b( x , N);

}
```

Solution to Problem C17

```
#include "iRRAM.h"
#include "nij_print.h"

int inline top(const REAL &x) {
        int rnd = round(x);
        if (rnd > x)
                return rnd;
        return rnd + 1;
}

INTEGER inline pow(int base, int exp) {
        INTEGER res(base);
        while (--exp > 0)
                res *= base;
        return res;
}

REAL inline c(REAL cmm, int n, int k) {
        int nk = n - k + 1;
        if (n+k < 40000)
                return cmm * (2 * (n+k-1) * nk) / ((2*k-1) * k);
        else
                return cmm * (2 * (n+k-1)) * nk / (2*k-1) / (k);
}

REAL zeta_n(int n, int x) {
        REAL sum1(0), sum2(0);
        int fact = 1;

        REAL ck = REAL(1) / n;
        REAL dk = REAL(1);

        for (int kpp = 1; kpp <= n; kpp++) {
                sum1 += dk * fact / REAL(pow(kpp,x));
                ck = c(ck, n, kpp);
                dk = dk + ck * n;
                fact *= -1;
        }

        REAL dnn = dk;
        fact = 1;
        for (int kpp = 1; kpp <= n; kpp++) {
                sum2 += fact * dnn / REAL(pow(kpp,x));
                fact *= -1;
        }

        return (sum2 - sum1) / (dnn - scale(dnn, 1-x));
}

REAL zeta_p(int p, const REAL &foo, int x) {
        int n = 3-p;
        return zeta_n(n, x);
}
```

```
REAL zeta(int x) {
        if (x==2) return square(pi())/6;
        REAL foo(42);
        return limit<REAL, int, REAL>(zeta_p, foo, x);
}

REAL apery_n(int n) {
        REAL sum(0);
        for (int i = 1; i < n; i++)
                sum += REAL(1) / (exp(pi()*2*i)-1) / i / i / i;
        return REAL(7) / 180 * pi()*pi()*pi() - sum * 2;

}

void compute() {

        int N;

        cout << "Problem C17: zeta function\n";
        cout << "Please enter parameter N: ";
        cin  >> N ;
        cout << "\n";

        REAL zeta2 = zeta(2) * 4;
        REAL zeta3 = zeta(3);
        REAL x = 2 * zeta(5) - zeta2 - 2 * zeta3 + zeta2 * zeta3;

        nij_print_a( x , N , 2);;

}
```

Solution to Problem C18

```
#include "iRRAM.h"
#include "nij_print.h"

REAL catalan_approx_slow(int p){

 INTEGER m;
 int n=1;
 int sgn=1;
 REAL sum = 0;
 REAL term;
 while (true) {
   term=1/REAL(sgn*n*n);
cerr << n ;
   sgn=-sgn;
   sum = sum + term;
   if (bound(term,p)) break;
   n+=2;
sum.rcheck();
 }
 return sum;
}

REAL catalan_approx(int p){

 int j=10;

 int n=0;
 REAL sum = 1;
 REAL partsum=0;
```

```
 REAL term;
 REAL factor=1;
 while (true) {
   n++;
   factor=factor*n/((2*n-1)*2);
   if ( n <15000 ) {
     term=factor/((2*n+1)*(2*n+1));
   } else {
     term=factor/(2*n+1)/(2*n+1);
   }
   partsum += term;
   if (n==j){
        sum += partsum;
        partsum=0;
        j=int(j*1.3);
        cerr << n <<":";
   }
   if (bound(term,p)) break;
 }
 sum += partsum;
 return sum;
}

REAL catalanBB(int p){

 int j=10;

 int n=0;
 REAL sum = 0;
 REAL partsum=0;
 REAL term;
 while (true) {
        term=scale(
        3072/square(REAL(24*n+1)) -
        3072/square(REAL(24*n+2)) -
        23040/square(REAL(24*n+3)) +
        12288/square(REAL(24*n+4)) -
        768/square(REAL(24*n+5)) +
        9216/square(REAL(24*n+6)) +
        10368/square(REAL(24*n+8)) +
        2496/square(REAL(24*n+9)) -
        192/square(REAL(24*n+10)) +
        768/square(REAL(24*n+12)) -
        48/square(REAL(24*n+13)) +
        360/square(REAL(24*n+15)) +
        648/square(REAL(24*n+16)) +
        12/square(REAL(24*n+17)) +
        168/square(REAL(24*n+18)) +
        48/square(REAL(24*n+20)) -
        39/square(REAL(24*n+21))
        ,-n*12);
   n++;
   partsum += term;
   if (n==j){
        sum += partsum;
        partsum=0;
        j=int(j*1.3);
        cerr << n <<":";
```

```
   }
   if (bound(term,p)) break;
 }
 sum += partsum;
 return sum;
}

void compute(){

int N;

cout << "Problem C18: Catalan's  G=0.9159655941...\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

timecheck("Starting...");

REAL x =  pi()/8*log(2+sqrt(REAL(3)));
timecheck("...computation of pi,log ready...");
x=x+3*limit(catalan_approx)/8;

timecheck("...end of computation");

nij_print_a( x , N , 2);

};
```

Solution to Problem C19

```
#include "iRRAM.h"
#include "nij_print.h"

int static_m;
REAL Lint_approx(int p){
 INTEGER m=static_m;
 int n=1;
 int k=1;
 REAL sum = 0;
 REAL term =  1/REAL(m);
 INTEGER m_n3p1=m;
 while (true) {
   m_n3p1=( m^(3*(n-1)*(n-1)+3*(n-1)+1) );
   term=term/REAL(m_n3p1);
   sum = sum + term;
   if (bound(term,p)) break;
   n++;
 }
 return sum;
}

REAL Lint(int m){ static_m=m; return limit(Lint_approx);  };

void compute(){

int N;

cout << "Problem C19: Sparse Power Series...\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

timecheck("Start");
```

```
REAL x =  Lint(7);

timecheck("End of computation");

nij_print_a( x , N , 2);

};
```

Solution to Problem C20

```
#include "iRRAM.h"
#include "nij_print.h"

INTEGER d1,d2,n1,n2;
INTEGER d1_old,d2_old,n1_old,n2_old;

REAL F(){return sin(scale(pi()/REAL(17),1));}

REAL FAPPROX(int p,const REAL& dummy){
cerr<<".";

  REAL x=F();
  return (REAL(d1)+REAL(d2)*x)/(REAL(n1)+REAL(n2)*x);
}

REAL FIT(){return limit_lip(FAPPROX, 1,all_true,REAL(0));}

INTEGER inline floor(const REAL &x) {
continous_begin();
        INTEGER rnd = round(x);
        if (rnd > x) rnd= rnd - 1;
continous_end();
        return rnd;
}

REAL inline fraction(const REAL &x) {
        REAL rnd = x - round2(x);
        if (rnd < 0)
                return rnd + REAL(1);
        return rnd;
}

INTEGER inline a(const REAL &x, int k) {

        REAL one(1);
        REAL r = x;
        INTEGER aval = floor(x);

int steps=k/8+1;

d1=0;d2=1;
n2=0;n1=1;
        for (int i = 1; i <= k; i++) {
                d1_old=d1;
                d2_old=d2;
                d1=n1;
                d2=n2;
                n1=d1_old-aval*d1;
                n2=d2_old-aval*d2;
                r = one / (r-REAL(aval));
                aval = floor(r);

if (i%steps==0){cerr<<i; r=FIT();cerr<<"\n";}
```

```
        }
        return aval;
}

void compute(){

        int N;

        cout << "Problem C20: partial quotient\n";
        cout << "Please enter parameter N: ";
        cin  >> N;
        cout << "\n";

        REAL x=F();
        INTEGER result=a(x, N);

        cout  << "result is "<< result << "\n";

}
```

Solution to Problem C21

```
#include "iRRAM.h"
#include "nij_print.h"
#include "iRRAM_limit_templates.h"

REAL exp_cos_iterate(const int&dummy, const REAL&x) {
   REAL s=cos(x);
   REAL es=my_exp(s);
   REAL c=-sqrt(1-s*s);
   return x-(x-es)/(1-es*c);
};

void compute(){

int N,dummy;

cout << "Problem C21: solve x=exp(cos(x))\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

REAL y=1.302964001216;

REAL fix = generic_fixpoint2r<int>(exp_cos_iterate,y,-20,5,dummy);

nij_print_c( fix , N , 2);

}
```

Solution to Problem C22

```
#include "iRRAM.h"
#include "nij_print.h"

class newton_cotes{
        int n;
        INTEGER* a;
        INTEGER* b;
        public:
        newton_cotes(){n=-1;};
        ~newton_cotes(){if (n>=0){
                delete[] a;
```

```
                delete[] b;}};
        void init(int m);
        REAL operator()(int k){return REAL(a[k])/REAL(b[k]);};
};
newton_cotes nc;
void newton_cotes::init(int m){
        if (m==n) return;
        if (n>=0) { delete[] a; delete[] b;}
        n=m;
        a=new INTEGER[n+1];
        b=new INTEGER[n+1];
        INTEGER lagr[n+2];
        INTEGER s=0,t=0;
        lagr[1]=1;
        for (int i=1;i<=n;i++){
        for (int k=i+1;k>=1;k--){
                lagr[k]=lagr[k-1]-lagr[k]*i;
                }
        }
        INTEGER scm=1;
        for (int i=1;i<=n+1;i++){
                 if (scm/i*i!=scm) scm=scm*i;
        }
        INTEGER nnk0=scm;
        for (int i=1;i<=n;i++){
                nnk0=nnk0*n;
                lagr[i]=lagr[i]*nnk0;
        }
        INTEGER facn=1;
        for (int i=1;i<=n;i++){facn=facn*i;}
        if (n%2==0) {
                b[0] =  facn*scm;
        } else {
                b[0] = -facn*scm;
        }
        for (int j=1;j<=n;j++){
                b[j]=b[j-1]*j/(j-n-1);
        }
        for (int j=n; j>=0;j--){
                if (j<n/2) {
                        if (n%2==0)
                                a[j] =  a[n-j];
                        else
                                a[j] = -a[n-j];
                } else {
                        INTEGER sum=0;
                        INTEGER la;
                        for (int k=n; k>=0;k--){
                                if ( k == n) la=nnk0;else
                                        la= (la*j + lagr[k+1])/n;
                        sum=sum+la/(k+1);
                        }
                        a[j]=sum;
                }
```

```
        }
}

inline REAL f(const REAL& x){return sin(sin(sin(x)));}

void compute(){

int n,N;

cout << "Problem P22: integrate sin(sin(sin(x)))\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";
n=int(0.9*N) + 450;

timecheck("Determining Newton-Cotes formula");
nc.init(n);
timecheck("... ready, now evaluating function ...");

REAL sum=0;
for (int j=0; j<=n; j++){ sum=sum+nc(j)* f(REAL(j)/n); }

nij_print_a( sum , N , 1);

}
```

Solution to Problem C23

```
#include "iRRAM.h"
#include "nij_print.h"

void write  (const REALMATRIX& x){
  for (unsigned int i=0;i<rows(x);i++) {
    for (unsigned int j=0;j<columns(x);j++) {
      cout << x(i,j)<< " ";
      }
    cout << "\n";
    }
}

void compute(){
int N;

cout << "Problem P23: inversion of a Hankel matrix\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

  hint(int(N*N*0.4));
  REALMATRIX m,n,o;
  m=REALMATRIX(N,N);
  m(0,0) = REAL(1);

  m(0,0) = 1;
  m(0,1) = 1;

  for (int j=2;j<N;j++) {
          m(0,j) = m(0,j-1)+m(0,j-2);

  }

  for (int i=1;i<N;i++) {
          for (int j=N-2;j>=0;j--)
                  m(i,j) = m(i-1,j+1);
```

```
        m(i,N-1) = m(i,N-2) + m(i,N-3);
  }
  for (int i=0;i<N;i++) {
        for (int j=0;j<N;j++) {
              m(i,j) = 1/m(i,j);
        }
  }
  o=eye(N);
  n=o/m;

  nij_print_b( n(N-2,N-4) , 10 );

};
```

Solution to Problem C24

```
#include "iRRAM.h"
#include "nij_print.h"

void compute(){
int N;

cout << "Problem C24: inversion of the 1+Hankel matrix\n";
cout << "Please enter parameter N: ";
cin  >> N ;
cout << "\n";

hint(N/2);

  REALMATRIX m,n,o;
  m=REALMATRIX(N,N);

  m(0,0) = 1;
  m(0,1) = 1;

  for (int j=2;j<N;j++) {
        m(0,j) = m(0,j-1)+m(0,j-2);

  }

  for (int i=1;i<N;i++) {
        for (int j=N-2;j>=0;j--)
              m(i,j) = m(i-1,j+1);
        m(i,N-1) = m(i,N-2) + m(i,N-3);
  }
  for (int i=0;i<N;i++) {
        for (int j=0;j<N;j++) {
              m(i,j) = 1/m(i,j);
        }
  }
  for (int i=0;i<N;i++) {
        m(i,i) += 1;
  }

  o=eye(N);
  n=o/m;

  nij_print_b( n(N-2,N-1) , 10 );

};
```

A.6 Maple

The Maple solution consists of one Maple file per problem, which are reproduced in full as the solutions below. For instance the solution of Problem C01 is in the file `c01.mpl`, which is listed in subsection A.6. This file is excuted using

```
maple -c "N:=5" c01.mpl
```

The solutions all refer to the common file `common.mpl`, which contains

```
dig := 10^N:
l10 := evalf[10](log10(exp(1))):
pres := proc(loc, x) local xx,res;
        xx := frac(x);
        res := sprintf("%.*f",max(dig,length(op(1,xx))),xx)[3..dig+2];
        fprintf(loc, res);
end:
```

Solution to Problem C01

```
$include "common.mpl":
res := evalf[dig+2](sin(tan(cos(1)))):
pres("c01",res);
```

Solution to Problem C02

```
$include "common.mpl":
extra := ceil(evalf[10](exp(1)/Pi));
res := evalf[dig+extra](sqrt(exp(1)/Pi)):
pres("c02",res);
```

Solution to Problem C03

```
$include "common.mpl":
expr := (exp(1)+1)^3:
extra := ceil(evalf[10](log10(expr))):
res := evalf[dig+extra](sin(expr)):
pres("c03",res);
```

Solution to Problem C04

```
$include "common.mpl":
extra := ceil(evalf[10](Pi*sqrt(2011)*log10(exp(1)))):
res := evalf[dig+extra+3](exp(Pi*sqrt(2011))):
pres("c04",res);
```

Solution to Problem C05

```
$include "common.mpl":
expr := exp(sqrt(exp(1))):
extra := ceil(evalf[10](expr*l10)):
res := evalf[dig+extra+1](exp(expr)):
pres("c05",res);
```

Solution to Problem C06

```
$include "common.mpl":
kernelopts(gcfreq=3*10^7):
res := evalf[dig+2](arctanh(1-arctanh(1-arctanh(1-arctanh(1/Pi))))):
pres("c06",res);
```

Solution to Problem C07

```
$include "common.mpl":
expr := 1000:
extra := ceil(evalf[10](expr*log10(Pi))):
res := evalf[dig+extra+3](Pi^expr):
pres("c07",res);
```

Solution to Problem C08

```
$include "common.mpl":
kernelopts(gcfreq=10^8):
expr := (6^6):
extra := ceil(evalf[10](expr*log10(6)));
extra := extra + ceil(log10(extra));
e2 := 6^expr:
p := round(evalf[dig+extra](e2/2/Pi)):
pp := evalf[dig+extra+2](e2 - 2*Pi*p):
res := evalf[dig+2](sin(pp)):
pres("c08",res);
```

Solution to Problem C09

