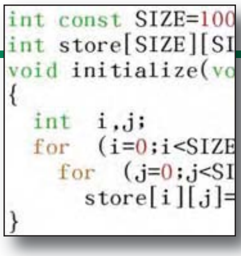
```
int const SIZE=100
int store[SIZE][SI
void initialize(vo
{
  int  i,j;
  for  (i=0;i<SIZE
    for  (j=0;j<SI
      store[i][j]=
}
```

Common Mistakes in Online and Real-time Contests

by Shahriar Manzoor

Introduction

Each year the Association for Computing Machinery (ACM) arranges a worldwide programming contest. This contest has two rounds: the regional contests and the World Final. The teams with the best results in the regional contests advance to the World Final. The contest showcases the best programmers in the world to representatives of large companies who are looking for talent. When practicing for programming competitions, remember that all your efforts should be directed at improving your programming skills. No matter what your performance is in a contest, do not be disappointed. Success in programming contests is affected by factors other than skill, most importantly, adrenaline, luck, and the problem set of the contest. One way of getting immediate feedback on your efforts is to join the Valladolid Online Programming Practice/Contest or the online judge hosted by Ural State University (USU). Successfully solving problems increases your online ranking in the respective competitions.

This article is for beginning programmers who are new to programming contests. I will discuss the common problems faced in contests, the University of Valladolid online judge, and the USU online judge. The suggestions are divided into three parts: General Suggestions, Online Contest Suggestions, and Valladolid-Specific Suggestions. Throughout this paper, please note that in real-time contests, the judges are human and in online contests, the judges are computer programs, unless otherwise noted.

Different Types of Programming Contests

Many programming contests take place throughout the year, such as ACM regional contests, International Olympiad in Informatics (IOI), Centrinës Europos Informatikos Olimpiados (CEOI), and Programmer of the Month (POTM) contest. The most prestigious live programming contest is the ACM International Collegiate Programming Contest (ICPC), and the most prestigious online contest is the Internet Problem Solving Contest (IPSC). In this section, I will discuss some of the contests.

ACM International Collegiate Programming Contest (ICPC)

ICPC, first held in 1977, is now held yearly [4]. The contest lasts five hours and generally contains eight problems. (However, the 2001 World Finals contained nine problems.) Three person teams are allotted a single computer. The teams submit their solutions to a judging software named PC^2 developed at California State University, Sacramento (CSUS). The permitted programming languages are C/C++, Pascal, and Java.

Online Contests

Online contests require no travel and are often less tense [1]. The submission rules for the online contests at the Valladolid site and the USU online judge site are the same: the contestants must mail their solutions to a certain e-mail address. The IPSC rules are quite different. The IPSC Contest Organizer provides inputs for the problems. Instead of e-mailing their solutions, the contestants have to e-mail their outputs.

Some Tips for Contestants

A good team is essential to succeeding in a programming contest. A good programming team must have knowledge of standard algorithms and the ability to find an appropriate algorithm for every problem in the set. Furthermore, teams should be able to code algorithms into a working program and work well together.

The problems presented in programming contests often fall into one of five categories including search, graph theoretic, geometric, dynamic programming, trivial, and non-standard. Search problems usually require implementing breadth-first search or depth-first search. Graph theoretic problems commonly include shortest path, maximum flow, minimum spanning tree, etc. Geometric problems are based on general and computational geometry. Dynamic programming problems are to be solved with tabular methods. Trivial problems include easy problems or problems that can be solved without much knowledge of algorithms, such as prime number related problems. Non-standard problems are those that do not fall into any of these classes, such as simulated annealing, mathematically plotting n-queens, or even problems based on research papers. To learn more about how problems are set in a contest you can read Tom Verhoeff's paper [6].

What You Should Do to Become a Good Team

There is no magic recipe to becoming a good team, however, by observing the points below (some of which were taken from Ernst et al. [3]) you can certainly improve. When training, make sure that every member of the team is proficient in the basics, such as writing procedures, debugging, and compiling. An effective team will have members with specialties so the team as a whole has expertise in search, graph traversal, dynamic programming, and mathematics. All team members should know each other's strengths and weaknesses and communicate effectively with each other. This is important, for deciding which member should solve each problem. Always think about the welfare of the team. Solving problems together can also be helpful. This strategy works when the problem set is hard. This strategy is also good for teams whose aim is to solve one problem very well. On the other hand, the most efficient way to write a program is to write it alone, avoiding extraneous communication and the confusion caused by different programming styles.

