# Some Tips for handling different status messages in Code Vita.

## 1. Compile Time Error:

A successful compilation simply returns silently. Hence your aim should be that your program is so agreeable with the compiler that the compiler happily returns silently.

If you get a CTE, do as follows

1. First and foremost you have to ensure that you use the same versions of compilers that the server-side uses. A list of compilers is provided here.
2. If you are using the same compiler, then mentally you have to treat Warnings as errors because warnings prevents the compilers from returning silently. So get rid of all Warnings in your compilation process.
3. If you have meticulously followed the above, it is highly unlikely that you will get CTE from the Code Vita judge
4. In extremely rare case, a negligibly small possibility exists that under heavy load, the servers and hence the compilers malfunction and hence your compilation fails. The probability of this happening is less than 0.01% because Code Vita engineering is now mature enough to handle thousands of concurrent compilations. In your thinking you should simply discount the possibility that the compiler has malfunctioned, but if you have a good enough reason to suspect that this is what might have happened, then simply resubmit the code after some time. This kind of behavior is temporal and the probability of CTE vanishing on its own is pretty high.
5. Finally, whenever a CTE occurs the Code Vita Judge provides the exact error message that the compiler has produced. Using this message as a clue you should be able to troubleshoot past your compilation problems.

## 2. Runtime Error (RTE):

A Runtime error is caused because either your program or the runtime has thrown some exception. Since RTE could be caused because of N number of reasons it is difficult to pin-point and hence provide a crisp error message unlike CTE.

Let us divide RTE into two parts

1) Systemic Faults and
2) Submitted Program Faults.

Systemic Faults: Systemic Faults can give rise to RTE if the Judge is not properly configured to evaluate submissions. In most cases the configurations are on a per-language basis. However systemic faults exhibit binary behavior i.e. either it will work for all submissions in a

given language or it may work for none of the submissions in that given language. It cannot be that some programs in a given language receive RTE and some don't. If this happens to be the case then it is almost always Submitted Program Fault.

It is also possible that one language is configured properly, but some other isn't. Let's say that C is properly configured and Java is not properly configured. In this case all the C submissions will be devoid of Systemic Faults whereas all the Java programs will be susceptible to Systemic Faults. Faults in configuration of one language does not have an impact on behavior of other language. It is the duty and task of Code Vita Engineering team that systemic faults are eliminated before the Judge is thrown open to Code Vita participants. Unlike CTE, RTE is not a transient fault and does not change behavior even under load.

Submitted Program Faults**:** In 99.99% of cases, RTE is caused by Submitted Program Faults. Very rarely, and I don't recollect any instance in past 4 seasons of Code Vita that a Systemic Fault causing RTE has ever been exposed in Live Rounds. Submitted Program Faults could be caused because of any reasons, but not limited to those stated below.

- Failure to adhere to input and output specifications is the number one cause of RTE.
- Known programming violations like Illegal memory access or Null Pointer Exception etc. result in RTE
- RTE are more common to languages like C and C++ where static time type checking is lenient and hence faults manifest only at runtime.
- Any logical mistake that leads to throwing an exception receives an RTE

If you have participated in previous seasons of Code Vita, request you share your stories on how you got past RTEs.

## 3. Time Limit Exceeded (TLE):

In automated code evaluation environments optimal utilization of shared resources such as CPU and memory are key to delivering good performance. However no matter how well a platform is engineered, performance can deteriorate if submitted code is a CPU or Memory hog. So one poorly written program can affect the evaluation times of several others. In such situations platforms, including Code Vita have no choice but to abort the rogue program. The threshold when this behavior kicks in, in Code Vita, is different for different questions.

For example, if a problem is purely compute-intensive i.e. has CPU-affinity then the programs may have a smaller threshold, say 1 second. Likewise, if a program requires lot of memory

accesses to be performed, it may have slightly higher threshold, say 2 seconds. In Code Vita, these limits will be implicit. What this means is that it will not be explicitly spelled out what is the Time Limit for each question. However, an intelligent reader will have already figured out that the moment this status message is received, one has to minimize the runtime of the program.

To cite a contextual example, let's say a question in Code Vita requires you to sort millions of elements. If you implement a naive algorithm like Bubble Sort whose Order complexity is O(n^2) you are almost certain to receive a TLE. A solution to get past TLE, would be to implement a better sorting algorithm, say Quick Sort whose Order complexity is O(n log n). This will drastically reduce the sorting time and the program can finish within thresholds.