```
$include "common.mpl":
i1 := Pi*sqrt(2011)/3:
extra := ceil(evalf[10](i1)):
res1 := evalf[dig + extra + 4](tanh(i1)):
res2 := evalf[dig + extra + 4](10*arctan(res1)):
res := evalf[dig+ 2*extra ](sin(res2)):
pres("c09",res);
```

Solution to Problem C10

```
$include "common.mpl":
kernelopts(gcfreq=5*10^7):
e1 := 7 + 2^(1/5) - 5*8^(1/5):
e2 := e1^(1/3) + 4^(1/5) - 2^(1/5):
res := evalf[dig+2](e2):
pres("c10",res);
```

Solution to Problem C11

```
$include "common.mpl":
res := evalf[dig+4](tan(sqrt(2)) + arctanh(sin(1))):
pres("c11",res);
```

Solution to Problem C12

```
$include "common.mpl":
res := evalf[dig+3](arcsin(1/exp(2)) + arcsinh(exp(2))):
pres("c12",res);
```

Solution to Problem C13

```
$include "common.mpl":
kernelopts(gcfreq=10^7):
logi := proc(n)  local x,i,d;
    x[0] := 0.5:
        d := Digits +200*N^2;
        for i from 0 to n-1 do
            x[1] := evalf[d](3.999*x[0]*(1-x[0]));
                x[0] := x[1];
        od;
        x[1];
end:

res := evalf[dig](logi(10^N)):
pres("c13",res);
```

Solution to Problem C14

```
$include "common.mpl":
kernelopts(gcfreq=3*10^7):
rec2 := proc(n)  local a,i;
    a[0] := 14/3;
        a[1] := 5;
        a[2] := 184/35;
        for i from 1 to n-2 do
            a[3] :=
                    (114 - (1463 - (6390 - 9000/a[0])/a[1])/a[2]);
                (a[0],a[1],a[2]) := (a[1],a[2], a[3]);
        od;
        a[3];
end:

res := (rec2(10^(N+2))):
res := evalf[ max(length(numer(res)), length(denom(res))) ](res):
pres("c14",res);
```

Solution to Problem C15

```
$include "common.mpl":
kernelopts(gcfreq=3*10^7):
h := unapply(sum(1/k,k=n..n^2),n):
res := evalf[dig](h(10^(N+1))):
pres("c15",res);
```

Solution to Problem C17

```
$include "common.mpl":
kernelopts(gcfreq=3*10^7):
S := -4*Zeta(2)-2*Zeta(3) + 4*Zeta(2)*Zeta(3) + 2*Zeta(5);
res := evalf[dig+5](S):
```

```
pres("c17",res);
```

Solution to Problem C18

```
$include "common.mpl":
kernelopts(gcfreq=3*10^7):
res := evalf[dig](Catalan):
pres("c18",res);
```

Solution to Problem C21

```
dig := 1010:
l10 := evalf[10](log10(exp(1))):
pres := proc(loc, x) local xx,res;
        xx := frac(x);
        res := sprintf("%.*f",max(dig,length(op(1,xx))),xx)[3..dig+2];
        fprintf(loc, res);
end:
kernelopts(gcfreq=3*10^7):
res := evalf[dig](fsolve(exp(cos(x))=x,x)):
pres("c21",res);
```

Solution to Problem C22

```
$include "common.mpl":
kernelopts(gcfreq=3*10^7):
res := evalf[dig](Int(sin(sin(sin(x))), x=0..1));
pres("c22",res);
```

A.7 MPFR

The solutions using MPFR are C programs. These programs are reproduced in full as the solutions below. For example the solution to Problem C01 is a program `c01.c` that is listed in subsection A.7. It has to be compiled with the command

```
gcc -I/home/mpfr/gmp-4.1.4/include -I/home/mpfr/mpfr-20050920/include               \
  -I/home/mpfr/mpfi-1.3.3-patched/include -O2 -g -W -Wall -pedantic -ansi c01.c     \
  /home/mpfr/mpfi-1.3.3-patched/lib/libmpfi.a /home/mpfr/mpfr-20050920/lib/libmpfr.a \
  /home/mpfr/gmp-4.1.4/lib/libgmp.a -lm -o c01
```

and then run with the command

```
./c01 100000
```

which will produce output like

```
Using GMP 4.1.4 and MPFR 2.3.0
Setting precision to 332229
many many digits
Cputime: 17798ms (output 64ms)
```

The many digits are then written to standard output while the other three lines are written to standard error.

Solution to Problem C01

```
/* solution to problem C01 from the "many digits friendly competition":
   compute the first 10^N decimal digits after the decimal point of
   sin(tan(cos(1))). */

#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (j, 32);
      mpfr_set_d (j, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (j, j, M, GMP_RNDU);
      mpfr_add_ui (j, j, 2, GMP_RNDU);
      p = mpfr_get_ui (j, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);
      mpfr_set_prec (j, 2);
      mpfr_set_ui (j, 1, GMP_RNDN);
      mpfr_cos (i, j, GMP_RNDN);
      mpfr_tan (i, i, GMP_RNDN);
      mpfr_sin (i, i, GMP_RNDN);
      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N, M) == 0);

  lo[N] = '\0';

  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C02

```
/* solution to problem C02 from the "many digits friendly competition":
```

```
   compute the first 10^N decimal digits after the decimal point of
   sqrt(e/pi). */

#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 3, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (j, 2);
      mpfr_set_prec (i, p);
      mpfr_set_ui (j, 1, GMP_RNDN);
      mpfr_exp (i, j, GMP_RNDN); /* i = exp(1) */
      mpfr_set_prec (j, p);
      mpfr_const_pi (j, GMP_RNDN);
      mpfr_div (i, i, j, GMP_RNDN);
      mpfr_sqrt (i, i, GMP_RNDN);

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N, M) == 0);

  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C03

```
/* solution to problem C03 from the "many digits friendly competition":
   compute the first 10^N decimal digits after the decimal point of
```

```
   sin((e+1)^3).

   We compute with precision p and rounding to nearest:
   x = o(e)
   y = o(x+1)
   z = o(y^2)
   t = o(z*y)
   u = o(sin(t))

   We have with theta a generic value such that |theta| <= 2^(-p):
   x = e*(1+theta)
   y = (x+1)*(1+theta) = (e*(1+theta)+1)*(1+theta) = (e+1)*(1+theta)^2
   z = y^2*(1+theta) = (e+1)^2*(1+theta)^5
   t = z*y*(1+theta) = (e+1)^3*(1+theta)^8

   p=3;
   x = [approx(exp(1),p,3), approx(exp(1),p,2)]
   y = [approx(x[1]+1,p,3), approx(x[2]+1,p,2)]
   z = [approx(y[1]^2,p,3), approx(y[2]^2,p,2)]
   t = [approx(z[1]*y[1],p,3), approx(z[2]*y[2],p,2)]

   For p >= 3, we have 40 <= t <= 64, and (1+theta)^8 can be written
   1+13*theta when |theta| <= 2^(-p), so the absolute error on t is
   bounded by 13*2^(6-p).

   Thus t = (e+1)^3 + r with |r| <= 13*2^(6-p).
   sin(t) = sin((e+1)^3) + r * cos(a) thus the final error, taking into
   account the rounding error on u, is bounded by 1/2*ulp*(u) + 13*2^(6-p)
   <= 2^(-p-1) + 13*2^(6-p) <= 2^(10-p).

   Thus for 2^(10-p) <= 1/2*10^(-M), i.e. p >= 11 + M*log(10)/log(2),
   the error on u is bounded by 1/2ulp of the output value.
*/

#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long M, N = atoi (argv[1]);
  mp_prec_t p;
  mpfr_t i, j;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 11, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
```

```
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);
      mpfr_set_prec (j, 2);

      mpfr_set_ui (j, 1, GMP_RNDN);
      mpfr_exp (i, j, GMP_RNDN); /* i = e */
      mpfr_add_ui (i, i, 1, GMP_RNDN); /* e+1 */
      mpfr_set_prec (j, p);
      mpfr_mul (j, i, i, GMP_RNDN); /* (e+1)^2 */
      mpfr_mul (j, j, i, GMP_RNDN); /* (e+1)^3 */

      st0 = cputime ();
      mpfr_sin (i, j, GMP_RNDN);
      st0 = cputime () - st0;
      fprintf (stderr, "mpfr_sin took %dms\n", st0);

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N, M) == 0);

  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C04

```
#define USE_MPFI
#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]);
  mp_prec_t p;
  mpfi_t i, j;
  char *lo, *hi;
  mp_exp_t exp_lo, exp_hi;
  int st;
  mpz_t intpart;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  /* initial estimate of the working precision */
  p = (mp_prec_t) (3.321928095 * (double) N + 10.0 + 61.0);
  /* note: 60.0 is not enough for N=100000 */
  fprintf (stderr, "Setting initial precision to %lu\n", p);

  mpfi_init (i);
  mpfi_init (j);
  mpz_init (intpart);

  do
```

```
    {
      mpfi_set_prec (i, p);
      mpfi_set_prec (j, 12); /* enough to store 2011 exactly */

      mpfi_set_ui (j, 2011);
      mpfi_sqrt (i, j);

      mpfi_set_prec (j, p);
      mpfi_const_pi (j);
      mpfi_mul (i, i, j);
      mpfi_exp (i, i);
      mpz_set_str (intpart,
           "152873250308385307260607189566326264602881352504957123549915 35",
           10);
      mpfi_sub_z (i, i, intpart);

      lo = mpfr_get_str (NULL, &exp_lo, 10, N, LEFT(i), GMP_RNDZ);
      hi = mpfr_get_str (NULL, &exp_hi, 10, N, RIGHT(i), GMP_RNDZ);
      if (exp_lo == exp_hi && strcmp (lo, hi) == 0)
        break;
      else
        {
          p += p / 100;
          fprintf (stderr, "Increasing precision to %lu\n", p);
        }
    }
  while (1);

  printf ("0%s\n", lo);

  mpfi_clear (i);
  mpfi_clear (j);
  mpz_clear (intpart);

  fprintf (stderr, "Cputime: %dms\n", cputime () - st);
  return 0;
}
```

Solution to Problem C05

```
#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);

  M = N;

  do
    {
```

```
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 25, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);
      mpfr_set_prec (j, 2);

      mpfr_set_d (j, 0.5, GMP_RNDN);
      mpfr_exp (i, j, GMP_RNDN);
      mpfr_exp (i, i, GMP_RNDN);
      mpfr_exp (i, i, GMP_RNDN);

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M + 3, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo + 3, N, M) == 0);

  lo += 3; /* skip 181 */
  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C06

```
/* solution to problem C06 from the "many digits friendly competition":
   compute the first 10^N decimal digits after the decimal point of
   arctanh(1-arctanh(1-arctanh(1-arctanh(1/Pi)))) ~ 1.123.

   We compute with precision p and rounding to nearest:

   q = o(Pi)
   r = o(1/q)
   s = o(arctahnh(r))
   t = o(1-s)
   u = o(arctahnh(t))
   v = o(1-u)
   w = o(arctahnh(v))
   x = o(1-w)
   y = o(arctahnh(x)).

   We can check that for p >= 5, we have 2 <= q < 4, 1/4 <= r, s < 1/2,
   1/2 <= t, u, x < 1 [t <= 11/16, x <= 27/32], 1/8 <= v, w < 1/4, 1 <= y < 2.

   p = 6;
   q = [approx(Pi,p,3),approx(Pi,p,2)]
   r = [approx(1/q[2],p,3),approx(1/q[1],p,2)]
   s = [approx(atanh(r[1]),p,3),approx(atanh(r[2]),p,2)]
   t = [approx(1-s[2],p,3),approx(1-s[1],p,2)]
   u = [approx(atanh(t[1]),p,3),approx(atanh(t[2]),p,2)]
   v = [approx(1-u[2],p,3),approx(1-u[1],p,2)]
```

```
   w = [approx(atanh(v[1]),p,3),approx(atanh(v[2]),p,2)]
   x = [approx(1-w[2],p,3),approx(1-w[1],p,2)]
   y = [approx(atanh(x[1]),p,3),approx(atanh(x[2]),p,2)]

   We have q = Pi * (1+theta) with |theta| <= 2^(-p),
   thus r = 1/Pi * (1+theta)^2, which can be written 1/Pi * (1+3*theta),
   thus the absolute error on r is at most 2^(-p).

   The error on s is at most 1/2*ulp(s) + err(r) / (1-alpha^2)
   <= 2^(-p-2) + 2^(-p) / (1-(1/2)^2) <= 2 * 2^(-p).

   The error on t is at most 1/2*ulp(t) + err(s) <= 2^(-p-1) + 2 * 2^(-p)
   <= 3 * 2^(-p).

   The error on u is at most 1/2*ulp(u) + err(t) / (1-alpha^2)
   <= 2^(-p-1) + 3 * 2^(-p) / (1-(11/16)^2) <= 7 * 2^(-p).

   The error on v is at most 1/2*ulp(v) + err(u) <= 2^(-p-3) + 7 * 2^(-p)
   <= 8 * 2^(-p).

   The error on w is at most 1/2*ulp(w) + err(v) / (1-alpha^2)
   <= 2^(-p-3) + 8 * 2^(-p) / (1 - (1/4)^2) <= 9 * 2^(-p).

   The error on x is at most 1/2*ulp(x) + err(w) <= 10 * 2^(-p).