As in all competitions, training under circumstances similar to contests is helpful. During the contest make sure you read all the problems and categorize them into easy, medium and hard. Tackling the easiest problems first is usually a good idea. If possible try to view the current standings and find out which problem is being solved the most. If that problem has not yet been solved by your team, try to solve it immediately, odds are it is an easy problem to solve. Furthermore, if the your solution to the easiest problem in the contest is rejected for careless mistakes, it is often a good idea to have another member redo the problem. When the judges reject your solution, try to think about your mistakes before trying to debug. Real-time debugging is the ultimate sin, you do not want to waste too much of your time with a single problem. In a five-hour contest you have 15 person-hours and five computer-hours. Thus, computer-hours are extremely valuable. Try not to let the computer sit idle. One way to keep the computer active is to use the chair in front of the computer only for typing and not for thinking. You can also save computer time by writing your program on paper, analyzing it, and then using the computer. Lastly, it is important to remember that the scoring system of a contest is digital. You do not get any points for a 99%-solved problem. At the end of the contest you may find that you have solved all the problems 90%, and your team is at the bottom of the rank list.

Different Types of Judge Responses

The following are the different types of judge replies that you can encounter in a contest [2]:

Correct

Your program must read input from a file or standard input according to the specification of the contest question. Judges will test your program with their secret input. If your program's output matches the judges' output you will be judged correct.

Incorrect Output

If the output of your program does not match what the judges expect, you will get an incorrect output notification. Generally, incorrect output occurs because you have either misunderstood the problem, missed a trick in the question, did not check the extreme conditions or simply are not experienced enough to solve the problem. Problems often contain tricks that are missed by not reading the problem statement very carefully.

No Output

Your program does not produce an output. Generally this occurs because of a misinterpretation of the input format, or file. For example, there might be a mixup in the input filename, e.g., the judge is giving input from "a.in," but your program is reading input from "b.in." It is also possible that the path specified in your program for the input file is incorrect. The input file is in most cases in the current directory. Errors often occurs because of poor variable type selection or because a runtime error has occurred, but the judge failed to detect it.

Presentation Error

Presentation errors occur when your program produces correct output for the judges' secret data but does not produce it in the correct format. Presentation error is discussed in detail later in this article.

Runtime Error

This error indicates that your program performs an illegal operation when run on judges' input. Some illegal operations include invalid memory references such as accessing outside an array boundary. There are also a number of common mathematical errors such as divide by zero error, overflow, or domain error.

Time Limit Exceeded

In a contest, the judge has a specified time limit for every problem. When your program does not terminate in that specified time limit you get this error. It is possible that you are using an inefficient algorithm, e.g., trying to find the factorial of a large number recursively, or perhaps that you have a bug in your program producing an infinite loop. One common error is for your program to wait for input from the standard input device when the judge is expecting you to take input from files. A related error comes from assuming wrong input data format, e.g., you assume that input will be terminated with a "#" symbol while the judge input terminates with end-of-file.

General Suggestions for Contests

Maximum Memory

The maximum memory allowed on the Valladolid site is 32MB. This includes memory for global variables, the heap, and the stack. Even if you find that you have allocated much less than 64K memory, you will find that the judge often shows that more memory has been allocated. Also, you should not allocate 32 MB of global memory because 32MB is maximum for all types of memory. The maximum memory for real contests varies; for the World Final, it is greater than 128MB.

