

Possible ways to improve Olympiads in Informatics

Mārtiņš OPMANIS

*Institute of Mathematics and Computer Science, University of Latvia,
29 Raina Boulevard, Riga, LV-1459, Latvia*

Martins.Opmanis@mii.lu.lv

Abstract. The paper describes some possible ways how to improve Olympiads in Informatics. Currently, in Olympiads tests are prepared by jury only. Ways to involve contestants in testing process are investigated. Possible scoring schemas are discussed. Ways to improve solution debugging process are described. In International Olympiads tasks with real numbers are quite rare. Possible reasons are investigated and a way how to return such tasks back to competition arena is suggested. Several kinds of tasks different from the traditional ones are presented.

Keywords. Olympiads in Informatics, testing, competition tasks, grading.

1. Introduction

Olympiads in Informatics (OI) are high school student competitions which have the following basic features:

- **Individual.** Contestant does not get support and help from teammates and other persons. There is a lot of other competitions where several contestants work together as a team. In fact, teamwork is more close to real work in industry. However, due to historical reasons for OI this feature still remains unchanged and changes are not planned.

- **In-present.** Nearly every international OI now has its satellite online competition in which every interested person may participate and solve the same task set in the same time limits as registered participants do. If competition has serious formal impact (such as team selection for higher-level OI), such competitions can be made “semi-online” by adding supervisors and a restriction that contestants are allowed to work only in specified workplace (still remote from the main competition place).

- **One or two competition rounds, five hours each.** Almost all OI follow this standard. Each competition round is provided in a separate day (and in this sense “competition round” and “competition day” may be used as synonyms), there may be a leisure day in-between. In Latvia, the first two OI levels (school and region) are one day competitions, but higher levels OI (Latvian, Baltic and International) are provided as two round competitions. Traditionally, three tasks per round are given.

- **Tasks are mainly algorithmic oriented.** The main goal for students is finding and implementing a correct and efficient algorithm solving the given problem. Tasks including heuristics and some presence of “luckiness”, as a rule, are avoided or at least including such tasks in problem set usually raises serious discussions. At this moment there are three kinds of tasks accepted in international arena:

1. Tasks where solution – a computer program must be submitted,
2. Tasks, where a set of input files is given and the corresponding output files must be submitted (this will be discussed briefly later in this paper),
3. Reactive tasks, where given library must be used (will not be discussed in this paper).

- **Solutions are tested automatically** on a previously prepared test set and correctness is determined by checking equivalence of results or checking it using the grader.

For international Olympiads (at least in the European region and International OI) one more point must be added:

- **Every contestant obtains task descriptions in English (this version is binding) and in his native tongue.** Translations are prepared in advance by team leaders or adjuncts of the team.

In this paper by mentioning OI (without more precise explanation) I will assume any OI in the hierarchy of Olympiads, starting with lower level and ending with the top level - International OI (IOI). Complete IOI rules can be found in [1].

Now I will give brief insight how a competition round is performed (assuming that all tasks are of the first kind (see above)):

- At the beginning, the contestant receives **task descriptions**, where the task together with **input** and **output** data formats is described. **Limits** of task data range and time limits for one test execution also are given. Task description also contains **example** input and the corresponding output. Maximum number of points per task is **100**.
- Created solutions must be **submitted** to grading system where they are compiled and executed on simple test cases (usually the same as given in task description as examples). This process usually is called **pre-testing**. If output of the program is correct, the submission receives the status "**accepted**", otherwise it is rejected.
- **Last submitted** solution version with status "accepted" is considered as the **final version** of the solution.
- After completion of competition round, the final version is tested on the **full test set**. This process usually is called **final testing**. Every test in this set has some number of **points** which are awarded to contestant if this test passes **with correct result in the given time limit**. Otherwise solution receives no points for this test.