   Finally, the error on y is at most 1/2*ulp(y) + err(x) / (1-alpha^2)
   <= 2^(-p) + 10 * 2^(-p) / (1 - (27/32)^2) <= 36 * 2^(-p) <= 2^(6-p).
*/

#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);

  M = N + 1;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 6, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);

      mpfr_const_pi (i, GMP_RNDN);
      mpfr_ui_div (i, 1, i, GMP_RNDN);
      mpfr_atanh (i, i, GMP_RNDN);
      mpfr_ui_sub (i, 1, i, GMP_RNDN);
```

```
      mpfr_atanh (i, i, GMP_RNDN);
      mpfr_ui_sub (i, 1, i, GMP_RNDN);
      mpfr_atanh (i, i, GMP_RNDN);
      mpfr_ui_sub (i, 1, i, GMP_RNDN);
      mpfr_atanh (i, i, GMP_RNDN);

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N + 1, M) == 0);

  lo ++;
  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C07

```
#include "many.h"

/* Factor > 1 between real value and computed value: < (1 + 2(^-p))^r */

static void
compval (mpfr_t x, mp_prec_t p)
{
  mpfr_t y;

  mpfr_set_prec (x, p);
  mpfr_init2 (y, p);
  mpfr_const_pi (x, GMP_RNDN); /* r = 1 */

  mpfr_sqr (y, x, GMP_RNDN); /* r = 3 */
  mpfr_sqr (y, y, GMP_RNDN); /* r = 7 */
  mpfr_mul (x, x, y, GMP_RNDN); /* r = 9 */

  mpfr_sqr (y, x, GMP_RNDN); /* r = 19 */
  mpfr_sqr (y, y, GMP_RNDN); /* r = 39 */
  mpfr_mul (x, x, y, GMP_RNDN); /* r = 49 */

  mpfr_sqr (y, x, GMP_RNDN); /* r = 99 */
  mpfr_sqr (y, y, GMP_RNDN); /* r = 199 */
  mpfr_mul (x, x, y, GMP_RNDN); /* r = 249 */

  mpfr_sqr (x, x, GMP_RNDN); /* r = 499 */
  mpfr_sqr (x, x, GMP_RNDN); /* r = 999 */
  mpfr_sqr (x, x, GMP_RNDN); /* r = 1999 -> 11 guard bits */

  mpfr_clear (y);
}

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j;
  char *lo;
```

```
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 1652 + 11, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);
      compval (i, p);
      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M + 498, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo + 498, N, M) == 0);

  lo += 498;
  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C08

```
#include "many.h"

/*  C08   -   Sin(6^6^6)

    c08a  -   Jim White -  base version adapted direct from p08a
                           magnitude is too huge for mpfr_sin to deal
                           with, and trisections/bisections are simply
                           too numerous, so we scale the value down
                           to  be < 2Pi and pass that remainder on
                           to mpfr_sin
 */

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j, k, m, pi;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
```

```
  st = cputime ();

  mpfr_init (i);  /* target result precision */
  mpfr_init (pi); /* pi to target precision */
  mpfr_init (j);  /* argument 6^(6^6), integer requiring 120640 bits (3769) */
  mpfr_init (k);  /* to get accurate remainder of j / 2PI we need a register of width prec(i) + prec(j) ? *
  mpfr_init (m);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);
      mpfr_set_prec (j, 120640);    /*  6^6^6  fits in this  */
      mpfr_set_prec (m, 120640);
      mpfr_set_prec (k,  p + 120640);
      mpfr_set_prec (pi, p + 120640);

      mpfr_const_pi (pi, GMP_RNDN);
      mpfr_mul_ui (pi, pi, 2, GMP_RNDN);         /*  pi = 2PI      */
      mpfr_ui_pow_ui (j, 6, 46656, GMP_RNDN);    /*  j  = 6^(6^6)  */

      mpfr_div   (k, j, pi, GMP_RNDN);           /*  k  = j / pi   */
      mpfr_trunc (m, k);                         /*  m  = floor(k) */
      mpfr_mul (k, pi, m, GMP_RNDN);             /*  k  = m * pi   */
      mpfr_sub (k, j, k, GMP_RNDN);              /*  k  = j - k    */
      mpfr_prec_round (k, p, GMP_RNDN);          /*  reduce it to target precision */

      mpfr_sin (i, k, GMP_RNDN);

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo + 1, N, M) == 0);

  /* lo ++;  this result is positive */
  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C09

```
/* solution to problem C09 from the "many digits friendly competition":
   compute the first 10^N decimal digits after the decimal point of
   sin(10*arctan(tanh(Pi*sqrt(2011)/3))) ~ 0.999.

   We compute with precision p and rounding to nearest:
```

```
   q = o(Pi)
   r = o(sqrt(2011))
   s = o(q*r)
   t = o(s/3)
   u = o(tanh(t))
   v = o(arctan(u))
   w = o(10*v)
   x = o(sin(w)).

   For p >= 6, we have 2 <= q < 4, 32 <= r, t < 64, 128 <= s < 256,
   1/2 <= u, v < 1, 4 <= w < 8.
   More precisely, 45 <= t <= 48, 63/64 <= u <= 1, 61/8 <= w <= 8,
   and sin(61/8) ~ 0.973 <= x <= 1.

   p=6;
   q = [approx(Pi,p,3),approx(Pi,p,2)]
   r = [approx(sqrt(2011),p,3),approx(sqrt(2011),p,2)]
   s = [approx(q[1]*r[1],p,3),approx(q[2]*r[2],p,2)]
   t = [approx(s[1]/3,p,3),approx(s[2]/3,p,2)]
   u = [approx(tanh(t[1]),p,3),approx(tanh(t[1]),p,2)]
   v = [approx(atan(u[1]),p,3),approx(atan(u[2]),p,2)]
   w = [approx(10*v[1],p,3),approx(10*v[2],p,2)]

   We have q = Pi * (1+theta) with |theta| <= 2^(-p),
   r = sqrt(2011) * (1+theta)
   s = Pi*sqrt(2011) * (1+theta)^3
   t = Pi*sqrt(2011)/3 * (1+theta)^4

   Since (1+theta)^4 can be written 1+5*theta, the absolute error on t
   is bounded by Pi*sqrt(2011)/3 * (5*theta) <= 235 * 2^(-p).
   Thus err(u) <= 1/2*ulp(u) + err(t) * (1-tanh(alpha)^2)
               <= 2^(-p-1) + 235 * 2^(-p) * (1-tanh(45)^2)
               <= 2^(-p-1) + 2^(-119-p)
               <= 2^(-p).

   err(v) <= 1/2*ulp(v) + err(u) / (1 + alpha^2)
          <= 2^(-p-1) + 2^(-p) / (1 + (63/64)^2)
          <= 2 * 2^(-p).

   err(w) <= 1/2*ulp(w) + 10*err(v)
          <= 2^(2-p) + 20*2^(-p)
          <= 24 * 2^(-p).

   Finally, err(x) <= 1/2*ulp(x) + err(w) * cos(alpha)
                   <= 2^(-p-1) + 24 * 2^(-p)
                   <= 2^(5-p).
*/

#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j, k;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0 = 0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
```

```
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);
  mpfr_init (k);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 6, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      if (p < 11)
        p = 11; /* so that 2011 is exact */
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);
      mpfr_set_prec (j, p);

      mpfr_const_pi (i, GMP_RNDN);
      mpfr_set_ui (j, 2011, GMP_RNDN);
      mpfr_sqrt (j, j, GMP_RNDN);
      mpfr_mul (i, i, j, GMP_RNDN);
      mpfr_div_ui (i, i, 3, GMP_RNDN);
      mpfr_tanh (i, i, GMP_RNDN);
      mpfr_atan (i, i, GMP_RNDN);
      mpfr_mul_ui (i, i, 10, GMP_RNDN);
      mpfr_sin (i, i, GMP_RNDN);

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N, M) == 0);

  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);
  mpfr_clear (k);

 end:
  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C10

```
/* solution to problem C10 from the "many digits friendly competition":
   compute the first 10^N decimal digits after the decimal point of
   (7+2^(1/5)-5*8^(1/5))^(1/3) + 4^(1/5) - 2^(1/5).

   With x = 2^(1/5), this expression is
   (7 + x - 5*x^3)^(1/3) + x^2 - x.

   We compute with precision p and rounding to nearest:
```

```
   x = o(2^(1/5))
   y = o(x^2)
   z = o(y*x)
   q = o(7+x)
   r = o(5*z)
   s = o(q-r)
   t = o(s^(1/3))
   u = o(t+y)
   v = o(u-x).

   p=10;
   x = [approx(2^(1/5),p,3),approx(2^(1/5),p,2)]
   y = [approx(x[1]^2,p,3),approx(x[2]^2,p,2)]
   z = [approx(y[1]*x[1],p,3),approx(y[2]*x[2],p,2)]
   q = [approx(7+x[1],p,3),approx(7+x[2],p,2)]
   r = [approx(5*z[1],p,3),approx(5*z[2],p,2)]
   s = [approx(q[1]-r[2],p,3),approx(q[2]-r[1],p,2)]
   t = [approx(s[1]^(1/3),p,3),approx(s[2]^(1/3),p,2)]
   u = [approx(t[1]+y[1],p,3),approx(t[2]+y[2],p,2)]
   v = [approx(u[1]-x[2],p,3),approx(u[2]-x[1],p,2)]

   For p >= 10, we have 1 <= x, y, z < 2, 8 <= q < 16,
   4 <= r < 8, 1/2 <= s, t < 1, 2 <= u < 4, 1/2 <= v <= 2.
   More precisely, 67/128 <= s <= 19/32.

   We have x = 2^(1/5) * (1+theta), and the absolute error on x can be
   bounded by 2^(1/5) * 2^(-p) <= 2^(1-p).

           y = 2^(2/5) * (1+theta)^3
           err(y) <= 2^(2/5) * (4*theta) <= 6 * 2^(-p).
           z = 2^(3/5) * (1+theta)^5

   err(q) <= 1/2*ulp(q) + err(x) <= 2^(3-p) + 2^(1-p) <= 10 * 2^(-p).

   r = 5*2^(3/5) * (1+theta)^6
   Since (1+theta)^6 can be written 1+7*theta, the absolute error on r
   is bounded by 5*2^(3/5)*7*theta <= 54 * 2^(-p).

   err(s) <= 1/2*ulp(s) + err(q) + err(r) <=
   2^(-p-1) + 10 * 2^(-p) + 54 * 2^(-p) <= 65 * 2^(-p).

   err(t) <= 1/2*ulp(t) + err(s) * (1/3*alpha^(-2/3))
          <= 2^(-p-1) + 65 * 2^(-p) * (1/3*(67/128)^(-2/3))
          <= 34 * 2^(-p).

   err(u) <= 1/2*ulp(u) + err(t) + err(y)
          <= 2^(1-p) + 34 * 2^(-p) + 6 * 2^(-p) <= 42 * 2^(-p).

   err(v) <= 1/2*ulp(v) + err(u) + err(x)
          <= 2^(-p) + 42 * 2^(-p) + 2^(1-p)
          <= 45 * 2^(-p) <= 2^(6-p).
*/

#define WANT_MPFR_ROOT
#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]);
  mp_prec_t p;
  mpfr_t i, j, k, l;
```

```
  char *lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init2 (i, 32);

  mpfr_set_d (i, LOG2_10, GMP_RNDU);
  mpfr_mul_ui (i, i, N, GMP_RNDU);
  mpfr_add_ui (i, i, 7, GMP_RNDU);
  p = mpfr_get_ui (i, GMP_RNDU);
  if (p < 10)
    p = 10;
  fprintf (stderr, "Setting precision to %lu\n", p);

  mpfr_set_prec (i, p);
  mpfr_init2 (j, p);
  mpfr_init2 (k, p);
  mpfr_init2 (l, p);

  mpfr_set_ui (i, 2, GMP_RNDN);
  mpfr_root (i, i, 5, GMP_RNDN); /* x = 2^(1/5) */
  mpfr_mul (j, i, i, GMP_RNDN); /* y */
  mpfr_mul (k, j, i, GMP_RNDN); /* z */
  mpfr_add_ui (l, i, 7, GMP_RNDN); /* q */
  mpfr_mul_ui (k, k, 5, GMP_RNDN); /* r */
  mpfr_sub (l, l, k, GMP_RNDN); /* s */
  mpfr_root (l, l, 3, GMP_RNDN); /* t */
  mpfr_add (l, l, j, GMP_RNDN); /* u */
  mpfr_sub (i, l, i, GMP_RNDN); /* v */

  st0 = cputime ();
  lo = fixed_point_output (i, N);
  st0 = cputime () - st0;

  printf ("%s\n", lo);
  free (lo);

  mpfr_clear (i);
  mpfr_clear (j);
  mpfr_clear (k);
  mpfr_clear (l);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C11

```
#include "many.h"

static void
compval (mpfr_t x, mp_prec_t p)
{
  mpfr_t y, i;

  mpfr_set_prec (x, p);
  mpfr_init2 (y, p);
  mpfr_init2 (i, 2);

  mpfr_set_ui (i, 2, GMP_RNDN);
```

```
  mpfr_sqrt (x, i, GMP_RNDN); /* err < 2^(-p) */
  mpfr_tan (x, x, GMP_RNDN); /* err < (42 + 4) * 2^(-p) = 46 * 2^(-p) */
  mpfr_set_ui (i, 1, GMP_RNDN);
  mpfr_sin (y, i, GMP_RNDN); /* err < 1/2 * 2^(-p) */
  mpfr_atanh (y, y, GMP_RNDN); /* err < (2 + 1) * 2^(-p) = 3 * 2^(-p) */
  mpfr_add (x, x, y, GMP_RNDN); /* err < 53 * 2^(-p) -> 6 guard bits */

  mpfr_clear (y);
  mpfr_clear (i);
}

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 6, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);
      compval (i, p);
      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M + 1, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo + 1, N, M) == 0);

  lo++;
  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C12

```
#include "many.h"

static void
compval (mpfr_t x, mp_prec_t p)
```

```
{
  mpfr_t y, i;

  mpfr_set_prec (x, p);
  mpfr_init2 (y, p);
  mpfr_init2 (i, 2);

  mpfr_set_si (i, -2, GMP_RNDN);
  mpfr_exp (y, i, GMP_RNDN); /* err < 1/8 * 2^(-p) */
  mpfr_sqr (x, y, GMP_RNDN); /* err < 1/20 * 2^(-p) */
  mpfr_add_ui (x, x, 1, GMP_RNDN); /* err < 21/20 * 2^(-p) */
  mpfr_sqrt (x, x, GMP_RNDN); /* err < 61/40 * 2^(-p) */
  mpfr_log1p (x, x, GMP_RNDN); /* err < (61/80 + 1/2) * 2^(-p) */
  mpfr_asin (y, y, GMP_RNDN); /* err < 1/4 * 2^(-p) */
  mpfr_add (x, x, y, GMP_RNDN); /* err < 121/80 * 2^(-p) -> 2 guard bits */

  mpfr_clear (y);
  mpfr_clear (i);
}

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 2, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);
      compval (i, p);
      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N, M) == 0);

  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C13

```
#include "many.h"

#define LOG2_39P99 5.3215674 /* upper approximation to log(39.99)/log(2) */

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j;
  char *lo;
  mp_exp_t exp_lo;
  int k;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_prec (j, 32);
      mpfr_set_d (i, LOG2_39P99, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 3, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);
      mpfr_set_prec (j, p);

      mpfr_set_d (i, 0.5, GMP_RNDN);
      for (k = 0; k < (int) N; k++)
        {
          mpfr_ui_sub (j, 1, i, GMP_RNDN);
          mpfr_mul (i, i, j, GMP_RNDN);
          mpfr_mul_ui (i, i, 3999, GMP_RNDN);
          mpfr_div_ui (i, i, 1000, GMP_RNDN);
        }

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N, M) == 0);

  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C14

```
#include "many.h"

#define CONST 438.28595410261431 /* upper approximation of 100*log(a)/log(2)
                                    where a = 10+sqrt(118). */

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t h, i, j;
  mpfr_ptr t[4];
  char *lo;
  mp_exp_t exp_lo;
  int k;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (h);
  mpfr_init (i);
  mpfr_init (j);

  M = 8 * N;

  do
    {
      M += 10;
      p = atoi (argv[2]);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (h, p);
      mpfr_set_prec (i, p);
      mpfr_set_prec (j, p);

      t[0] = h;
      mpfr_set_ui (h, 14, GMP_RNDN);
      mpfr_div_ui (h, h, 3, GMP_RNDN); /* a0 = 14/3 */

      t[1] = i;
      mpfr_set_ui (i, 5, GMP_RNDN); /* a1 = 5 */

      t[2] = j;
      mpfr_set_ui (j, 184, GMP_RNDN);
      mpfr_div_ui (j, j, 35, GMP_RNDN); /* a1 = 184/35 */

      for (k = 0; k < 100 * (int) N; k++)
        {
          mpfr_ui_div (t[0], 9000, t[0], GMP_RNDN);
          mpfr_ui_sub (t[0], 6390, t[0], GMP_RNDN);
          mpfr_div (t[0], t[0], t[1], GMP_RNDN);
          mpfr_ui_sub (t[0], 1463, t[0], GMP_RNDN);
          mpfr_div (t[0], t[0], t[2], GMP_RNDN);
          mpfr_ui_sub (t[0], 114, t[0], GMP_RNDN);
          t[3] = t[0];
          t[0] = t[1];
          t[1] = t[2];
          t[2] = t[3];
        }
```