Problems with DOS Compilers and Memory Allocation

Many of us like to use DOS compilers like Turbo C++ 3.0 and Borland C++, which do not support allocating more than 64K memory at a time. It is always a good idea to allocate memory with a constant so that your test runs use less than 64K memory. Before the submit run, the size of memory can be increased by just changing the value of the constant. If you do not practice this, it is very likely that you will face problems like "Run time error," "Time limit exceeded," and "Wrong answer." An example:

```
int const SIZE = 100;
int store[SIZE][SIZE];
void initialize(void)
{
   int  i,j;
   for  (i = 0; i < SIZE; i++)
      for (j = 0; j < SIZE; j++)
         store[i][j] = 0;
}
```

"Time Limit Exceeded" is Not Always "Time Limit Exceeded"

When you submit a program to the judge, the judge gives you a response, but this response is not always accurate. For example, if you allocate less memory than is required, the program may not terminate (it may not even crash), and the judge will tell you "Time limit exceeded." On seeing this message, if you try to optimize your program rather than correcting the memory allocation problem, your program will never be accepted. The following example illustrates this problem. The skeleton of your program is as follows:

```
#include <stdio.h>
int const MAX = 100;
int array[MAX], i;
void main( void )
{
    for (i = 0; i< =100;i++)
    {
        if (array[i] = =100)
        {
            array[i] = –10000;
            – – – – – –
            – – – – – –
            – – – – – –
        }
    }
}
```

In this example, you have allocated a 100 element array. Your program attempts to access array element 100, which is out of the range [0..99], because of an error in the for loop statement. It will instead access the address of counter variable i. Because the value array[100] is set to –10000, the counter value will be set to –10000, so your loop will take a much longer time to terminate and may not even complete at all. So, the judge will give you the message "Time limit exceeded" even though it actually is a memory allocation error.

Test the Program with Multiple Datasets

There is always a sample input and output provided with each contest question. Inexperienced contestants get excited when one of their programs matches the sample output for the corresponding input, and they think that the problem has been solved. So they submit the problem for judgment without further testing and, in many cases, find they have the wrong answer. Testing only one set of data does not check if the variables of the program are properly initialized because by default all global variables have the value zero (integers = 0, chars = '\x0', floats = 0.0 and pointers = NULL). Even if you use multiple datasets the error may remain untraced if the input datasets are all the same size, in some cases descending in size or ascending in size. So, the size of the dataset sequence should be random. It is always a good idea to write a separate function for initialization.

Take the Input of Floats in Arrays

Consider the following program segment:

```
#include <stdio.h>
float store[100];
void main( void ) {
    int j;
    for (j = 0;j<100;j++)
        scanf( "%f", &store[j]);
}
```

When this program is run, many C/C++ compilers show the error "Floating point format not linked." To get rid of this type of error, just change it to take the input into a normal floating point variable then assign that variable to the array, as follows:

```
#include <stdio.h>
float store[100];
void main( void )
{
    int j;
    float temp;
    for (j = 0;j<100;j++)
    {
        scanf( "%f", &temp);
        store[j] = temp;
    }
}
```

Mark Dettinger's Suggestions on Geometric Problems

Mark Dettinger was the coach for the team from the University of Ulm. He suggested to me that sometimes it is a good idea to avoid geometric problems unless one has prewritten routines. The routines that can be useful are:

- Line intersection.
- Line segment intersection.
- Line and line segment intersection.
- Convex hull.
- If a point is within a polygon.
- From a large number of points what is the number of maximum points on a single line.
- Closest pair problem. Given a set of points you have to find out the closest two points between them.
- Try to learn how to use C's built-in qsort function to sort integers and records.
- Area of a polygon (convex or concave).
- Center-of-gravity of a polygon (convex or concave).
- Minimal circle, a circle with the minimum radius that can include the coordinates for a given number of points.
- Minimal sphere.
- Whether a rectangle fits in another rectangle even with rotation.
- Identify where two circles intersect. If they do not, determine whether one circle is inside another or if they are far away.
- Line clipping algorithms against a rectangle, circle, or ellipse.

Judging the Judge!

Judges often omit information. For example, judges in my country give the error "Time limit exceeded" but never say what the time limit is. In Valladolid, often the input size is not specified (e.g., problem 497-Strategic defense initiative).

Suppose that the maximum number of inputs is not given. This is often vital information because if the number is small, you can use backtracking, and if it is large, you have to use techniques like dynamic programming or backtracking with memorization. In problem 497, the maximum possible number of missiles to intercept is not given. Suppose that the loop `for(j = 0;j<I*100000000;j++)` takes one second to run for the judge, and an unknown N is the number of inputs given by the online judge. Send the following program with your code. Place it just after you have read the value of N.

```
for (I=1;I<=20;I++)
{
   if (I*1000>=N)
   {
      for(j=0;j<I*100000000;j++);
   }
}
```

From the runtime of the program you will know the number of input N. Using this method you can also determine how fast the judge's computer is compared with yours and thus find out the approximate time limit for any problem on your computer. Most of the live contests have a practice session prior to the contest. On this day you should try to determine the speed of the judge computer by sending programs consisting of many loops and nested loops.