The rest of paper proceeds as follows. Section 2 describes weaknesses in existing grading procedure and indicates some effort of contestants that is not reflected in final score. Ways how to involve contestants in overall testing process are described in Section 3. Some ways how to improve program debugging is discussed in Section 4. Section 5 describes problems connected with usage of real numbers in competition tasks and one approach how to overcome this is suggested. Section 6 gives examples of some kinds of tasks used in local Latvian competition's and different from usual.

2. Testing and debugging

For me, automated testing is one of the biggest last decade achievements in OI area. Despite the fact, that such testing excludes human grader and therefore shortens the list of kinds of possible tasks (pushing out theoretical essays and proofs), at the same time, such an approach radically increases comparison objectivity of submitted solutions. It is impossible to imagine smooth grading in international arena without such a procedure.

However, there still are some deficiencies:

- The author of test set (same as etalon solution) is jury. It is possible, that test set may be in some sense "misbalanced", especially if the number of tests is too low (I'm not speaking about trivial problems). For example, on IOI'99 there were problems which had one simple test and nine hard ones, leading to low scores and giving no reasonable result distribution. As the result, the 50% rule was invented (in task descriptions additional limits were stated, fulfilling

which 50% of points were guaranteed), which led to surprising results on IOI'05 when several contestants had chosen the relatively safe way to get half of points by solving task in these "light" limits and not attempting the "hard" ones.

In practice, also technical mistakes happen. Disagreement between task description and some test case (due to changed formulation, variable limits or simply typo) is quite usual. As a rule, this is discovered by some contestant or delegation leader only after investigation of testing results. Such mistakes are corrected afterwards and all submitted solutions are re-tested on the new test set.

- If we look at the final version of submitted task solution from the "industrial" viewpoint, we can see that the expected result is similar to "release version" in terms of software industry. In OI practice a working version of a weaker algorithm is better than a perfect implementation containing one typo. I'm sure that any team leader can tell you stories about silly mistakes where one erroneous symbol led to 0 points for a task. I will call such programs "one wrong symbol" programs. One such situation I saw myself was at IOI'99 where a contestant got 0 points, but by changing "N" to "M" in a "for" loop the solution could be made worth of 100 points, still, according to rules this was not counted. To be honest, rarely a contestant may be "lucky", when solution is not penalized due to "gaps" in used test set. For example, on CEOI'97 contestant got 90% of score despite the fact that he implemented a solution for square areas instead of rectangular ones mentioned in the task description.

So, summarizing, what we have today after a competition round for each of tasks (still assuming only tasks of the first kind) is:

- one test set, made by jury;
- N+1 solution (N contestant solutions and (best possible) jury solution).

Today, testing systems allow very fast testing of all N submitted solutions even if there are hundreds of them. To be more precise, by distributing computing power between several computers it is possible to test solutions during competition right after their submissions and bottleneck is now printing of results.

During preparing their solutions contestants are forced to test their solutions by themselves (again, I'm not speaking about trivial tasks, where correctness can be verified even without serious testing).

Till now, this self-testing effort is partially thrown away. The value of such "internal" or "local" testing allows for contestant himself either have a feeling that solution is correct (if program solves all constructed test cases) or shows incapability to write such program (there is test case which can not be solved by the written program).

3. One test set vs. Total testing

What if, together with the solution program, it would be allowed for the contestant to submit also test cases prepared for testing of his own program? If all contestants would submit their tests (more precisely, test sets) together with their solutions, we would get N+1 solution together with N+1 test set. Testing every solution on such a large amount of tests will lead to deeper testing of each particular solution. To mention it once again - technically it can be done in reasonable time limits. Existing solution submission system must be improved by adding possibility to submit test sets and some checker must be written to validate tests.