```
      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M + 1, t[2], GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo + 1, N, M) == 0);

  lo += 3; /* to skip the initial 5 */
  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C15

```
/* solution to problem C15 from the "many digits friendly competition":
   compute the first 10^N decimal digits after the decimal point of
   h(10^(N+1)) where
   h(n) = sum(1/k, k=n..n^2).

   According to (6.3.2) in Abramowitz & Stegun, we have:

   Psi(n) = -gamma + sum(1/k, k=1..n-1)

   thus h(n) = Psi(n^2+1) - Psi(n) and we are reduced to the problem of
   computing n/10 digits of Psi(n^2+1) - Psi(n).

   Formula (6.3.18) from Abramowitz & Stegun gives:

   Psi(z) ~ log(z) - 1/(2z) - sum(B[2k]/(2kz^(2k)), k=1..infinity).

   Since we have |B[2k]| <= 2*(2k)!/(2Pi)^(2k)/(1-2^(1-2k)) (23.1.15),
   we deduce |B[2k]/(2kz^(2k))| <= 2/k (2k)!/(2Piz)^(2k) for k >= 1

   Using n! <= (n/e)^n*sqrt(8*n) which is true for n >= 1, we get

   |B[2k]/(2kz^(2k))| <= 8 * (k/(ePiz))^(2k).

   The minimum is of (k/(ePiz))^(2k) is obtained for k ~ Piz,
   with a value of e^(-2k) ~ e^(-2Piz). Thus to get a sufficient
   accuracy, we need that z is large enough.

   Here we have:

   (a) for Psi(n): z=n and we want n/10 digits, i.e. an error of 10^(-n/10).
       We have e^(-2Piz) ~ 2^(-9.06n) and 10^(-n/10) ~ 2^(-0.33n), thus ok.
   (b) for Psi(n^2+1): z=n^2+1 and we still want n/10 digits. This is ok too.

   Let R(z) = sum(B[2k]/(2kz^(2k)), k=1..infinity).

   We have Psi(n^2+1) - Psi(n) ~ log((n^2+1)/n) - 1/(2(n^2+1)) + 1/(2n)
                                + R(n) - R(n^2+1).
*/

#include "many.h"

#define USE_H

#ifdef USE_H
/* return 1/(a+1) + ... + 1/b into p/q */
void
```

```
h (mpz_t P, mpz_t Q, unsigned long a, unsigned long b, mp_prec_t p,
   mp_rnd_t rnd)
{
  if (a + 1 == b)
    {
      mpz_set_ui (P, 1);
      mpz_set_ui (Q, b);
    }
  else
    {
      unsigned long c = (a + b) / 2 ;
      mpz_t P2, Q2;
      size_t lq, lp, l;
      h (P, Q, a, c, p, rnd);
      mpz_init (P2);
      mpz_init (Q2);
      h (P2, Q2, c, b, p, rnd);
      /* P/Q + P2/Q2 = (P*Q2+Q*P2)/(Q*Q2) */
      mpz_mul (P, P, Q2);
      mpz_mul (P2, P2, Q);
      mpz_add (P, P, P2);
      mpz_mul (Q, Q, Q2);
      if ((lq = mpz_sizeinbase (Q, 2)) > p && (lp = mpz_sizeinbase (P, 2)) > p)
        {
          l = (lq > lp) ? lp : lq;
          l = l - p;
          if (rnd == GMP_RNDD)
            {
              mpz_tdiv_q_2exp (P, P, l);
              mpz_cdiv_q_2exp (Q, Q, l);
            }
          else /* GMP_RNDU */
            {
              mpz_cdiv_q_2exp (P, P, l);
              mpz_tdiv_q_2exp (Q, Q, l);
            }
        }
      mpz_clear (P2);
      mpz_clear (Q2);
    }
}
#endif

/* [Copied from mpfr/lngamma.c.]
   assuming b[0]...b[2(n-1)] are computed, computes and stores B[2n]*(2n+1)!

   t/(exp(t)-1) = sum(B[j]*t^j/j!, j=0..infinity)
   thus t = (exp(t)-1) * sum(B[j]*t^j/j!, n=0..infinity).
   Taking the coefficient of degree n+1 > 1, we get:
   0 = sum(1/(n+1-k)!*B[k]/k!, k=0..n)
   which gives:
   B[n] = -sum(binomial(n+1,k)*B[k], k=0..n-1)/(n+1).

   Let C[n] = B[n]*(n+1)!.
   Then C[n] = -sum(binomial(n+1,k)*C[k]*n!/(k+1)!,  k=0..n-1),
   which proves that the C[n] are integers.
*/
static mpz_t*
```

```
bernoulli (mpz_t *b, unsigned long n)
{
  if (n == 0)
    {
      b = (mpz_t *) (*__gmp_allocate_func) (sizeof (mpz_t));
      mpz_init_set_ui (b[0], 1);
    }
  else
    {
      mpz_t t;
      unsigned long k;

      b = (mpz_t *) (*__gmp_reallocate_func)
        (b, n * sizeof (mpz_t), (n + 1) * sizeof (mpz_t));
      mpz_init (b[n]);
      /* b[n] = -sum(binomial(2n+1,2k)*C[k]*(2n)!/(2k+1)!,  k=0..n-1) */
      mpz_init_set_ui (t, 2 * n + 1);
      mpz_mul_ui (t, t, 2 * n - 1);
      mpz_mul_ui (t, t, 2 * n);
      mpz_mul_ui (t, t, n);
      mpz_div_ui (t, t, 3); /* exact: t=binomial(2*n+1,2*k)*(2*n)!/(2*k+1)!
                               for k=n-1 */
      mpz_mul (b[n], t, b[n-1]);
      for (k = n - 1; k-- > 0;)
        {
          if (k < 128)
            mpz_mul_ui (t, t, (2 * k + 1) * (2 * k + 2) * (2 * k + 2) * (2 * k + 3));
          else if (k < 32767)
            {
              mpz_mul_ui (t, t, (2 * k + 1) * (2 * k + 3));
              mpz_mul_ui (t, t, (2 * k + 2) * (2 * k + 2));
            }
          else
            {
              mpz_mul_ui (t, t, 2 * k + 1);
              mpz_mul_ui (t, t, 2 * k + 2);
              mpz_mul_ui (t, t, 2 * k + 2);
              mpz_mul_ui (t, t, 2 * k + 3);
            }
          if (n - k < 32768)
            mpz_div_ui (t, t, (2 * (n - k) + 1) * 2 * (n - k));
          else
            {
              mpz_div_ui (t, t, 2 * (n - k) + 1);
              mpz_div_ui (t, t, 2 * (n - k));
            }
          mpz_addmul (b[n], t, b[k]);
        }
      /* take into account C[1] */
      mpz_mul_ui (t, t, 2 * n + 1);
      mpz_div_2exp (t, t, 1);
      mpz_sub (b[n], b[n], t);
      mpz_neg (b[n], b[n]);
      mpz_clear (t);
    }
  return b;
}
```

```
int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M, k, n;
  mp_prec_t p;
  mpfr_t i, j, t, s;
  char *lo;
  int st, st0, stb = 0;
  mp_exp_t e;
  mpz_t *B, tmp;
  unsigned long Bm = 0; /* number of allocated B[] */
  unsigned long oldBm;
  int overflow;
  mpz_t P, Q;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);
  mpfr_init (t);
  mpfr_init (s);
  mpz_init (tmp);
  mpz_init (P);
  mpz_init (Q);

  n = 10 * N;

  /* h(10*N) is about log(10*N).
     For N<=10^7, this is less than 8 digits. */
  M = N + 8;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 6, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);
      mpfr_set_prec (j, p);
      mpfr_set_prec (t, p);
      mpfr_set_prec (s, p);

      if (Bm == 0)
        {
          B = bernoulli ((mpz_t *) 0, 0);
          B = bernoulli (B, 1);
          Bm = 2;
        }

      /* first compute R(n) in s */
      st0 = cputime ();
#ifndef USE_H
      mpfr_set_ui (s, 0, GMP_RNDN); /* sum */
      mpfr_set_ui (j, 1, GMP_RNDN); /* 1/(2k+1)!/z^(2k) */
      for (k = 1; ; k++)
        {
```

```
          mpfr_div_ui (j, j, 2 * k, GMP_RNDN);
          mpfr_div_ui (j, j, 2 * k + 1, GMP_RNDN);
          mpfr_div_ui (j, j, n, GMP_RNDN);
          mpfr_div_ui (j, j, n, GMP_RNDN);
          /* compute C[2k] */
          if (Bm <= k)
            {
              stb -= cputime ();
              B = bernoulli (B, k); /* B[2m]*(2m+1)!, exact */
              stb += cputime ();
              Bm ++;
            }
          mpfr_mul_z (t, j, B[k], GMP_RNDN);
          mpfr_div_ui (t, t, 2 * k, GMP_RNDN);
          mpfr_add (s, s, t, GMP_RNDN);
          if (mpfr_get_exp (t) + (mp_exp_t) p < 0)
            break;
        }
      fprintf (stderr, "Psi(n) needed %u terms\n", k);
      fprintf (stderr, "last B[2k] has %lu bits\n", mpz_sizeinbase (B[k], 2));
      fprintf (stderr, "bernoulli took %dms\n", stb);
#else
      h (P, Q, 0, 10 * N - 1, p, GMP_RNDD);
      mpfr_set_z (s, P, GMP_RNDN);
      mpfr_div_z (s, s, Q, GMP_RNDN);
      mpfr_neg (s, s, GMP_RNDN);
#endif
      st0 = cputime () - st0;
      fprintf (stderr, "Psi(n) needed %dms\n", st0);

      /* now compute R(n^2+1) and subtract it to s */
      st0 = cputime ();
      mpfr_set_ui (j, 1, GMP_RNDN); /* 1/(2k+1)!/z^(2k) */
      overflow = n >= 1 << (mp_bits_per_limb / 2);
      if (overflow)
        {
          mpz_set_ui (tmp, n);
          mpz_mul (tmp, tmp, tmp); /* n^2 */
          mpz_add_ui (tmp, tmp, 1); /* n^2+1 */
          mpz_mul (tmp, tmp, tmp); /* (n^2+1)^2 */
          mpfr_set_prec (i, mpz_sizeinbase (tmp, 2));
          mpfr_set_z (i, tmp, GMP_RNDN);
        }
      for (k = 1; ; k++)
        {
          mpfr_div_ui (j, j, 2 * k, GMP_RNDN);
          mpfr_div_ui (j, j, 2 * k + 1, GMP_RNDN);
          if (overflow)
            {
              mpfr_div (j, j, i, GMP_RNDN);
            }
          else
            {
              mpfr_div_ui (j, j, n * n + 1, GMP_RNDN);
              mpfr_div_ui (j, j, n * n + 1, GMP_RNDN);
            }
          /* compute C[2k] */
```

```
          if (Bm <= k) /* should not happen here */
            {
              stb -= cputime ();
              B = bernoulli (B, k); /* B[2m]*(2m+1)!, exact */
              stb += cputime ();
              Bm ++;
            }
          mpfr_mul_z (t, j, B[k], GMP_RNDN);
          mpfr_div_ui (t, t, 2 * k, GMP_RNDN);
          mpfr_sub (s, s, t, GMP_RNDN);
          if (mpfr_get_exp (t) + (mp_exp_t) p < 0)
            break;
        }
      st0 = cputime () - st0;
      fprintf (stderr, "Psi(n^2+1) needed %u terms and %dms\n", k, st0);
      fprintf (stderr, "bernoulli took %dms\n", stb);
      if (overflow)
        mpfr_set_prec (i, p);

      /* add log((n^2+1)/n), or log(n^2+1) for USE_H */
      mpz_set_ui (tmp, n);
      mpz_mul_ui (tmp, tmp, n);
      mpz_add_ui (tmp, tmp, 1); /* n^2+1 */
      mpfr_set_z (i, tmp, GMP_RNDN);
#ifndef USE_H
      mpfr_div_ui (i, i, n, GMP_RNDN);
#endif
      mpfr_log (i, i, GMP_RNDN);
      mpfr_add (s, s, i, GMP_RNDN);

      /* add 1/(2n) - 1/(2(n^2+1)) = 1/2 * (n^2 - n + 1)/(n^2 + 1)/n */
      mpfr_set_z (i, tmp, GMP_RNDN); /* n^2+1 */
#ifdef USE_H
      mpfr_ui_div (i, 1, i, GMP_RNDN);
      mpfr_neg (i, i, GMP_RNDN);
#else
      mpfr_sub_ui (i, i, n, GMP_RNDN); /* n^2 - n + 1 */
      mpfr_div_z (i, i, tmp, GMP_RNDN);
      mpfr_div_ui (i, i, n, GMP_RNDN);
#endif
      mpfr_div_2exp (i, i, 1, GMP_RNDN);
      mpfr_add (s, s, i, GMP_RNDN);

#ifdef USE_H
      mpfr_const_euler (j, GMP_RNDN);
      mpfr_add (s, s, j, GMP_RNDN);
#endif

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &e, 10, M, s, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo + e, N, M - e) == 0);

  lo += e;
  lo[N] = '\0';
  printf ("%s\n", lo);

  oldBm = Bm;
  while (Bm--)
```

```
    mpz_clear (B[Bm]);
  (*__gmp_free_func) (B, oldBm * sizeof (mpz_t));

  mpfr_clear (i);
  mpfr_clear (j);
  mpfr_clear (t);
  mpfr_clear (s);
  mpz_clear (tmp);
  mpz_clear (P);
  mpz_clear (Q);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C16

```
/* solution to problem C16 from the "many digits friendly competition" */

#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j;
  char *lo, *hi;
  mp_exp_t exp_lo, exp_hi;
  int k;
  int st;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);

  M = N;

  do
    {
      M += 10;

      p = (mp_prec_t) (9.33 * (double) M + 10.0); /* 9.33 ~ log(640)/log(2) */
      fprintf (stderr, "Setting initial precision to %lu\n", p);

      mpfr_set_prec (i, p);
      mpfr_set_prec (j, p);

      mpfr_set_ui (j, 1, GMP_RNDN);
      for (k = N; k >= 1; k--)
        {
          if (k <= 1625)
            mpfr_mul_ui (j, j, k * k * k, GMP_RNDN);
          else if (k <= 65535)
            {
              mpfr_mul_ui (j, j, k * k, GMP_RNDN);
              mpfr_mul_ui (j, j, k, GMP_RNDN);
            }
          else
            {
```

```
              mpfr_mul_ui (j, j, k, GMP_RNDN);
              mpfr_mul_ui (j, j, k, GMP_RNDN);
              mpfr_mul_ui (j, j, k, GMP_RNDN);
            }
          if (k <= 812)
            mpfr_div_ui (j, j, (2 * k + 1) * (2 * k + 1) * (2 * k + 1), GMP_RNDN
);
          else if (k <= 32767)
            {
              mpfr_div_ui (j, j, (2 * k + 1) * (2 * k + 1), GMP_RNDN);
              mpfr_div_ui (j, j, 2 * k + 1, GMP_RNDN);
            }
          else
            {
              mpfr_div_ui (j, j, 2 * k + 1, GMP_RNDN);
              mpfr_div_ui (j, j, 2 * k + 1, GMP_RNDN);
              mpfr_div_ui (j, j, 2 * k + 1, GMP_RNDN);
            }

          mpfr_add_ui (j, j, 21 * k - 8, GMP_RNDN);
          mpfr_div_2exp (j, j, 3, GMP_RNDN);
        }
      mpfr_const_pi (i, GMP_RNDN);
      mpfr_mul (i, i, i, GMP_RNDN);
      mpfr_div_ui (i, i, 6, GMP_RNDN);
      mpfr_sub (i, i, j, GMP_RNDN);

      lo = mpfr_get_str (NULL, &exp_lo, 10, N, i, GMP_RNDZ);
    }
  while (can_round (lo, N, M) == 0);

  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);

  fprintf (stderr, "Cputime: %dms\n", cputime () - st);
  return 0;
}
```

Solution to Problem C17