Did you know that there was a mistake in a problem of the World Final 2000? The culprit problem was Problem F. The problem specification said that the input graph would be complete but not all inputs by the judge were complete graphs. At least one of the teams sent a program that checked if the input graph was complete. If the input graph was incomplete, then their program entered an infinite loop. So, the response from the judge was "Time limit exceeded." From this response they were able to know that some of the input graphs were not complete and solved the problem accordingly.

Use Double Instead of Float

It is always a good idea to use double instead of float because double gives higher precision and range. Always remember that there is also a data type called a long double. In Unix/Linux C/C++, there is also a long long integer. Sometimes it is specified in the problem statement to use float type. In those cases, use floats.

Advanced Use of printf and scanf

Those who have forgotten the advanced use of printf and scanf, recall the following examples:

```
scanf( "%[ABCDEFGHIJKLMNOPQRSTUVWXYZ]", &line);
//line is a string
```

This scanf function takes only uppercase letters as input to line and any other characters other than A..Z terminates the string. Similarly the following scanf will behave like gets:

```
scanf( "%[^\n]", line);  //line is a string
```

Learn the default terminating characters for scanf. Try to read all the advanced features of scanf and printf. This will help you in the long run.

Using New Line with scanf

If the content of a file (input.txt) is

```
abc
def
```

and the following program is executed to take input from the file:

```
char  input[100], ch;
void  main( void )
{
   freopen( "input.txt", "rb", stdin);
   scanf( "%s", &input);
   scanf( "%c", &ch);
}
```

What will be the value of input and ch?

The following is a slight modification to the code:

```
char  input[100], ch;
void  main( void )
{
   freopen( "input.txt", "rb", stdin);
   scanf( "%s\n", &input);
   scanf( "%c", &ch);
}
```

What will be their value now? The value of ch will be "\n" for the first code and "d" for the second code.

Memorize the Value of Pi

You should always try to remember the value of pi as far as possible, *3.141592653589*79323846264338327 95, certainly the part in *italics*. The judges may not give the value in the question, and if you use values like 22/7 or 3.1416 or 3.142857, then it is very likely that some of the critical judge inputs will cause you to get the wrong answer. You can also get the value of pi as a compiler-defined constant or from the following code: `Pi = 2*acos(0)`.

Problems with Equality of Floating Point (Double or Float) Numbers

You cannot always check the equality of floating point numbers with the == operator in C/C++. Logically their values may be same, but due to precision limit and rounding errors they may differ by some small amount and may be incorrectly deemed unequal by your program. So, to check the equality of two floating point numbers a and b, you may use codes like:

```
if (fabs(a-b)<ERROR) printf( "They are equal\n" );
```

Here, ERROR is a very small floating-point value like 1e-15. Actually, 1e-15 is the default value that the judge solution writers normally use. This value may change if the precision is specified in the problem statement.

The Cunning Judges

Judges always try to make easy problem statements longer to make them look harder and the difficult problem statements shorter to make them look easy. For example, a problem statement can be "Find the common area of two polygons"—the statement is simple, but the solution is very difficult. Another example is "For a given number find two such equal numbers whose multiplication result will be equal to the given number." Though the second statement is longer than the first, the second problem statement is only asking to find the square root of a number, which can be done using a built-in function.

Use the Assert Function

It is always nice to use the C/C++ `assert` function, which is in the header file `assert.h`. With the `assert` function you can check for a predefined value for a variable or an expression at a certain stage of your program. If for some reason the variable or expression does not have the specified value, assert will print an error message. See your C/C++ documentation for further details.