Positive features of such an approach are:

1. Testing is quite a serious part of software creation process. Efforts made in this direction also must be awarded. In OI practice there are examples when by creating good test examples contestant recovers his inability to write a correct solution. If a test set will be submitted without a working solution, I would vote for granting some points

also for this. In software industry, programmers and testers are everyday opponents but we should not underestimate tester's side, because creative test creation is not an easy job;

2. If contestant's solution is capable of solving N+1 test set created by different authors (300+ on IOI), it is a more serious product than a program passing only one test set. With all respect to jury as task authors, in a particular problem, there may be some hidden traps which are not covered by jury test set, but can be shown when tested on test sets of other contestants.

So, let's assume that together with problem solution the contestant submits also test set - a set of tests, where every test case is supplied also with amount of points which will be awarded if the program passes this particular test case (natural additional constraints are positive integer number of points for particular test case and the total sum equivalent to 100).

It is easy to imagine two formats of test case data:

- Input file and the corresponding output file (appropriate for "unique correct output" tasks).
- Input file and a program producing output file (for "non-unique correct output" tasks).

It is obvious that in the second case a wrong program leads to incorrect test set, while in the first case it is possible to construct correct test cases without a proper solution program. For example, this could be done by a correct, but inefficient solution. Submitting input and output files in non-unique correct output case fits in category "solution of open input task" and is not subject of analysis in this paper.

Conclusion is: submission of test sets can be technically accomplished. But how about weaknesses?

Unfortunately, there are some:

1. Only "good" tests are worth of scoring some amount of points. Moreover - we are looking for good test sets, not only particular tricky test cases. If points will be granted simply "for tests", then there is a potential danger to get a lot of simple tests or generated hard ones without any shadow of inspiration. It is hard to formulate criteria for a "good test set". However, some rules can be stated:
 - Perfect solution must be given 100 points;
 - Partially correct solution must not be given 100 points;
 - Partially correct solution must be given non-zero amount of points (this is very slippery point, because "one wrong symbol" program mentioned above will get 0 on nearly all test sets and this is not fault of the test set);
 - Non-efficient (slow) solution must not be given 100 points;
 - Non-efficient, but correct solution must be given some non-zero amount of points (this rule corresponds with IOI 50% rule);
 - Wrong program must not be given significant amount of points (how to separate wrong programs from partially correct programs?)
2. What if some task is so hard, that all contestants from programmers revert to testers and submit only test sets? Would this be acceptable or not?
3. It is not obvious how points must be given for separate test cases. One possible approach is to execute all programs on all original tests (avoiding executing the same test more than once by excluding repeating tests during test merge process) and normalizing score to $100/\text{maximum_possible_amount_of_points}$.

However, in this case, if all contestants submit too easy or too hard test sets, then the results can be seriously biased. Solution to this could be awarding a part of points (say, the famous 50%) according to tests prepared by jury and, the remaining part - according to tests prepared by contestants.

Another possible way could be to look at the testing process as a great tournament, where each contestant program fights with every contestant's (yes - even with himself) test set. In the case of three contestants it can look like this:

		Tests submitted by:			
		Jury	Contestant A	Contestant B	Contestant C
Program submitted by:	Jury	Passed all tests (100)	Passed all tests (100)	Tests not submitted (0)	Passed partially (95)
	Contestant A	Passed all tests (100)	Passed all tests (100)	Tests not submitted (0)	Passed all tests (100)
	Contestant B	Passed all tests (100)	Passed all tests (100)	Tests not submitted (0)	Passed partially (80)
	Contestant C	Passed partially (90)	Passed partially (55)	Tests not submitted (0)	Passed partially (85)

As you can see, Contestant A got 300 points against 295 got by Jury, and Contestant C failed on some of his own tests. If tests are not submitted (Contestant B), then all participants get equivalent number of points (it can be either 0 - there were no tests, or 100 - if none of the test submitted by this contestant was failed). Weakness of this "tournament" approach is that contestants are stimulated in submitting only tricky tests to "catch" opponents, and not balanced test sets for throughout analysis of programs. In the actual OI only one (grayed) column is taken into account.

