**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](http://xrds.acm.org/article.cfm?aid=1375976#links)).

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](http://xrds.acm.org/article.cfm?aid=1375976#links), the maximum possible number of missiles to intercept is not given. Suppose that the loopfor(j=0;j<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 followingscanf 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,*3.14159265358*97932384626433832795, 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, assertwill 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](http://xrds.acm.org/article.cfm?aid=1375976#links) 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.)