Avoid Recursion

It is almost always a good idea to avoid recursion in programming contests. Recursion takes more time, recursive programs crash more frequently especially in the case of parsing, and, for some people, recursion is harder to debug. But recursion should not be discounted completely, as some problems are very easy to solve recursively (DFS, backtracking), and there are some people who like to think recursively. However, it is a bad habit to solve problems recursively if they can be easily solved iteratively. In live programming contests, there is no point in writing classic code, or code that is compact but often hard to understand and debug. In programming contests, classic code serves only to illustrate the brilliance of the programmer. For example, the code for swapping two values can be written classically as:

```
#define swap(xxx, yyy) (xxx) ^= (yyy) ^= (xxx) ^= (yyy)
```

But in a contest you will not get extra points for this type of code writing.

Improve Your Understanding of Probability and Card Games

Having a good understanding of probability is vital to being a good programmer. If you want to measure your grasp of probability, just solve problem 556 of Valladolid and go through a statistics book on probability. Know about probability theorems, independent and dependent events, and heads/tails probability. You should also be able to solve common card game-related problems.

Be Careful About Using gets and scanf Together

You should also be careful about using `gets` and `scanf` in the same program. Test it with the following scenario. The code is:

```
scanf("%s\n", &dummy);
gets(name);
```

And the input file is:

```
ABCDEF
bbbbbXXX
```

What do you get as the value of name? "`XXX`" or "`bbbbbXXX`" (Here, "`b`" means blank or space.)

Suggestions for UNIX-based Online Judges and Contests

Function Portability

Not all C/C++ functions available in DOS are available in UNIX. Check the documentation for the portability among operating systems. If a function is portable to UNIX, you can use it to solve problems on the Valladolid and USU sites. Use only standard input and output functions for taking inputs and producing outputs.

itoa, the Important Function that UNIX Does Not Have

UNIX does not support the important function itoa, which converts an integer to a string. The replacement for this function can be:

```
char numstr[100];
int num = 1200;
sprintf(numstr, "%d", num);  //to decimal
sprintf(numstr, "%X", num);  //to uppercase hexadecimal
```

Try to find replacements for other functions that are not available in UNIX/LINUX.

Problems with the Settings of Mailer Programs

Some problems do not get accepted even if they are solved correctly. Such problems from Valladolid are 371–Ackermann Function, 336–A node too far, 466–Mirror, mirror, etc. It is because our e-mail programs (e.g., Outlook Express, Eudora) break longer lines, and these problems have long lines in their output. So in Outlook Express you should go to Tools → Options → Send → Send text setting and change the Automatically Wrap Text from 76 (default) to 132. Similar options can be found in other mailer programs. The Ural State University online judge has a program submission form with which you can directly submit your program without sending an e-mail. Remember that problems with mailer settings can cause both wrong answers and compile errors.

Presentation Error

Presentation errors are neither caused by algorithmic nor logical mistakes. There is a difference between the presentation error of online judges and that of live judges. The latter are able to detect mistakes such as misspellings, extra words, extra spaces, etc., and differentiate them from algorithmic errors, such as wrong cost, wrong decisions, etc. These mistakes are the presentation errors as graded by the human judges. On the other hand, online judges in most cases compare the judge output and the contestant output with the help of a file compare program so that even spelling mistakes can cause a "wrong answer." Generally, when the file compare program finds extra new lines, these are considered to be presentation error. Human judges, though, do not typically detect these mistakes. But now computers are becoming more powerful, larger judge inputs are being used and larger output files are being generated. In live contests, special judge programs are being used that can detect presentation errors, multiple correct solutions, etc. We are advancing towards better judging methods and better programming skills. The recent statistics of the ACM shows that participation in the ACM International Collegiate Programming Contest is increasing dramatically, and in the near future the competition in programming contests will be more intense [5]. So the improvement of the judging system is almost a necessity.

A Common Mistake of Contestants

Recently, I arranged several contests with Rezaul Alam Chowdhury and in collaboration with the University of Valladolid, and have seen contestants make careless mistakes. The most prominent mistake is taking things for granted. In a problem I specified that the inputs will be integers (as defined in mathematics) but did not specify the range of input and many contestants assumed that the range will be `0 -> (2^32-1)`. But in reality many large numbers were given as input. The maximum input file size was specified from which one could assume what was the maximum possible number. There were also some negative numbers in the input because integers can be negative.