Also another version of tournament is possible: **source code** of every contestant program which passed all jury tests is given to all contestants. The task now would be - find **one** test case which this particular ("perfect" from the viewpoint of jury!) program **does not pass** and receive some points for this case. In example given above, A and B programs would be investigated, and C can potentially raise his score by finding a test which "beats" B's program. At the end, all total scores must be normalized to 100 points and the contestant who got full score after jury's testing and did not find (or even did not try to find) any mistakes in opponent's programs may be overtaken by a diligent tester who was not perfect after jury's tests.

In the case of "tournaments", jury solution must step down from pedestal and stay in one line together with all other solutions. Our expectation says that jury solution must pass all tests in the given time limits and contestants will not found counterexamples investigating jury source code. But what to do, if the answer of jury solution differs from contestants one on some test case? What if such test cases are more than one? Obvious answer is that preparation level of jury solution must be improved to exclude any unconscious solution linking to test set. Can this be done in practice? Untrivial disagreement between jury's and contestants solutions easily leads to additional trouble after the first competition day when usually jury must start to worry about the second round tasks. Besides that, to check quality of submitted test sets, jury must also prepare in advance a bunch of incorrect and inefficient solutions.

Concluding thoughts about testing: my vision is that test sets made by contestants or their testing capabilities may be included in grading process and total number of points could be calculated as a sum of points for program and sum of points for test set.

4. α -version, β -version, Release!

If we come back to "one wrong symbol" programs, it is quite clear that in real life such mistakes are found and corrected quite easily, sometimes even in seconds. On OI there is no way how to manage this and nearly every year there are protests or begging for making exceptions in rules to accept such solutions which, in fact, **are** really good.

One possible solution could be giving some amount of time for correction of mistakes when the results of full-range testing are announced (similar to ACM competitions). If this time is not very big (say, one hour for three tasks), there is no way to rewrite all from scratch, but is plenty of time for correcting small mistakes (and creating new ones). However, it may be hard to change existing timetables of international OI, where right after a competition round is dinner and contestants can discuss their solutions so making impossible further return to solutions not breaking at the same time the idea of individual competition.

A more realistic approach could be permission to check solutions on the full jury test set. If it would show that results in all test cases are perfect, contestant would know in advance that his solution is perfect, but in case of total testing he can be sure that solution is at least "fairly good". If the result after this "first try" is not perfect, there is time for error fixing (and making new errors). Of course, if the contestant works in a hurry and has no time to validate, then the old schema works - last submitted program which passes initial tests, qualifies for full grading and the obtained result will be final.

To avoid some "gambling" ("first try" gives some information about test set and solution is modified in such a way to pass only this particular test set) number of these "preliminary testings on a complete test set" must not be big. One, or maybe two, but no more. It will help on testing, but in contradistinction to ACM, solution can get points also if the task is not solved perfectly. To stimulate thinking before sending there can be involved a rule that the total amount of points is calculated as $(\text{points_in_first_version} + \text{points_in_final_version})/2$. If there is only one submission, then these two numbers are equal. If we use this formula, a contestant who solves the task perfectly and makes only one submission, still has advantage. Of course, such counting does not exclude possibility to worsen the first result (say, 90 points after the first testing, 40 points after the final testing due to new errors leads to overall score 65).

There could be also other grading schemas: only final version result is taken into account and in case of equivalent results higher rank has the contestant who used less testings (like counting jumps in high jump competitions). In case of usage of tests made by other contestants, this "one version" approach is preferable to keeping records of several distinct program versions for each contestant.

Together with visible pros there are also serious contras. There will be much more serious responsibility on jury during preparation of the final test set - it simply must be PERFECT. If usually final test set becomes "visible" only after competition round during final testing, now possible errors will have impact during the first "big submissions". Failing in some test case due to jury mistake can mislead contestant forcing him seek for nonexistent error so wasting time or stimulating making inappropriate changes and submitting a worse version. Also server workload must be limited to avoid overloading in last competition round minutes due to lot of requests for first try full testings.