```
#include <assert.h>
#include "many.h"

#ifndef MPFR_PREC
#define MPFR_PREC(z) mpfr_get_prec(z)
#endif

#ifndef MPFR_INT_CEIL_LOG2
#define MPFR_INT_CEIL_LOG2(n) (unsigned long) ceil(log((double) n) * 1.4426950408889634)
#endif

static void
zeta_ui (mpfr_ptr z3, mpfr_ptr z5, mp_prec_t p)
{
  mp_prec_t pz;
  unsigned long n, k, err3, err5, kbits;
  mpz_t d, t, s3, s5, q3, q5;
```

```
int loop = 0;

assert(p > 5);
pz = p;

mpz_init (s3);
mpz_init (s5);
mpz_init (d);
mpz_init (t);
mpz_init (q3);
mpz_init (q5);

p += MPFR_INT_CEIL_LOG2(p); /* account of the n term in the error */

do
  {
    if (loop++)
      fprintf (stderr, "recompute\n");
    p += MPFR_INT_CEIL_LOG2(p) + 1;
    /* 0.39321985067869744 = log(2)/log(3+sqrt(8)) */
    n = 1 + (unsigned long) (0.39321985067869744 * (double) p);
    err3 = err5 = n + 4;

    mpfr_set_prec (z3, p);
    mpfr_set_prec (z5, p);

    /* computation of the d[k] */
    mpz_set_ui (s3, 0);
    mpz_set_ui (s5, 0);
    mpz_set_ui (t, 1);
    mpz_mul_2exp (t, t, 2 * n - 1); /* t[n] */
    mpz_set (d, t);
    for (k = n; k > 0; k--)
      {
        mpz_ui_pow_ui (q3, k, 3);
        kbits = mpz_sizeinbase (q3, 2);
        /* if k^m is too large, use mpz_tdiv_q */
        if (kbits > 2 * GMP_LIMB_BITS)
          {
            mpz_tdiv_q (q3, d, q3);
          }
        else /* use several mpz_tdiv_q_ui calls */
          {
            unsigned long km = k, mm = 3 - 1;
            while (mm > 0 && km < ULONG_MAX / k)
              {
                km *= k;
                mm --;
              }
            mpz_tdiv_q_ui (q3, d, km);
            while (mm > 0)
              {
                km = k;
                mm --;
                while (mm > 0 && km < ULONG_MAX / k)
                  {
                    km *= k;
                    mm --;
                  }
                mpz_tdiv_q_ui (q3, q3, km);
```

```
            }
        }
      if (k % 2)
        mpz_add (s3, s3, q3);
      else
        mpz_sub (s3, s3, q3);

      mpz_ui_pow_ui (q5, k, 5);
      kbits = mpz_sizeinbase (q5, 2);
      /* if k^m is too large, use mpz_tdiv_q */
      if (kbits > 2 * GMP_LIMB_BITS)
        {
          mpz_tdiv_q (q5, d, q5);
        }
      else /* use several mpz_tdiv_q_ui calls */
        {
          unsigned long km = k, mm = 5 - 1;
          while (mm > 0 && km < ULONG_MAX / k)
            {
              km *= k;
              mm --;
            }
          mpz_tdiv_q_ui (q5, d, km);
          while (mm > 0)
            {
              km = k;
              mm --;
              while (mm > 0 && km < ULONG_MAX / k)
                {
                  km *= k;
                  mm --;
                }
              mpz_tdiv_q_ui (q5, q5, km);
            }
        }
      if (k % 2)
        mpz_add (s5, s5, q5);
      else
        mpz_sub (s5, s5, q5);

      /* we have d[k] = sum(t[i], i=k+1..n)
         with t[i] = n*(n+i-1)!*4^i/(n-i)!/(2i)!
         t[k-1]/t[k] = (2k)*(2k-1)/(n-k+1)/(n+k-1)/4 */
#if (GMP_LIMB_BITS == 32)
#define KMAX 46341 /* max k such that k*(2k-1) < 2^32 */
#elif (GMP_LIMB_BITS == 64)
#define KMAX 3037000500
#endif
#ifdef KMAX
      if (k <= KMAX)
        mpz_mul_ui (t, t, k * (2 * k - 1));
      else
#endif
        {
          mpz_mul_ui (t, t, k);
          mpz_mul_ui (t, t, 2 * k - 1);
        }
      mpz_div_2exp (t, t, 1);
```

```
        if (n < 1UL << (GMP_LIMB_BITS / 2))
          {
            /* (n - k + 1) * (n + k - 1) < n^2 */
            mpz_divexact_ui (t, t, (n - k + 1) * (n + k - 1));
          }
        else
          {
            mpz_divexact_ui (t, t, n - k + 1);
            mpz_divexact_ui (t, t, n + k - 1);
          }
        mpz_add (d, d, t);
      }

    /* multiply by 1/(1-2^(1-m)) = 1 + 2^(1-m) + 2^(2-m) + ... */
    mpz_div_2exp (t, s3, 3 - 1);
    do
      {
        err3 ++;
        mpz_add (s3, s3, t);
        mpz_div_2exp (t, t, 3 - 1);
      }
    while (mpz_cmp_ui (t, 0) > 0);

    /* divide by d[n] */
    mpz_mul_2exp (s3, s3, p);
    mpz_tdiv_q (s3, s3, d);

    mpfr_set_z (z3, s3, GMP_RNDN);
    mpfr_div_2exp (z3, z3, p, GMP_RNDN);

    /* multiply by 1/(1-2^(1-m)) = 1 + 2^(1-m) + 2^(2-m) + ... */
    mpz_div_2exp (t, s5, 5 - 1);
    do
      {
        err5 ++;
        mpz_add (s5, s5, t);
        mpz_div_2exp (t, t, 5 - 1);
      }
    while (mpz_cmp_ui (t, 0) > 0);

    /* divide by d[n] */
    mpz_mul_2exp (s5, s5, p);
    mpz_tdiv_q (s5, s5, d);

    mpfr_set_z (z5, s5, GMP_RNDN);
    mpfr_div_2exp (z5, z5, p, GMP_RNDN);

    err3 = MPFR_INT_CEIL_LOG2 (err3);
    err5 = MPFR_INT_CEIL_LOG2 (err5);
  }
while (mpfr_can_round (z3, p - err3, GMP_RNDN, GMP_RNDZ, pz + 1) == 0 &&
       mpfr_can_round (z5, p - err5, GMP_RNDN, GMP_RNDZ, pz + 1) == 0);

mpz_clear (d);
mpz_clear (t);
mpz_clear (q3);
mpz_clear (q5);
mpz_clear (s3);
mpz_clear (s5);
mpfr_prec_round (z3, pz, GMP_RNDN);
mpfr_prec_round (z5, pz, GMP_RNDN);
```

```
}

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i, j, k;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);
  mpfr_init (j);
  mpfr_init (k);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 4, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);
      mpfr_set_prec (j, p);
      mpfr_set_prec (k, p);

      mpfr_const_pi (i, GMP_RNDN); /* err <= 2^(-p+1) */
      mpfr_sqr (i, i, GMP_RNDN); /* err < 21 * 2^(-p) */
      mpfr_div_ui (i, i, 3, GMP_RNDN); /* err < 9 * 2^(-p) */
      mpfr_sub_ui (i, i, 1, GMP_RNDN); /* err < 9 * 2^(-p) */
      zeta_ui (j, k, p); /* err <= 2^(-p) on j and k */
      mpfr_sub_ui (j, j, 1, GMP_RNDN); /* err <= 2^(-p) */
      mpfr_sub_ui (k, k, 1, GMP_RNDN); /* err <= 2^(-p) */
      mpfr_mul (i, i, j, GMP_RNDN); /* err < 4.5 * 2^(-p) */
      mpfr_add (i, i, k, GMP_RNDN); /* err < 6 * 2^(-p) */
      mpfr_mul_2ui (i, i, 1, GMP_RNDN); /* err < 12 * 2^(-p) */
      /* We chose p to satisfy:
           p > log2(12 * 10^M) = log2(12) + M.log2(10) */

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N, M) == 0);

  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);
  mpfr_clear (j);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C18

```
/* solution to problem C18 from the "many digits friendly competition":
   compute the first 10^N decimal digits after the decimal point of
   Catalan's constant G. */

#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p;
  mpfr_t i;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);
      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);

      mpfr_const_catalan (i, GMP_RNDN);

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N, M) == 0);

  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (i);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C19

```
/* solution to problem C19 from the "many digits friendly competition":
   compute the first 10^N decimal digits after the decimal point of
   L = sum((1/7)^(n^3+1), n=1..infinity). */

#include "many.h"

int
main (int argc, char *argv[])
```

```
{
  unsigned long N = atoi (argv[1]), M;
  mp_prec_t p, q, j;
  mpfr_t i;
  char *lo;
  mp_exp_t exp_lo;
  int st, st0;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  mpfr_init (i);

  M = N;

  do
    {
      M += 10;
      mpfr_set_prec (i, 32);

      /* first compute q = ceil(1 + M*log(10)/log(7)) */
      mpfr_set_d (i, LOG7_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 1, GMP_RNDU);
      q = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Truncating L to %lu digits\n", q);

      lo = (char*) malloc (q + 3);
      memset (lo, '0', q + 2);
      lo[1] = '.';
      lo[q + 2] = '\0';

      for (j = 1; j * j * j < q; j++)
        lo[j * j * j + 2] = '1';

      mpfr_set_d (i, LOG2_10, GMP_RNDU);
      mpfr_mul_ui (i, i, M, GMP_RNDU);
      mpfr_add_ui (i, i, 1, GMP_RNDU);
      p = mpfr_get_ui (i, GMP_RNDU);
      fprintf (stderr, "Setting precision to %lu\n", p);

      mpfr_set_prec (i, p);

      mpfr_set_str (i, lo, 7, GMP_RNDN);

      free (lo);

      st0 = cputime ();
      lo = mpfr_get_str (NULL, &exp_lo, 10, M, i, GMP_RNDN);
      st0 = cputime () - st0;
    }
  while (can_round (lo, N - 1, M) == 0);

  lo[N-1] = '\0';
  printf ("0%s\n", lo);

  mpfr_clear (i);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C20

```
/* implement Lehmer's method (page 4, equation (3)) from
   http://web.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub166.html
*/

#include "many.h"

/* Error on alpha < 10.7 * 2^(-p) */
static unsigned long *
confrac (mpfr_t alpha, unsigned long *n, mp_prec_t prec)
{
  unsigned long *l, alloc, a;
  mpz_t beta0, beta1, gamma0, gamma1, g0, g1;
  mp_size_t n0;

  alloc = 1;
  l = malloc (alloc * sizeof (unsigned long));
  *n = 0; /* number of stored elements */

  mpz_init (beta0);
  mpz_init (beta1);
  mpz_init (gamma0);
  mpz_init (gamma1);
  mpz_init (g0);
  mpz_init (g1);

  mpfr_mul_2exp (alpha, alpha, prec, GMP_RNDN); /* err < 10.7 */
  mpfr_get_z (beta1, alpha, GMP_RNDD);
  mpz_sub_ui (gamma1, beta1, 10);
  mpz_add_ui (beta1, beta1, 11);
  mpz_set_ui (beta0, 1);
  mpz_mul_2exp (beta0, beta0, prec);
  mpz_set (gamma0, beta0);

  /* beta0/beta1 < 1/alpha < gamma0/gamma1 */

  while (1)
    {
      /* we must have gamma0 >= gamma1 */
      n0 = mpz_sizeinbase (gamma0, 2);
      n0 = (n0 >= mp_bits_per_limb) ? n0 - mp_bits_per_limb : 0;
      mpz_tdiv_q_2exp (g0, gamma0, n0);
      mpz_cdiv_q_2exp (g1, gamma1, n0);

      /* we have g0 = floor(gamma0/2^n0) and g1 = ceil(gamma1/2^n0)
         thus floor (g0/g1) <= floor(gamma0/gamma1) */

      a = mpz_get_ui (g0) / mpz_get_ui (g1);

      mpz_submul_ui (gamma0, gamma1, a);
      /* if a was too small, then gamma0 >= gamma1 */
      while (mpz_cmp (gamma0, gamma1) > 0)
        {
          mpz_sub (gamma0, gamma0, gamma1);
          a ++;
        }
      mpz_submul_ui (beta0, beta1, a);
      if (mpz_cmp_ui (beta0, 0) < 0 || mpz_cmp (beta0, beta1) >= 0)
        break;
      mpz_swap (gamma0, gamma1);
      mpz_swap (beta0, beta1);
```

```
      if (*n >= alloc)
        {
          alloc = 2 * alloc;
          l = realloc (l, alloc * sizeof (unsigned long));
        }
      l[*n] = a;
      (*n)++;

    }

  mpz_clear (beta0);
  mpz_clear (beta1);
  mpz_clear (gamma0);
  mpz_clear (gamma1);
  mpz_clear (g0);
  mpz_clear (g1);

  l = realloc (l, *n * sizeof (unsigned long));

  return l;
}

static void
compsin (mpfr_t s, mp_prec_t p)
{
  mpfr_t x, y, z;

  mpfr_set_prec (s, p);
  mpfr_inits2 (p, x, y, z, (void *) 0);

  /* Note: sqrt(x+eps) - sqrt(x) < eps / (2 * sqrt(x)) */

  mpfr_sqrt_ui (x, 17, GMP_RNDN); /* err < 2^(2-p) */
  mpfr_mul_2ui (z, x, 1, GMP_RNDN); /* err < 2^(3-p) */
  mpfr_ui_sub (y, 34, z, GMP_RNDN); /* err < (8+16) * 2^(-p) = 24 * 2^(-p) */
  mpfr_add_ui (z, z, 34, GMP_RNDN); /* err < (8+32) * 2^(-p) = 40 * 2^(-p) */
  mpfr_sqrt (y, y, GMP_RNDN); /* err < (2.4 + 4) * 2^(-p) = 6.4 * 2^(-p) */
  mpfr_sqrt (z, z, GMP_RNDN); /* err < (3.1 + 4) * 2^(-p) = 7.1 * 2^(-p) */
  mpfr_mul_2ui (z, z, 3, GMP_RNDN); /* err < 56.8 * 2^(-p) */
  mpfr_mul_ui (s, x, 6, GMP_RNDN); /* err < (24+16) * 2^(-p) = 40 * 2^(-p) */
  mpfr_add_ui (s, s, 34, GMP_RNDN); /* err < (40+32) * 2^(-p) = 72 * 2^(-p) */
  mpfr_sub (s, s, z, GMP_RNDN); /* err < (72 + 56.8) * 2^(-p) */
  mpfr_sub_ui (z, x, 1, GMP_RNDN); /* err < 2^(2-p) */
  mpfr_mul (z, z, y, GMP_RNDN); /* err < 48.3 * 2^(-p) */
  mpfr_add (s, s, z, GMP_RNDN); /* err < (128.8 + 48.3 + 16) * 2^(-p) */
  mpfr_mul_2ui (s, s, 1, GMP_RNDN); /* err < 387 * 2^(-p) */
  mpfr_sqrt (s, s, GMP_RNDN); /* err < (28.8 + 4) * 2^(-p) = 32.8 * 2^(-p) */
  mpfr_add (x, x, y, GMP_RNDN); /* err < 18.4 * 2^(-p) */
  mpfr_add (s, s, x, GMP_RNDN); /* err < 59.2 * 2^(-p) */
  mpfr_sub_ui (s, s, 1, GMP_RNDN); /* err < 59.2 * 2^(-p) */
  mpfr_div_2ui (s, s, 4, GMP_RNDN); /* err < 3.7 * 2^(-p) */
  mpfr_sqr (s, s, GMP_RNDN); /* err < 7.5 * 2^(-p) */
  mpfr_ui_sub (s, 1, s, GMP_RNDN); /* err < 7.5 * 2^(-p) */
  mpfr_sqrt (s, s, GMP_RNDN); /* err < 10.7 * 2^(-p) */

  mpfr_clears (x, y, z, (void *) 0);
}

int
main (int argc, char *argv[])
{
```

```
  unsigned long N = atoi (argv[1]), n;
  mp_prec_t p;
  mpfr_t s;
  mpz_t z1, z2;
  int st, st1;
  unsigned long *l;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  /* initial estimate of the working precision */
  p = (mp_prec_t) (3.45 * (double) N + 17.0);
  fprintf (stderr, "Setting initial precision to %lu\n", p);

  mpfr_init (s);
  mpz_init (z1);
  mpz_init (z2);

  do
    {
      st1 = cputime ();
      compsin (s, p);
      fprintf (stderr, "Computing X took %dms\n", cputime () - st1);

      l = confrac (s, &n, p);
      fprintf (stderr, "found %lu partial quotients\n", n);

      if (n >= N)
        break;
      else
        {
          p += p / 10;
          fprintf (stderr, "Increasing precision to %lu\n", p);
        }
    }
  while (1);

  printf ("%lu\n", l[N - 1]);

  mpfr_clear (s);
  mpz_clear (z1);
  mpz_clear (z2);
  free (l);

  fprintf (stderr, "Cputime: %dms\n", cputime () - st);
  return 0;
}
```