The Causes of Compile Error

Compile error is a common error on the Valladolid site. It may seem annoying to compile and run a program, then send it to the online judge and get a compile error. Generally these errors occur because contestants omitted `#include` files. Some compilers do not require including the header files even when we use functions under those header files. However, the online judge never allows this. For example, some functions exist both in math.h and stdlib.h. For the online judge, you need to include both of the header files if you want to use them. Compiler errors also occur commonly when contestants do not specify the correct language. Often C code implemented in some compilers inadvertently takes advantage of C++ features. When the language specified to the judge is C, a compile error is generated. For example, the following may be compiled as a C program in a DOS/Windows environment but not in UNIX/LINUX.

```
for (int i = 0;i<100;i++)
{
   printf("Compile Error\n");
}
```

E-mail Sending Format. Mail sent to the online judge should be in plain text format. If the mail is in Rich Text or HTML, the program will not compile. You should not send your program as an attachment.

Mysterious Characters. When I first started programming for Valladolid, I used Turbo C++. After a program was successfully completed, I opened the source code in Notepad, selected the whole text, copied and pasted it in my mail editor, and sent the program to the Valladolid site. I got a Compile error message but could not discover the cause. One day, I pasted it in my email editor, saved it as a text file, and then opened it in my DOS text editor. I discovered some mysterious characters in the file, which were invisible in Windows. If you receive a Compile error message and cannot discover the cause, check if your mail or text editor is adding extra symbols to your code.

Using Nonportable Functions. Compile errors are caused by the use of the functions which are only available in DOS and not in LINUX, such as `strrev`, `itoa`, etc.

Using C++ Style Comments. C++ allows a comment style that starts with //. If the mailer wraps a comment to two lines, you may get a compile error.

Valladolid-specific Suggestions

The next section provides suggestions for solving problems for the Valladolid online judge.

Types of Input in the Valladolid Online Judge

There are four types of input in the online judge:

- Nonmultiple input without special correction program (red flag)
- Nonmultiple input with special correction program (orange flag).
- Multiple input without special correction program (blue flag).
- Multiple input with special correction program (green flag).

What is a Special Correction Program?

There are some problems that have one unique output for a single input, and other problems with multiple output for the same input. For example if you are asked to find the maximum appearing string of length 3 in the string "`abcabcabcijkijkijk`," unfortunately the answer can be both "`abc`" and "`ijk`." So, if your program gives the output "`abc`," it is correct, "`ijk`" is also correct. The judge program cannot determine the correctness of your program by simply comparing your output to the judge program output. The judge must write a special program, which will read your answer and determine if it is right or wrong. This special program is described as a special correction program in the Valladolid online judge. For the problems with special correction programs, (Problem 104, 120, 135, etc., or the problems with orange or green flag), you cannot be sure that your program is incorrect even if your program output does not match the sample output for the given sample input.

"Multiple input programs" are an invention of the online judge. The online judge often uses the problems and data that were first presented in live contests.

Many solutions to problems presented in live contests take a single set of data, give the output for it, and terminate. This does not imply that the judges will give only a single set of data. The judges actually give multiple files as input one after another and compare the corresponding output files with the judge output. However, the Valladolid online judge gives only one file as input. It inserts all the judge inputs into a single file and at the top of that file, it writes how many sets of inputs there are. This number is the same as the number of input files the contest judges used. A blank line now separates each set of data. So the structure of the input file for multiple input program becomes:

```
Integer N   //denoting the number of sets of input
—blank line—
input set 1  //As described in the problem statement
—blank line—
input set 2  //As described in the problem statement
—blank line—
input set 3  //As described in the problem statement
—blank line—
.
.
.
—blank line—
input set n  //As described in the problem statement
—end of file—
```

Note that there should be no blank after the last set of data. Sometimes there may be, so always check. The structure of the output file for a multiple input program becomes:

```
Output for set 1  //As described in the problem statement
—blank line—
Output for set 2  //As described in the problem statement
—blank line—
Output for set 3  //As described in the problem statement
—blank line—
.
.
.
—blank line—
Output for set n  //As described in the problem statement
—end of file—
```

The USU online judge does not have multiple input programs like Valladolid. It prefers to give multiple files as input and sets a time limit for each set of input.