5. Rehabilitation of real numbers

During the last years in international OI only a few tasks were using real numbers. Such dislike can be explained by understanding the difficulties in testing of results, where exact equivalence of answers, as a rule, is impossible.

There are two widely used approaches how the expected precision of answer is defined asking for:

- some number of correct digits after decimal point;
- some number of correct significant digits;
- limited allowed difference between contestant's and jury's results.

In the automated Latvian OI task testing system "OLIMPS" [2] we are using a different approach for tasks with real answers (see [3], [4]).

Let's assume a usual OI task of a first kind where some input file is given and some output file containing one real number must be produced. Task description includes one of the necessary conditions for answers, given above.

In our approach, the original input will be appended by additional 100 real numbers - possible answers, one of which is "correct" (the answer obtained by jury). Task section is changed in such a way that instead of original "compute and output" must be stated "compute and decide which of given possible answers is closest to your result, where "closest" means that absolute value of difference between your result and this answer among all possible answers is minimum." And instead of real number in original formulation now one **integer** - index of answer must be written in file. Obviously, this simplifies testing of solutions. The number 100 was chosen arbitrary just to minimize the risk of successful guessing and avoiding essential additional computations. Possible answers can be given in sorted or random order - generally it does not matter..

Thus the original solution program must be extended by adding code like the following:

```
 {"result" at this point is computed real answer}
readln(input,possible_answer);
i_best := 1;
delta_best := abs (result - possible_answer);
for i:=2 to 100 do
begin
  readln(input,possible_answer);
  delta := abs (result - possible_answer);
  if delta<delta_best
  then
  begin
    i_best := i;
    delta_best := delta;
  end;
end;
```

and instead of output of `result` in original formulation, now `i_best` must be written as answer in output file. Such improvements can be done by an average OI contestant quite easy.

But is this "modified description" the same as the original one in the sense of precision and how this precision can be managed?

If, in the possible answer set, the gap between two given answers is 2ϵ (assuming the same gap between any two neighbor answers), then the contestant must compute his result with precision ϵ to get closer to the correct answer (see Fig.1). This corresponds directly with the third accuracy criterion given above and the value obtained by contestant must fit in the same region. The only difference is in case if correct value is minimum or maximum in the given set of possible answers. To avoid such situation, the correct answer must be strictly an inside value.

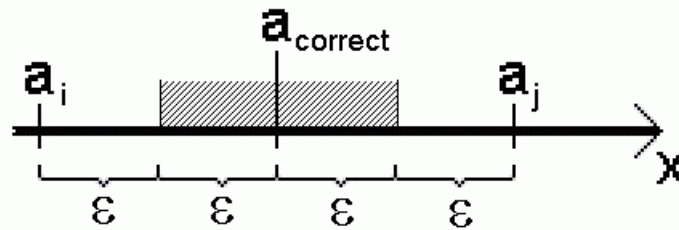


Figure 1. Acceptance region. a_i , a_j , a_{correct} - three of 100 given possible answers,
 a_{correct} - correct answer

Even more - jury can test accuracy of contestant's solution, giving similar tests with the same essential part and changing only possible answers section by increasing or decreasing gaps between values. It is a well-known practice [5] that the same task may be given asking for different precision of result. In the described approach in test cases only ϵ must be changed.

Unfortunately, the described approach cannot be used in all tasks without a preliminary analysis. It cannot be used in tasks where knowledge of answer values gives essentially new information. For example, if original task asks for finding coordinates of the point common to all given triangles, the above-mentioned switching to new data format essentially decreases the level of task, because now you need just to check all possible answers without any other calculations which is not possible in original task.