Now that we have seen what a TLE is and why it occurs and how system responds to it, let's see some of the ways in which you can overcome TLE

- It is a good practice to insert timestamps in your code to know how much time is spent in different parts of your code. So in case you get a TLE, you already know where your bottlenecks are.
- Your choice of data structure and algorithm plays a critical role in assessing whether you will or will not receive a TLE
- Keep a profiler handy and more importantly know how to use it so that in case of TLE you may get insights on runtime of your code.
- With TLE, some good programmers have a reverse gripe i.e. they write so optimized code that they feel that the thresholds are too lenient. Such programmers are advised to have patience. May be with a few questions, few programmers can get away even with sub-optimal code but it cannot happen always. Keep up the good habit of writing optimized code. There will come a question where only optimal code will pass and rest will receive TLE. It's just that, not all questions are geared towards figuring out if the participant can write optimal code. Also, finally if all others things are equal, and two teams are tied for the last spot, then Code Efficiency which is one of the governing parameter, will kick in and the more efficient code will win. So please don't be too hung up if you have written a (n^2) complexity algorithm and your friend's (n^3) implementation also passes the evaluation.
- So all the best to you all and let there be no more TLEs.
- Sophisticated programmers use many brilliant techniques, like commenting certain sections of the code and figure out the bottlenecks if a TLE status gets converted into Wrong Answer status.

## 4. Memory Limit Exceeded (MLE):

Just like TLE appears due to longer than allowed runtime execution times, MLE occurs due to higher than permissible memory utilization. More memory utilization is a function of two things - the language of your choice and how you handle memory allocation / deallocation in your code.

Memory footprint of languages: Standalone languages like C and C++ have better memory footprint than managed runtime languages like Java and C#. Interpreted languages like Perl, Python, Ruby etc. are somewhere in between. Code Vita systems are 64-bit and we calibrate the footprint of languages before setting memory limits on them. It can be safely said for the kind of problems that are asked in Code Vita, the memory footprint of language runtimes far exceed the amount of memory that a program may possibly need to use to arrive at the correct solution. Hence MLE status is almost always due to poor memory management strategy implemented in the program.

Memory Management in own code: Statically allocating large chunks of memory or speculatively allocating memory based on own understanding of questions is usually the main reason for MLE. Other than that poor data structures and algorithms also cause more than permissible usage of memory. In Code Vita, memory footprint of every process is tracked and rogue processes using more memory are terminated and a status of MLE is returned to the submitter of that program. There are no general rules on how to reduce memory utilization. The rules are language and context dependent. So ensure that you are aware of how to techniques to reduce memory utilization in language of your choice.

## 5. Wrong Answer:

Wrong Answer is caused because your program didn't give the same output as expected for 1 or more test cases.

Whatever questions are being asked in Code Vita goes through thorough testing. So even if your program executes successfully on your system,that does not indicate its correctness. It should pass through all the private test cases of our problem. So before raising any queries, verify your program thoroughly.

## 6. Accepted:

Accepted status comes when your program have passed all the test cases i.e it provided the same output as expected.

If your program shows this status, your problem is solved and you should move on to next problem.

## 7. Presentation Error:

Presentation Error status comes when your program output differs from the expected output by a whitespace character.

Even If your program shows this status, your problem is considered as solved. So don't try to get it in Accepted status and move on to next problem.

## 8. Difference between Accepted and Presentation Error:

Both Accepted and Presentation Error status are same. If either of these two statuses come while evaluating your program, consider it as solved and move on to next problem.

## 9. Public Test Case Vs Private Test Case:

While submitting solutions on CodeVita site, On clicking on ***Compile and Run*** button, The program gets evaluated for public testcases **only** which are given in problem text.

Getting Presentation Error or Accepted status here only means that your program is compiling and executing successfully against public testcases.

Simply executing Compile and Run is not enough . In order to run private testcases against programs one needs to Click on ***Go to Submit page*** button, then click on Submit button to evaluate program.

This submission will be considered for Ranking. This can end up in any of the 8 statuses.

Also, getting Accepted or Presentation Error while Compile and Run does not guarantee that the same status will appear after Submit functionality.

As described above, this is because Compile and Run is run against public testcases while Submit is run against private testcases.

Public testcases are those testcases which are present in problem text whereas private testcases are hidden.