Problems of Multiple Input Programs

There are some issues that you should consider differently for multiple input programs. Even if the input specification says that the input terminates with the end of file (EOF), each set of input is actually terminated by a blank line, except for the last one, which is terminated by the end of file. Also, be careful about the initialization of variables. If they are not properly initialized, your program may work for a single set of data but give incorrect output for multiple sets of data. All global variables are initialized to their corresponding zeros. Thus, for a single set of input, the initialization may not be necessary, but for multiple inputs, it is a must.

The Fixing Mistake Section

Always be sure to see the Fixing Mistake section of the Valladolid online judge. Some of the problems in the Valladolid online judge have errors, which are corrected on this page.

Read the Message Board

Always try to read the message board of the Valladolid site. You will learn many things from other programmers. The USU online judge also has a message board. You can also submit your own views and problems via these boards.

Conclusion

Many people believe that the best programmer is the one with greatest knowledge of algorithms. However, problem-solving skills contribute to programming success as much as raw knowledge of algorithms. Do not lose your nerve during a contest, and always try to perform your best.

Acknowledgements

I am grateful to Prof. Miguel A. Revilla for letting me arrange online contests and to Prof. William B. Poucher for asking people to participate in the World Final Warm-up Contest. I am also grateful to Ciriaco Garcia, Antonio Sanchez, F. P. Najera Cano, Fu Zhaohui, Dr. M. Kaykobad, Rezaul Alam Chowdhury, Munirul Abedin, Tanbir Ahmed, Reuber Guerra, and above all my family.

Useful Links

ACM Home Page: http://www.acm.org/

ACM International Collegiate Programming Contest Problem Set Archive: http://www.acm.inf.ethz.ch/ProblemSetArchive.html

ACM International Collegiate Programming Contest Web Page: http://acm.baylor.edu/acmicpc/

American Computer Science League (ACSL) Homepage: http://www.acsl.org/

Centrinës Europos informatikos olimpiados (CEOI) Resource Page: http://aldona.mii.lt/pms/olimp/tarpt/ceoi.html

Informatics Competitions Link Page: http://olympiads.win.tue.nl/ioi/misc/other.html

Internet Problem Solving Contest (IPSC) Web Page: http://ipsc.ksp.sk/

International Olympiad in Informatics (IOI) Web Page: http://olympiads.win.tue.nl/ioi/index.html

Mark Dettinger's Home Page: http://www.informatik.uni-ulm.de/pm/mitarbeiter/mark/

New POTM Master's Home Page: http://contest.uvarov.ru/

PC2 Home Page: http://www.ecs.csus.edu/pc2/

POTM Master's Home Page: http://members.tripod.com/~POTM/fah_home.html

Ural State University (USU) Problem Set Archive with Online Judge System: http://acm.timus.ru

University Waterloo Contest Page: http://plg.uwaterloo.ca/~acm00/

Valladolid 24-hour Online Judge: http://acm.uva.es/problemset

Valladolid Online Contest Hosting System: http://acm.uva.es/contest

References

1. Astrachan, O., Khera, V., and Kotz, D. 1990. The Duke Internet programming contest. Tech. rep. TR-1998-21, Duke University.
2. Chowdhury, R. A. and Manzoor, S. Orientation: National Computer Programming Contest 2000, Bangladesh National Programming Contest 2000.
3. Ernst, F., Moelands, J., and Pieterse, S. Teamwork in programming contests: 3 * 1 = 4. *Crossroads* 3, 2.
4. Kaykobad, M. Bangladeshi Students in the ACM ICPC and World Championships. *Computer Weekly*.
5. Poucher, W. B. ACM-ICPC 2001, RCD Remarks, RCD Meeting of World Finals 2001.
6. Verhoeff, T. Guidelines for producing a programming-Contest problem set. http://wwwpa.win.tue.nl/wstomv/publications/guidelines.html.

Biography

Shahriar Manzoor (shahriar@neksus.com) is a BSc student of Bangladesh University of Engineering & Technology (BUET). He participated in the 1999 ACM Regional Contest in Dhaka, and his team was ranked third. He is a very successful contest organizer. He has arranged six online contests for the Valladolid online judge including the "World Final Warm-up Contest." His research interests are contests, algorithms, and Web-based applications.

This article originally appeared in *Crossroads* 7.5 (Midsummer 2001), "Tools Tutorial."