Solution to Problem C21

```
#include "many.h"

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), M;
  int st, st0;
  mpfr_t x, y, z, u;
  mp_prec_t Q, p, q, newq;
  char *lo;
  mp_exp_t exp_lo;
```

```
fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
st = cputime ();

mpfr_init2 (x, 32);
mpfr_init (y);
mpfr_init (z);
mpfr_init (u);

M = N + 10;

mpfr_set_d (x, LOG2_10, GMP_RNDU);
mpfr_mul_ui (x, x, M, GMP_RNDU);
Q = mpfr_get_ui (x, GMP_RNDU);
fprintf (stderr, "Target precision is %lu\n", Q);

/* initialization */
mpfr_set_prec (x, 2);
mpfr_set_d (x, 1.3, GMP_RNDN);
q = 2;

/* invariant: we have |x-rho| <= 2^(-q), it is true here since rho ~ 1.302 */
while (1)
  {
    if (q < Q)
      {
        for (newq = Q; newq > 2 * q; newq = (newq + 1) / 2);
        /* now newq <= 2 * q */
      }
    else
      {
        M += 10;
        mpfr_set_d (x, LOG2_10, GMP_RNDU);
        mpfr_mul_ui (x, x, M, GMP_RNDU);
        Q = mpfr_get_ui (x, GMP_RNDU);
        fprintf (stderr, "Target precision is %lu\n", Q);
        newq = Q;
      }

    p = newq + 4;

    /* start iteration */
    mpfr_prec_round (x, p, GMP_RNDN); /* extend x to p bits */
    mpfr_set_prec (y, p);
    mpfr_set_prec (z, p);
    mpfr_set_prec (u, p);
    mpfr_sin_cos (y, z, x, GMP_RNDN);
    mpfr_exp (z, z, GMP_RNDN); /* t */
    mpfr_sub (u, x, z, GMP_RNDN);
    mpfr_mul (y, y, z, GMP_RNDN); /* v */
    mpfr_add_ui (y, y, 1, GMP_RNDN); /* w */
    mpfr_div (y, u, y, GMP_RNDN);
    mpfr_sub (x, x, y, GMP_RNDN);

    /* try is precision is enough */
    q = newq;

    if (q >= Q)
      {
        st0 = cputime ();
        lo = mpfr_get_str (NULL, &exp_lo, 10, M + 1, x, GMP_RNDN);
        st0 = cputime () - st0;
```

```
          if (can_round (lo + 1, N, M))
            break;
        }
    }

  lo ++; /* to skip the initial 2 */
  lo[N] = '\0';
  printf ("%s\n", lo);

  mpfr_clear (x);
  mpfr_clear (y);
  mpfr_clear (z);
  mpfr_clear (u);

  fprintf (stderr, "Cputime: %dms (output %dms)\n", cputime () - st, st0);
  return 0;
}
```

Solution to Problem C22

```
#include "many.h"
#include "crq.h"

#define DEBUG

/* On [0, 1], |f^{(100 n)}| <= 2^{BD[n]}
 * THIS IS STILL A CONJECTURE */
unsigned long BD[] = {
    2, 451, 1071, 1760, 2494, 3261, 4055, 4871 };

/* Bound the grows of f^{2n} between 100k and 100k + 100 */
unsigned long growth[] = { 5, 7, 7, 8, 8, 8, 9, 9 };

unsigned long bound_f2n (unsigned long n)
{
    unsigned long step, rem;
    if ((n > 800) || (n < 2))
        abort ();

    step = n / 100;
    rem = n % 100;
    return BD[step] + rem * growth[step];
}

void sin3 (mpfr_t res, mpfr_t x, mp_prec_t rndm)
{
    mpfr_set_prec (res, mpfr_get_prec (res) + 10);
    mpfr_sin (res, x, rndm);
    mpfr_sin (res, res, rndm);
    mpfr_sin (res, res, rndm);
    mpfr_prec_round (res, mpfr_get_prec (res) - 10, GMP_RNDN);
}

int main (int argc, char *argv[])
{
    long total_time;
    mpfr_t a, b, res, reslo, reshi, M, un;
    mpfr_t matherror, evalerror, totalerror;
    mp_prec_t p;
    char *lo, *hi;
    mp_exp_t exp_lo, exp_hi;
```

```
    char *filename;
    unsigned long m, n, wp, correct_bits;
    unsigned long N = atoi (argv[1]);

    fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
    total_time = cputime ();

    mpfr_init (a);
    mpfr_init (b);
    mpfr_init (res);
    mpfr_init (reshi);
    mpfr_init (reslo);
    mpfr_init (evalerror);
    mpfr_init (matherror);
    mpfr_init (totalerror);
    mpfr_init (M);
    mpfr_init (un);
    mpfr_set_ui (a, 0, GMP_RNDN);
    mpfr_set_ui (b, 1, GMP_RNDN);
    mpfr_set_ui (un, 1, GMP_RNDN);
    /* m, n, wp "opt" m = 133; n = 150; wp = 3490; */
    n = atoi (argv[2]);
    m = atoi (argv[3]);
    p = (mp_prec_t) (LOG2_10 * (double) N + 10.0);
    wp = p + 42;
    /* p is the expected binary accuracy to reach to be able to get
     * an accuracy of N decimal digits. wp is the binary working precision */
    do {
#ifdef DEBUG
        fprintf (stderr, "Starting GL with n = %ld, m = %ld, wp = %ld\n", n,
                 m, wp);
#endif
        mpfr_set_prec (res, wp);
        /* Do we want to reuse a precomputed table ? */
        if (argc >= 5) {
            filename = argv[4];
            fprintf (stderr, "Creating/reusing precomputed table %s\n",
                     argv[4]);
        } else
            filename = NULL;
        compose_gl (res, evalerror, a, b, m, n, sin3, un, wp, 0, filename);
        mpfr_set_ui (M, 1, GMP_RNDN);
        mpfr_mul_2si (M, M, bound_f2n (n), GMP_RNDU);
        gl_error (matherror, a, b, m, n, M);
        mpfr_add (totalerror, matherror, evalerror, GMP_RNDU);
        correct_bits = mpfr_get_exp (res) - mpfr_get_exp (totalerror);
#ifdef DEBUG
        fprintf (stderr, "Correct bits = %ld, wanted %ld\n", correct_bits, p);
        fprintf (stderr, "Exp res = %ld", mpfr_get_exp (res));
        fprintf (stderr, " exp error = %ld\n", mpfr_get_exp (totalerror));
        mpfr_dump (totalerror);
#endif
        if (correct_bits >= p) {
            mpfr_set_prec (reslo, wp);
            mpfr_set_prec (reshi, wp);
            mpfr_sub (reslo, res, totalerror, GMP_RNDD);
            mpfr_add (reshi, res, totalerror, GMP_RNDU);
            lo = mpfr_get_str (NULL, &exp_lo, 10, N, reslo, GMP_RNDZ);
```

```
            hi = mpfr_get_str (NULL, &exp_hi, 10, N, reshi, GMP_RNDZ);
            if ((exp_lo == exp_hi) && (strcmp (lo, hi) == 0))
                break;
            else { /* p insufficient */
                p += p / 10;
                wp += wp / 10;
            }
        } else {
            if (mpfr_cmp (evalerror, matherror) > 0)
                wp += wp / 10;
            else
                n += n / 10;
        }
    } while (42 == 42);

    printf ("%s\n", lo);
    total_time = cputime () - total_time;
    fprintf (stdout, "Cputime: %dms\n", total_time);
    return 0;
}
```

Solution to Problem C23

```
#define USE_MPFI
#include "many.h"

#define THRES_EXP (1 << 25)

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]), k, ok, flag = 0;
  mp_prec_t p;
  char *lo, *hi;
  mp_exp_t exp_lo, exp_hi;
  mpfi_t fib0, fib1, fibtmp;
  mpfi_t num, den, renorm;
  int st;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();

  /* initial estimate of the working precision */
  p = (argc > 2 ? atoi(argv[2]) :
       (N <= 10000000 ? 100 : 10000));
  fprintf (stderr, "Setting initial precision to %lu\n", p);

  mpfi_init2(fib0, p);
  mpfi_init2(fib1, p);
  mpfi_init2(fibtmp, p);
  mpfi_init2(den, p);
  mpfi_init2(num, p);

  do
    {
      mpfi_set_ui(fib0, 1);
      mpfi_set_ui(fib1, 1);
      mpfi_set_ui(den, 1);

      for (k = 3; k <= N-4; k++)
```

```
  {
    mpfi_add(fibtmp, fib0, fib1);
    mpfi_mul(den, den, fibtmp);
    if (RIGHT(den)->_mpfr_exp > THRES_EXP)
      {
        if (flag)
          mpfi_div(den, den, renorm);
        else
          {
            flag = 1;
            mpfi_init2(renorm, p);
            mpfi_set_ui(renorm, 10);
            mpfi_log(renorm, renorm);
            mpfi_mul_ui(renorm, renorm,
                        (unsigned long)(THRES_EXP *
                                        0.30102999566398119521373889));
            mpfi_exp(renorm, renorm);
            mpfi_div(den, den, renorm);
          }
      }
    mpfi_swap(fib0, fib1);
    mpfi_swap(fib1, fibtmp);
  }

mpfi_add(fibtmp, fib0, fib1); /* F_(N-3) */
mpfi_swap(fib0, fib1);
mpfi_swap(fib1, fibtmp);

mpfi_add(fibtmp, fib0, fib1); /* F_(N-2) */
mpfi_swap(fib0, fib1);
mpfi_swap(fib1, fibtmp);

mpfi_ui_div(num, 1, den);

for (k = N - 1; k <= 2*N - 6; k++)
  {
    mpfi_add(fibtmp, fib0, fib1); /* F_k */
    mpfi_mul(num, num, fibtmp);
    /* If exponent grows too large, renormalize */
    if (RIGHT(num)->_mpfr_exp > THRES_EXP)
      {
        if (flag)
          mpfi_div(num, num, renorm);
        else
          {
            flag = 1;
            mpfi_init2(renorm, p);
            mpfi_set_ui(renorm, 10);
            mpfi_log(renorm, renorm);
            mpfi_mul_ui(renorm, renorm,
                        (unsigned long)(THRES_EXP *
                                        0.30102999566398119521373889));
            mpfi_exp(renorm, renorm);
            mpfi_div(num, num, renorm);
          }
      }
    mpfi_swap(fib0, fib1);
    mpfi_swap(fib1, fibtmp);
  }
```

```
      mpfi_mul(num, num, num);
      mpfi_div_ui(num, num, 2);

      mpfi_add(fibtmp, fib0, fib1); /* F_{2N-5} */
      mpfi_swap(fib0, fib1);
      mpfi_swap(fib1, fibtmp);

      mpfi_mul(num, num, fib1);

      mpfi_add(fibtmp, fib0, fib1); /* F_{2N-4} */
      mpfi_swap(fib0, fib1);
      mpfi_swap(fib1, fibtmp);

      mpfi_mul(num, num, fib1); mpfi_mul(num, num, fib1);

      mpfi_add(fibtmp, fib0, fib1); /* F_{2N-3} */
      mpfi_mul(num, num, fibtmp);
      mpfi_swap(fib0, fib1);
      mpfi_swap(fib1, fibtmp);

      mpfi_add(fibtmp, fib0, fib1); /* F_{2N-2} */
      mpfi_mul(num, num, fibtmp);

      lo = mpfr_get_str (NULL, &exp_lo, 10, 10, LEFT(num), GMP_RNDZ);
      hi = mpfr_get_str (NULL, &exp_hi, 10, 10, RIGHT(num), GMP_RNDZ);
      ok = exp_lo == exp_hi && strcmp (lo, hi) == 0;

      if (ok)
        break;
      else
        {
          p = p+p/10;
          fprintf(stderr, "Increasing precision to %d.\n", p);
          mpfi_set_prec(num, p);
          mpfi_set_prec(den, p);
          mpfi_set_prec(fib0, p);
          mpfi_set_prec(fib1, p);
          mpfi_set_prec(fibtmp, p);
        }
    }
  while (1);

  mpfi_clear(num); mpfi_clear(den); mpfi_clear(fibtmp);
  mpfi_clear(fib0); mpfi_clear(fib1);

  printf ("%s\n", lo);

  fprintf (stderr, "Cputime: %dms\n", cputime () - st);
  return 0;
}
```

Solution to Problem C24

```
#define USE_MPFI
#include "many.h"
#include "gauss.h"

/* row and column of wanted element of the inverse
   (one less than in original problem, since indices start at 0, not 1)
*/
#define ROW(N) (N-1)-1
#define COL(N) (N)-1
```

```
/*
N     10 20 50 100 200 500 1000
pmin  40 39 42  43  42  46   46
*/

int
main (int argc, char *argv[])
{
  unsigned long N = atoi (argv[1]);
  mp_prec_t p;
  char *lo, *hi;
  mp_exp_t exp_lo, exp_hi;
  mpfi_t **A, *b;
  unsigned long i, j;
  int st, ok;
  mpz_t fib;

  fprintf (stderr, "Using GMP %s and MPFR %s\n", gmp_version, mpfr_version);
  st = cputime ();
  mpz_init (fib);

  /* initial estimate of the working precision */
  p = (mp_prec_t) (argc > 2) ? atoi (argv[2]) : 46;
  fprintf (stderr, "Setting initial precision to %lu\n", p);

  A = malloc (N * sizeof (mpfi_t*));
  b = malloc (N * sizeof (mpfi_t));
  for (i = 0; i < N; i++)
    {
      mpfi_init2 (b[i], p);
      A[i] = malloc (N * sizeof (mpfi_t));
      for (j = 0; j < N; j++)
        mpfi_init2 (A[i][j], p);
    }

  do
    {
      /* A[i][j] = 1/fib(i+j-1) + delta(i,g) */
      for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
          {
            mpz_fib_ui (fib, i + j + 1);
            mpfi_set_z (A[i][j], fib);
            mpfi_ui_div (A[i][j], 1, A[i][j]);
            if (i == j)
                mpfi_add_ui (A[i][j], A[i][j], 1);
          }
      for (i = 0; i < N; i++)
        mpfi_set_ui (b[i], (i == COL(N)) ? 1 : 0);

      ok = gauss (A, b, N) == 0;

      if (ok)
        {
          lo = mpfr_get_str (NULL, &exp_lo, 10, 10, LEFT(b[ROW(N)]), GMP_RNDZ);
          hi = mpfr_get_str (NULL, &exp_hi, 10, 10, RIGHT(b[ROW(N)]), GMP_RNDZ);
          ok = exp_lo == exp_hi && strcmp (lo, hi) == 0;
        }

      if (ok)
        break;
```

```
      else
        {
          p += p / 10;
          fprintf (stderr, "Increasing precision to %lu\n", p);
          for (i = 0; i < N; i++)
            {
              mpfi_set_prec (b[i], p);
              for (j = 0; j < N; j++)
                mpfi_set_prec (A[i][j], p);
            }

        }
    }
  while (1);

  for (i = 0; i < N; i++)
    {
      mpfi_clear (b[i]);
      for (j = 0; j < N; j++)
        mpfi_clear (A[i][j]);
      free (A[i]);
    }
  free (b);
  free (A);

  printf ("%s\n", lo);

  fprintf (stderr, "Cputime: %dms\n", cputime () - st);
  return 0;
}
```