6. Some other kinds of tasks

In this section I would like to share some experience in preparing different kind of tasks for the annual (since 1996) team competition in mathematics and informatics "Ugāle" [6]. In the final round team of three contestants may use one computer and must solve ten tasks in 5½ hours. Specifics of tasks is here that any solution is short in its form (one number, short program, etc.). These competitions are conducted by Ugāle secondary school teacher Aivars Žogla and a lot of my work is inspired by his ideas.

Task 5.1 "Smart program" (finals of 1998)

Write a program in Pascal, C or BASIC, which

- *outputs on the monitor number 1998 only once;*
- *if program code is modified replacing one symbol by another in such a way that program still compiles and executes, it still only once outputs the number 1998.*

Despite the short codes submitted, grading was done by three previous year IOI participants and each program text was investigated by careful looking and trying to broke the code. Because there were only 12 teams, this task was successfully completed and only those programs where experts did not find any faults got full score. Technically possible, but hard to imagine is testing of this task without such a group of experts. In 2004 I tried to manage grading of task similar to 5.1 only by myself and failed, so be careful with such tasks!

Task 5.2 "Function of functions" (finals of 2005)

Function $f(x)$ is defined for all integers from 1 to 2000000000 and function values are positive integers. Function values are known for one hundred argument values:

x	f(x)
1	2
2	1
3	2
5	2
7	2
9	3
10	5
11	3
13	3
15	5
16	8
17	3
21	7
22	11
23	3
33	11
67	7
77	11
105	35
107	7
109	7
111	37
115	23
117	39
119	17
123	41
125	25
126	63
131	11
137	11
499	19
3055	611
3059	437
3061	53

x	f(x)
3071	83
3073	439
3381	1127
3383	199
3385	677
3403	83
3419	263
3493	499
173005	34601
173007	57669
173009	10177
173021	409
173027	2437
173029	7523
173049	57683
173051	1321
173419	4687
173973	57991
173975	34795
173983	9157
173989	677
9173003	1310429
9173005	1834601
9173009	23581
9173011	3023
9174653	5639
9174661	20899
9174667	295957
9174671	834061
9174673	6323
987654321	329218107
987654325	197530865
987654327	329218109

x	f(x)
987654329	4312901
987654331	9588877
987654791	31397
987654793	31397
987654803	31397
987654809	83227
987654821	31397
987654857	141093551
987654861	329218287
987654865	197530973
987654881	57559
987654883	31397
987654901	31397
987654971	48341
987688883	6059441
987688885	197537777
1999999001	285714143
1999999003	44711
1999999005	666666335
1999999009	32786869
1999999015	399999803
1999999019	155171
1999999027	153846079
1999999037	42553171
1999999039	4640369
1999999957	7782101
1999999961	54054053
1999999967	285714281
1999999969	181818179
1999999975	399999995
1999999981	285714283
1999999997	37735849
1999999999	64516129

Your task is to write **as short as possible** program in one of the programming languages Pascal, C or C++, which implements this function for all x values given in table above. You must not worry about the values other than given, because your program will be tested only with the argument values given in the table.

Function source code must be presented in one separate file and usage of other data files is prohibited. Program must read one x value from standard input and the corresponding $f(x)$ value must be written on a screen and program must exit without waiting for additional user input (such as pressing a key). Program may use only modules and libraries included in compiler's standard configuration.

If for any of hundred given argument values the answer will be different from the value given in the table, score of task will be 0 points. Executing time for one particular test case must not exceed one second.

Programs with lower **amount of source code in bytes** will get higher scores.

Contestants were also supplied by table values in a separate text file.

The simplest approach was trying to code given values without any investigation. However, much better result could be obtained, by discovering some mathematical regularities.

Besides that, for choosing the right approach, contestant's deep understanding of possibilities of different languages and compilers is a great advantage. It is quite easy to understand usability of such tasks in industry - in a world of microprocessors necessity to fit in a given amount of memory is usual.

Task 5.3. "Twenty-digit number" (finals of 2001)

Construct a twenty digit number using every of digits 0,1,2,3,4,5,6,7,8,9 exactly twice. Leading digit must not be 0.