A.8 RealLib

The RealLib solution is a single C++ program called `comp.cpp`. This program is compiled using the command

```
g++ -O3    -c -o comp.o comp.cpp
g++ -O3  -o comp.out comp.o ../Real.a -lm -lstdc++
```

and is then invoked as

```
time ./comp.out 1 100000 > c.1.100000
```

The source of comp.cpp looks like

```
#include <iostream>
#include <strstream>
#include <iomanip>
#include "Real.h"
#include "gaussian.h"
using namespace std;
using namespace RealLib;
```

various function definitions

```
void PrintAfterPoint(const Real &r, int v)
{
   ostrstream os;
   os << fixed << setprecision(v);
```

```
   os << r;
   char *p = os.str();
   char *pp = strchr(p, '.');
   assert(pp);
   pp[v+1] = 0;
   cout << pp+1;
}

void PrintFromFirstNonZero(const Real &r, int v)
{
        ostrstream os;
        os << scientific << showpos << setprecision(v-1);
        os << r.ForceNonZero();
        char *p = os.str();
        //cout << p << endl;
        cout << p[1]; // first digit
        p[v+2] = 0;       // cut exponent
        cout << p+3; // skip .
}

int main(int argc, char **argv) {
    InitializeRealLib();
    int v = 10;
         int problem = 24;

     if (argc > 2)
        v = atoi(argv[2]);

          if (argc > 1)
                  problem = atoi(argv[1]);

          if (v <= 0) return -1;

    {
        Real e(exp(Real(1.0)));
                  Real r;

                  switch (problem) {
                                  various cases
                          default:
                                  cout << "not available\n";
                                  return -1;
                  }

        //cout << r << endl;

                  switch (problem) {
                                  various cases
                          default:
                                  cout << "not available\n";
                                  return -1;
                  }
                  cout << endl;
    }
    FinalizeRealLib();
    return 0;
}
```

The first switch statement calculates `r`, while the second switch statement prints it. In the solutions below the relevant lines from these switch statements have been combined. The various function definitions are there given before the first statement that needs them.

Solution to Problem C01

```
r = (sin(tan(cos(Real(1.0)))));
PrintAfterPoint(r, v);
```

Solution to Problem C02

```
r = (sqrt(exp(Real(1.0))/Pi));
PrintAfterPoint(r, v);
```

Solution to Problem C03

```
template <class TYPE>
TYPE ppow(const TYPE &b, UserInt p)
{
        //if (p<0) return recip(pow(b, -p));
        if (p==0) return TYPE(1.0);
        TYPE q(sq(ppow(b, p/2)));
        return p%2 ? q*b : q;
}
CreateUnaryAndIntRealFunction(ppow)

r = (sin(ppow(e+1, 3)));
PrintAfterPoint(r, v);
```

Solution to Problem C04

```
r = (exp(Pi * sqrt(Real(2011.0))));
PrintAfterPoint(r, v);
```

Solution to Problem C05

```
r = (exp(exp(sqrt(exp(Real(1.0))))));
PrintAfterPoint(r, v);
```

Solution to Problem C06

```
Real atanh(const Real &x)
{
    return log((1+x)/(1-x))/2;
}

r = (atanh(1-atanh(1-atanh(1-atanh(recip(Pi))))));
PrintAfterPoint(r, v);
```

Solution to Problem C07

```
template <class Real>
Real pow(const Real &b, const Real &e)
{
    return exp(log(b)*e);
}

r = (pow(Pi, Real(1000)));
PrintAfterPoint(r, v);
```

Solution to Problem C08

```
r = (sin(ppow(Real(6), ppow(6, 6))));
PrintAfterPoint(r, v);
```

Solution to Problem C09

```
r = (sin(10 * atan(tanh(Pi * sqrt(Real(2011))/3))));
PrintAfterPoint(r, v);
```

Solution to Problem C10

```
Real q = pow(Real(2), Real(1)/5);
Real s = sq(q);
r = pow(7+q - 5*s*q, Real(1)/3) + s - q;
PrintAfterPoint(r, v);
```

Solution to Problem C11

```
r = (tan(sqrt(Real(2.0))) + atanh(sin(Real(1.0))));
PrintAfterPoint(r, v);
```

Solution to Problem C12

```
Real asinh(const Real &x)
{
    return log(x + sqrt(sq(x)+1));
}

Real s(sq(e));
r = (asin(recip(s))+asinh(s));
PrintAfterPoint(r, v);
```

Solution to Problem C13

```
template <class TYPE>
TYPE P13(const long prec, UserInt v)
{
    TYPE t(0.5);
    //TYPE c("3.999");
    for (int i=0;i<v;++i)
        t = 3999*t*(1-t)/1000;
    return t;
}
CreateIntRealFunction(P13);

r = (P13(v));
PrintAfterPoint(r, v);
```

Solution to Problem C14

```
template <class TYPE>
TYPE P14(const long prec, UserInt v)
{
    TYPE a(TYPE(14.0)/3);
```

```
         TYPE b(5);
    TYPE c(TYPE(184.0)/35);
    for (int i=0;i<v;++i) {
        TYPE t = 114 - (1463 - (6390-9000/a)/b)/c;
        a = b;
        b = c;
                  c = t;
    }
    return b;
}
CreateIntRealFunction(P14);

r = (P14(v*100));
PrintAfterPoint(r, v);
```

Solution to Problem C15

```
template <class TYPE>
TYPE P15(const long prec, UserInt v)
{
    TYPE t(0.0);
    TYPE one(1.0);
         UserInt e=v*v;
         if ((e & 0xffffffff) == e) {
           int ie = int(e&0xffffffff);
                for (int i=v;i<=ie;++i)
                          t += one/i;
         } else {
                for (double i=v;i<=double(e);++i)
                        t += recip(TYPE(i));
         }
    return t;
}
CreateIntRealFunction(P15)

r = (P15(v*10));
PrintAfterPoint(r, v);
```

Solution to Problem C16

```
template <class TYPE>
TYPE P16(const long prec, UserInt v)
{
        TYPE m(1.0);

        for (int i=i32(v);i>0;--i) {
                TYPE i3 = TYPE(i)*i*i;
                TYPE j3 = i3*8 + i*6*TYPE(2*i+1) + 1;
                m *= i3;
                m /= j3 * 8;
                m += TYPE(21*i-8)>>3;
        }
        return sq(pi<TYPE>(prec))/6 - m;
}
CreateIntRealFunction(P16)

r = (P16(v));
PrintFromFirstNonZero(r, v);
```

Solution to Problem C17

```
template <class TYPE>
TYPE zeta(const long prec, UserInt v)
{
        int n = int((prec*32.0+3.0) * log(2.0) /
                log(5.8));      // this works for v at least 2

        TYPE d(1.0);
        TYPE m(1.0);    // n * n-1! / n!
        int i;
        for (i=0;i<n;++i) {
                m *= 2 * (n + i) * (n - i);
                m /= (i+1) * (2*i+1);           // this is m1
                d += m;                                                 // d1
        }
        // mn and dn here

        TYPE dn(d);

        // restart d
        m = 1.0;
        d = m;
        int s(1);
        TYPE t(0.0);
        for (i=0;i<n;++i) {
                t += s* (d-dn) / ppow(TYPE(i+1), v);

                m *= 2 * (n + i) * (n - i);
                m /= (i+1) * (2*i+1);
                d += m;

                s = -s;

        }
        t /= (dn * ((TYPE(2.0) >> i32(v)) - 1));

        return t;
        // no AddError needed since the formula is exact to the prec request
        //.AddError(TYPE(1.0)>>(prec*32));
}
CreateIntRealFunction(zeta)

Real z2(zeta(2));
Real z3(zeta(3));
r = -4*z2 - 2*z3 + 4*z2*z3 + 2*zeta(5);
PrintAfterPoint(r, v);
```

Solution to Problem C18

```
template <class TYPE>
TYPE catalan(const int prec)
{
        int n = prec * 32 / 4;

        TYPE t(0.0);
        TYPE q(0.0);

        for (int k=0;k<n;++k) {
                TYPE k8(TYPE(64.0)*(k*k));
                int k16(k*16);
```

```
                k8 = k8 + (k16 + 1);
                TYPE u(recip(k8));
                TYPE v(u);
                TYPE w(u);
                k8 = k8 + (k16 + 3);
                u = recip(k8);
                v = v - u;
                w = w + (u>>1);
                k8 = k8 + (k16 + 5);
                u = recip(k8);
                v = v + (u>>1);
                w = w + (u>>3);
                k8 = k8 + (k16 + k16 + 7+9);
                u = recip(k8);
                v = v - (u>>2);
                w = w - (u>>6);
                k8 = k8 + (k16 + 11);
                u = recip(k8);
                v = v + (u>>2);
                w = w - (u>>7);
                k8 = k8 + (k16 + 13);
                u = recip(k8);
                v = v - (u>>3);
                w = w - (u>>9);
                t += v >> (k*4);
                q += w >> (k*12);
        }

        return (6*t - q) >> 2;
}
CreateRealConstant(Catalan, catalan)

r = Catalan;
PrintAfterPoint(r, v);
```

Solution to Problem C19

```
template <class TYPE>
TYPE P19fun(const int prec)
{
        int n = int(pow(prec * 32 * log(2.0)/log(7.0),
                0.33333333333))+1;
        // +1 is included in formula

        TYPE t(recip(TYPE(7.0)));
        TYPE s(0.0);
        TYPE i2(1.0);
        TYPE i3(2.0);
        for (int i=1;i<n;++i) {
                s += ppow(ppow(t, i*i), i)*t;
                //i2 += 6*i;
                //i3 += i2;
        }

        return s.AddError(TYPE(1.0)>>(prec*32));
}
CreateRealConstant(P19, P19fun)
```

```
r = P19;
PrintAfterPoint(r, v);
```

Solution to Problem C23

```
template <class TYPE>
TYPE P23(unsigned prec, UserInt N)
{
   int i,j;
        typedef valarray<TYPE> RealVec;
        typedef valarray<RealVec> RealMat;
        RealVec row(TYPE(), N+N);
        RealMat mat(row, N);

        TYPE a(1.0);
        TYPE b(1.0);
        row[0] = a;
        row[1] = b;
        for (int i=2;i<N+N;++i) {
                TYPE t = a+b;
                row[i] = recip(t);
                b = a;
                a = t;
        }

   for (i=0;i<N;++i) {
      for (j=0;j<N;++j) {
         mat[i][j] = row[i+j];
         mat[i][N+j] = 0.;
      }
      mat[i][N+i] = 1.;
   }

   doGaussian(mat);

        int m = N - 1 - 1;
        int n = N - 3 - 1;

        return mat[m][N+n];
}
CreateIntRealFunction(P23)

r = P23(v);
PrintFromFirstNonZero(r, 10);
```

Solution to Problem C24

```
template <class TYPE>
TYPE P24(unsigned prec, UserInt N)
{
   int i,j;
        typedef valarray<TYPE> RealVec;
        typedef valarray<RealVec> RealMat;
        RealVec row(TYPE(), N+N);
        RealMat mat(row, N);

        TYPE a(1.0);
        TYPE b(1.0);
        row[0] = a;
```

```
        row[1] = b;
        for (int i=2;i<N+N;++i) {
                TYPE t = a+b;
                row[i] = recip(t);
                b = a;
                a = t;
        }

   for (i=0;i<N;++i) {
      for (j=0;j<N;++j) {
         mat[i][j] = row[i+j];
         mat[i][N+j] = 0.;
      }
      mat[i][N+i] = 1.;
   }

        for (i=0;i<N;++i)
                mat[i][i] += 1.;

   doGaussian(mat);

        int m = N - 1 - 1;
        int n = N - 1;

        return mat[m][N+n];
}
CreateIntRealFunction(P24)

r = P24(v);
PrintFromFirstNonZero(r, 10);
```

A.9 Wolfram

The solution by Wolfram using Mathematica did not store anything in a file. Instead the commands were just typed in into the text only version of Mathematica called `math`. (The reason for not making a notebook out of this undoubtedly was that this way a trace of what was done could be stored using `ttyrecord`.) Into this `math` program for each problem first was typed

```
1+1
```

such that the loading of the Mathematica kernel would not spoil the timing of the actual command. Then the precision for problems C01 until C07 was set with

```
$MaxExtraPrecision = 1000
```

and for problems C08 until C12 with

```
$MaxExtraPrecision = 100000
```

After that the functions

```
ManyDigits4[k_] :=
  Timing[FromDigits[Take[First[RealDigits[Mod[k, 1], 10, 10010, -1]], -10]]]
ManyDigits5[k_] :=
  Timing[FromDigits[Take[First[RealDigits[Mod[k, 1], 10, 100010, -1]], -10]]]
ManyDigits6[k_] :=
  Timing[FromDigits[Take[First[RealDigits[Mod[k, 1], 10, 1000010, -1]], -10]]]
```

were defined. Finally the command from the solution was executed.

Solution to Problem C01

```
ManyDigits5[Sin[Tan[Cos[1]]]]
```

Solution to Problem C02

```
ManyDigits5[Sqrt[E/Pi]]
```

Solution to Problem C03

```
ManyDigits5[Sin[(E + 1)^3]]
```

Solution to Problem C04

```
ManyDigits5[Exp[Pi Sqrt[2011]]]
```

Solution to Problem C05

```
ManyDigits5[Exp[Exp[Exp[1/2]]]]
```

Solution to Problem C06

```
ManyDigits5[ArcTanh[1 - ArcTanh[1 - ArcTanh[1 - ArcTanh[1/Pi]]]]]
```

Solution to Problem C07

```
ManyDigits6[Pi^1000]
```

Solution to Problem C08

```
ManyDigits5[Sin[6^(6^6)]]
```

Solution to Problem C09

```
ManyDigits5[Sin[10 ArcTan[Tanh[Pi Sqrt[2011]/3]]]]
```

Solution to Problem C10

```
Timing[N[(7 + 2^(1/5) - 5*(8^(1/5)))^(1/3) + 4^(1/5) - 2^(1/5), 1000000]]
```

Solution to Problem C11

```
ManyDigits5[Tan[Sqrt[2]] + ArcTanh[Sin[1]]]
```

Solution to Problem C12

```
ManyDigits5[ArcSin[1/E^2] + ArcSinh[E^2]]
```

B Raw Final Timings and Used Commands

In table below 'Command' refers to the command together with arguments on which the `/usr/bin/time` was applied. Sometimes `/usr/bin/time` itself is replaced by `@`. The actual paths on the competition machine were longer and are simplified or altered for clarity. In the commands for the iRRAM system, `-pf` should be replaced by `-prec_factor`.

#	Command	user	sys	real	%
C01	`~bignum/compete 1 10 > bignum_C01_1`	43.66	0.89	45.03	98
C02	`~bignum/compete 2 100 > bignum_C02_2`	4.52	0.62	5.19	99
C03	`~bignum/compete 3 100 > bignum_C03_2`	3.19	0.34	3.57	99
C04	`~bignum/compete 4 100 > bignum_C04_2`	33.41	0.79	34.52	99
C05	`~bignum/compete 5 100 > bignum_C05_2`	97.89	0.26	99.04	99
C07	`~bignum/compete 7 100 > bignum_C07_2`	19.06	0.82	20.07	99
C02	`~comp/C02 100000 > comp_C02_5`	111.49	0.07	112.24	99
C04	`~comp/C04 10000 > comp_C04_4`	4.8	0	4.93	97
C05	`~comp/C05 10000 > comp_C05_4`	12.98	0.01	13.26	98
C07	`~comp/C07 50000 > comp_C07_4`	3.69	0	3.93	94
C13	`~comp/C13 10000 10000 > comp_C13_4`	106.1	0.07	107.19	99
C15	`~comp/C15 10 100 +RTS -K100000000 > comp_C15_1`	0.71	0	0.81	87
C01	`~fewdigits/c01 > fewdigits_C01_3`	23.34	0.01	23.49	99
C02	`~fewdigits/c02 > fewdigits_C02_3`	2.28	0	2.33	98
C03	`~fewdigits/c03 > fewdigits_C03_3`	5.02	0	5.1	98
C04	`~fewdigits/c04 > fewdigits_C04_3`	28.46	0.02	28.78	98
C05	`~fewdigits/c05 > fewdigits_C05_3`	43.52	0.05	44.01	99
C06	`~fewdigits/c06 > fewdigits_C06_1`	0.46	0	0.49	93
C09	`~fewdigits/c09 > fewdigits_C09_3`	86.95	0.09	88.19	98
C10	`~fewdigits/c10 > fewdigits_C10_2`	8.79	0.01	8.91	98
C11	`~fewdigits/c11 > fewdigits_C11_2`	20.98	0.03	21.23	98
C12	`~fewdigits/c12 > fewdigits_C12_2`	14.16	0.02	14.33	98
C01	`~ginac/c01 > ginac_C01_5`	16.63	0.37	17.36	97
C02	`~ginac/c02 > ginac_C02_6`	13.17	0.26	13.64	98
C03	`~ginac/c03 > ginac_C03_5`	6.2	0.13	6.47	97
C04	`~ginac/c04 > ginac_C04_5`	4.19	0.06	4.4	96
C05	`~ginac/c05 > ginac_C05_5`	6.39	0.12	6.66	97
C06	`~ginac/c06 > ginac_C06_5`	12.27	0.26	12.27	102
C07	`~ginac/c07 > ginac_C07_6`	11.94	0.22	12.36	98
C08	`~ginac/c08 > ginac_C08_5`	6.52	0.13	6.8	97
C09	`~ginac/c09 > ginac_C09_5`	13.29	0.29	13.79	98
C10	`~ginac/c10 > ginac_C10_6`	8.66	0.19	9.03	98
C11	`~ginac/c11 > ginac_C11_5`	15.02	0.32	15.58	98
C12	`~ginac/c12 > ginac_C12_5`	9.02	0.2	9.38	98
C13	`~ginac/c13 > ginac_C13_4`	21.35	0	21.7	98
C14	`~ginac/c14 > ginac_C14_2`	56.63	0	57.48	98
C15	`~ginac/c15 > ginac_C15_2`	1.6	0	1.85	86

#	Command	user	sys	real	%
C16	`~ginac/c16 > ginac_C16_4`	2.38	0	2.47	96
C17	`~ginac/c17 > ginac_C17_4`	2.3	0.01	2.42	95
C18	`~ginac/c18 > ginac_C18_5`	9.2	0.2	9.58	98
C19	`~ginac/c19 > ginac_C19_6`	64.44	2.37	67.52	98
C20	`~ginac/c20 > ginac_C20_4`	24.75	0	25.17	98
C21	`~ginac/c21 > ginac_C21_4`	9.62	0	9.79	98
C22	`~ginac/c22 > ginac_C22_2`	78.21	0.01	79.36	98
C23	`~ginac/c23 > ginac_C23_2`	43.01	0.01	43.53	98
C24	`~ginac/c24 > ginac_C24_2`	1.58	0	1.7	93
C01	`echo 100000 | @~irram/C01 -pf=1.05 > irram_C01_5`	25.7	0.27	26.37	98
C02	`echo 100000 | @~irram/C02 -pf=1.05 > irram_C02_5`	2.84	0.03	2.92	98
C03	`echo 100000 | @~irram/C03 -pf=1.05 > irram_C03_5`	8.75	0.09	8.94	98
C04	`echo 100000 | @~irram/C04 -pf=1.05 > irram_C04_5`	52.79	0.44	53.74	99
C05	`echo 100000 | @~irram/C05 -pf=1.05 > irram_C05_5`	10.9	0.06	11.08	99
C06	`echo 100000 | @~irram/C06 -pf=1.05 > irram_C06_5`	20.94	0.12	21.28	98
C07	`echo 1000000 | @~irram/C07 -pf=1.05 > irram_C07_6`	73.72	0.79	75.23	99
C08	`echo 100000 | @~irram/C08 -pf=1.05 > irram_C08_5`	9.46	0.12	9.69	98
C09	`echo 100000 | @~irram/C09 -pf=1.05 > irram_C09_5`	19.28	0.17	19.64	99
C10	`echo 1000000 | @~irram/C10 -pf=1.05 > irram_C10_6`	35.73	0.1	36.19	99
C11	`echo 100000 | @~irram/C11 -pf=1.05 > irram_C11_5`	29.03	0.22	29.54	99
C12	`echo 100000 | @~irram/C12 -pf=1.05 > irram_C12_5`	24.63	0.16	25.05	99
C13	`echo 10000 | @~irram/C13 -pf=1.05 > irram_C13_4`	6.03	0	6.11	98
C14	`echo 100 | @~irram/C14 -pf=1.02 > irram_C14_2`	30.11	0	30.41	99
C15	`echo 10000 | @~irram/C15 -pf=1.05 > irram_C15_4`	5.03	0	5.1	98
C16	`echo 10000 | @~irram/C16 -pf=1.05 > irram_C16_4`	1.28	0	1.31	97
C17	`echo 10000 | @~irram/C17 -pf=1.05 > irram_C17_4`	8.32	0	8.43	98
C18	`echo 100000 | @~irram/C18 -pf=1.05 > irram_C18_5`	47.96	0.1	48.58	98
C19	`echo 1000000 | @~irram/C19 -pf=1.05 > irram_C19_6`	37.6	0.09	38.07	98
C20	`echo 10000 | @~irram/C20 -pf=1.25 > irram_C20_4`	1.11	0	1.15	97
C21	`echo 100000 | @~irram/C21 -pf=1.05 > irram_C21_5`	18.57	0.12	18.89	98
C22	`echo 1000 | @~irram/C22 -pf=1.05 > irram_C22_3`	19.79	0.02	20.01	98
C23	`echo 100 | @~irram/C23 -pf=1.05 > irram_C23_2`	57.31	0.08	57.92	99
C24	`echo 100 | @~irram/C24 -pf=1.05 > irram_C24_2`	1.26	0.01	1.3	97
C01	`~maple/maple -c "N := 5" ~/maple/c01.mpl`	0.02	0.02	96.94	0
C02	`~maple/maple -c "N := 5" ~/maple/c02.mpl`	0.02	0.01	7.54	0
C03	`~maple/maple -c "N := 5" ~/maple/c03.mpl`	0.01	0.01	45.72	0
C04	`~maple/maple -c "N := 5" ~/maple/c04.mpl`	0.01	0.01	21.16	0
C05	`~maple/maple -c "N := 5" ~/maple/c05.mpl`	0.01	0.01	33.82	0
C06	`~maple/maple -c "N := 4" ~/maple/c06.mpl`	0.01	0.02	3.5	0
C07	`~maple/maple -c "N := 5" ~/maple/c07.mpl`	0.01	0.01	4.11	0
C08	`~maple/maple -c "N := 5" ~/maple/c08.mpl`	0.01	0.01	49.93	0
C09	`~maple/maple -c "N := 5" ~/maple/c09.mpl`	0.01	0.02	67.18	0
C10	`~maple/maple -c "N := 4" ~/maple/c10.mpl`	0.01	0.02	1.71	2
C11	`~maple/maple -c "N := 5" ~/maple/c11.mpl`	0.02	0.02	99.39	0
C12	`~maple/maple -c "N := 5" ~/maple/c12.mpl`	0.01	0.01	54.64	0

#	Command	user	sys	real	%
C13	`~maple/maple -c "N := 2" ~/maple/c13.mpl`	0.01	0.01	0.12	28
C14	`~maple/maple -c "N := 2" ~/maple/c14.mpl`	0.01	0.02	35.98	0
C17	`~maple/maple -c "N := 3" ~/maple/c17.mpl`	0.01	0.02	10.3	0
C18	`~maple/maple -c "N := 4" ~/maple/c18.mpl`	0.01	0.01	2.99	1
C21	`~maple/maple -c "N := 3" ~/maple/c21.mpl`	0.01	0.02	1.06	3
C21*	`~maple/maple ~/maple/c21_extra_10.mpl`	0.01	0.01	0.69	4
C22	`~maple/maple -c "N := 2" ~/maple/c22.mpl`	0.01	0.01	1.06	3
C01	`~mpfr/c01 100000 > mpfr_C01_5`	17.79	0	18.09	98
C02	`~mpfr/c02 1000000 > mpfr_C02_6`	31.93	0.52	32.79	98
C03	`~mpfr/c03 100000 > mpfr_C03_5`	9.08	0.01	9.21	98
C04	`~mpfr/c04 100000 > mpfr_C04_5`	11.6	0.03	11.79	98
C05	`~mpfr/c05a 100000 > mpfr_C05_5`	5.11	0.01	5.21	98
C06	`~mpfr/c06 100000 > mpfr_C06_5`	8.18	0	8.31	98
C07	`~mpfr/c07a 1000000 > mpfr_C07_6`	24.8	0.39	25.45	98
C08	`~mpfr/c08a 100000 > mpfr_C08_5`	8.7	0.01	8.82	98
C09	`~mpfr/c09 100000 > mpfr_C09_5`	33.05	0.25	33.63	99
C10	`~mpfr/c10 100000 > mpfr_C10_5`	4.05	0.05	4.17	98
C11	`~mpfr/c11 100000 > mpfr_C11_5`	24.18	0.05	24.46	99
C12	`~mpfr/c12 100000 > mpfr_C12_5`	8.48	0.03	8.62	98
C13	`~mpfr/c13 10000 > mpfr_C13_4`	9.14	0	9.24	98
C14	`~mpfr/c14tmp 100 40933 > mpfr_C14_2`	45	0.02	45.42	99
C15	`~mpfr/c15a 10000 > mpfr_C15_4`	7.18	0.01	7.29	98
C16	`~mpfr/c16 10000 > mpfr_C16_4`	0.87	0	0.89	97
C17	`~mpfr/c17best 100000 > mpfr_C17_5`	115.76	0.03	116.86	99
C18	`~mpfr/c18 100000 > mpfr_C18_5`	8.09	0.09	8.31	98
C19	`~mpfr/c19 1000000 > mpfr_C19_6`	3.34	0.06	3.46	98
C20	`~mpfr/c20b 100000 > mpfr_C20_5`	4.26	0	4.33	98
C21	`~mpfr/c21 100000 > mpfr_C21_5`	14.79	0.01	14.97	98
C22	`~mpfr/c22 1000 218 26 > mpfr_C22_3`	26.21	0	26.6	98
C23	`~mpfr/c23-c 10000000 > mpfr_C23_7`	18.46	0	18.68	98
C24	`~mpfr/c24 100 > mpfr_C24_2`	0.2	0	0.22	92
C01	`~reallib/comp.out 1 10000 > reallib_C01_4`	1.98	0	2.08	95
C02	`~reallib/comp.out 2 10000 > reallib_C02_4`	11.53	0	11.64	99
C03	`~reallib/comp.out 3 10000 > reallib_C03_4`	14.61	0	14.76	99
C04	`~reallib/comp.out 4 10000 > reallib_C04_4`	11.56	0	11.99	96
C05	`~reallib/comp.out 5 10000 > reallib_C05_4`	17.94	0	18.15	98
C06	`~reallib/comp.out 6 10000 > reallib_C06_4`	30.85	0	31.16	99
C07	`~reallib/comp.out 7 10000 > reallib_C07_4`	17.37	0	17.55	98
C08	`~reallib/comp.out 8 10000 > reallib_C08_4`	56.26	0.01	56.8	99
C09	`~reallib/comp.out 9 10000 > reallib_C09_4`	20.19	0	20.39	99
C10	`~reallib/comp.out 10 10000 > reallib_C10_4`	26.42	0	26.72	98
C11	`~reallib/comp.out 11 10000 > reallib_C11_4`	20.62	0	20.82	99
C12	`~reallib/comp.out 12 10000 > reallib_C12_4`	20.5	0	20.74	98
C13	`~reallib/comp.out 13 10000 > reallib_C13_4`	83.21	0.01	84.01	99
C14	`~reallib/comp.out 14 10 > reallib_C14_1`	5.88	0	5.94	99

#	Command	user	sys	real	%
C15	`~reallib/comp.out 15 100 > reallib_C15_2`	1.68	0	1.69	99
C16	`~reallib/comp.out 16 1000 > reallib_C16_3`	6.41	0	6.47	99
C17	`~reallib/comp.out 17 1000 > reallib_C17_3`	18.19	0	18.36	99
C18	`~reallib/comp.out 18 1000 > reallib_C18_3`	2.04	0	2.07	98
C19	`~reallib/comp.out 19 100000 > reallib_C19_5`	47.06	0.01	47.53	99
C23	`~reallib/comp.out 23 10 > reallib_C23_1`	0.01	0	0.01	93
C24	`~reallib/comp.out 24 100 > reallib_C24_2`	0.07	0	0.07	98
C01	`cat ~/wolfram/c01.ma | @~wolfram/math`	9.5	0.15	10.85	88
C02	`cat ~/wolfram/c02.ma | @~wolfram/math`	0.58	0.07	0.67	96
C03	`cat ~/wolfram/c03.ma | @~wolfram/math`	3.6	0.09	3.74	98
C04	`cat ~/wolfram/c04.ma | @~wolfram/math`	2.15	0.09	2.28	98
C05	`cat ~/wolfram/c05.ma | @~wolfram/math`	3.5	0.09	3.65	98
C06	`cat ~/wolfram/c06.ma | @~wolfram/math`	6.78	0.12	6.96	99
C07	`cat ~/wolfram/c07.ma | @~wolfram/math`	11.08	0.57	11.78	98
C08	`cat ~/wolfram/c08.ma | @~wolfram/math`	4.24	0.1	4.17	104
C09	`cat ~/wolfram/c09.ma | @~wolfram/math`	6.94	0.12	7.14	98
C10	`cat ~/wolfram/c10.ma | @~wolfram/math`	21.55	0.87	22.62	99
C11	`cat ~/wolfram/c11.ma | @~wolfram/math`	8.72	0.15	8.96	98
C12	`cat ~/wolfram/c12.ma | @~wolfram/math`	5.54	0.12	5.73	98