Scoring:

For any n ($n > 1$) consecutive digits where the first digit is not 0 and the number formed by these digits is square of integer number, you will receive n points. For example, 98543676011023475928 will be scored by 5 points (two for '36' and three for '676').

Task 5.3 can be easily turned into a second kind ("open input") task by specifying different sets of usable digits. However, the essence of task remains the same - contestants are in the same starting positions and still can compete for a better result where possibilities to get perfect result is unclear.

Task 5.4. "Sorting" (finals of 2003, author Aivars Žogla)

In file SORT.PAS an algorithm is given that sorts elements of number array $A[0..n-1]$ from position `low` till position `high` in nondecreasing order. By taking `low = 0` and `high = n - 1`, the entire array will be sorted.

Your task is to find an array containing each of the numbers from 1 to 20 exactly once, for which sorting by calling procedure `sort(A, 0, 19)`, uses maximum number of array element comparison operations (these rows are marked by `{*}`) For example, sorting of the array $A = \{3, 4, 1, 2, 5\}$, uses 7 comparison operations.

```
{ ===== Start of SORT.PAS ===== }
const MAXN = 100;

type TArray = array[0..MAXN] of integer;

procedure swap(var A:TArray; i, j:integer);
var temp:integer;
begin
    temp := A[i]; A[i] := A[j]; A[j] := temp;
end;

procedure sort(var A:TArray; low, high: integer);
var i, j, middle:integer;
begin
    if high - low < 5 then
        for i := low + 1 to high do
            begin
                j := i;
                while j > low do
                    {*} if A[j - 1] > A[j] then
                        begin
                            swap(A, j - 1, j);
                            j := j - 1;
                        end
                    else j := low;
                end
            end
        else
            begin
                middle := (low + high) div 2;
                {*} if A[middle] < A[low] then swap(A, low, middle);
                {*} if A[high] < A[low] then swap(A, low, high);
```

```

{*}      if A[high] < A[middle] then swap(A, middle, high);
        swap(A, middle, high - 1);
        i := low;
        j := high - 1;
        repeat
            repeat
                i := i + 1;
{*}      until A[i] >= A[high - 1];
            repeat
                j := j - 1;
{*}      until A[j] <= A[high - 1];
            if i < j then swap(A, i, j);
            until i >= j;
            swap(A, i, high - 1);
            sort(A, low, i - 1);
            sort(A, i + 1, high);
        end;
end;

begin
end.
{ ===== End of SORT.PAS ===== }

```

No special comments, but if we remember software industry once again, understanding of programs written by another authors also is part of everyday job, isn't it?

7. Conclusions

Technical development of Olympiads in Informatics in the past decade is marked by overall involving of automatic testing and systems allowing program submission together with preliminary testing. However, there are still ways how this development could be continued. Involving contestants in testing, giving more freedom in debugging process, returning tasks with real numbers together with new kinds of tasks may be possible ways of further successful development. The described ideas should be investigated further to determine their usefulness at particular OI levels.

References

- [1] IOI competition rules, <http://olympiads.win.tue.nl/ioi/rules/index.html> (accessed October 28, 2005).
- [2] Automated grading system "OLIMPS", <http://www.lio.lv/olimps> (accessed October 28, 2005, in Latvian).
- [3] Task "Dolāru maiņa", <http://www.lio.lv/olimps/uzdevumi.php?show=300> (accessed October 28, 2005, in Latvian).
- [4] Task "Slaloms", <http://www.lio.lv/olimps/uzdevumi.php?show=288> (accessed October 28, 2005, in Latvian).
- [5] Tasks "Map Generator" and "Map Generator returns". Regional ACM Programming Contest 2005, Minsk, October 26, 2005 (to appear).
- [6] Team competitions in mathematics and informatics "Ugāle", <http://www.uvsk.apollo.lv/p113.htm> (accessed October 28, 2005, in Latvian).