SANDIA REPORT

SAND2005-5610
Unlimited Release
Printed September 2005

compareTestOutput Users Manual

Neal Bierbaum

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

compareTestOutput User Manual

Sandia National Laboratory Version 1.0

The compareTestOutput test utility provides comprehensive evaluation of the output of a test program to determine the success or failure. Its primary methodology is a targeted comparison of the current test's results with those of a known good "reference" test as reported in their text output files. This allows ready migration to new test program versions when a new reference file is created. It may also be used without a reference file with only fixed value comparisons.

Contents

1 Concepts

compareTestOutput is used to determine if a program run has passed or failed based upon the test *program*'s output and return code. The test program's output is called the *result file.* This is compared with the output of a known good run, the *reference file*. *Comparison rules* in the *rules file* define the evaluations to be performed. This is a powerful method that allows easy migration of test evaluation by simple replacement of the reference file when the program is modified or when it is run with a different set of inputs while the rule file remains unchanged. The same rules file may also be used with several different test programs. If the value defined for a comparison is not present in the reference file then no comparison is performed; the rule is ignored. Text which is not checked by rules can vary without limit.

The rules in the rules file define what should be compared between the two files and how it is to be compared. Each rule is on a separate line. Most rules begin with *match expression*, a text field demarcated by double or single quotes, that is used to perform an independent search in the reference file and the result file. The unique line found in each file, the *match lines,* are processed by the rule's action. A very limited number of rule types begin with special keywords that are not in quotes and perform an action that does not use either the result or reference file.

There are several different ways that the rules can be used to compare and validate the result file. The absolute commands use the match expression to search the result file by itself for values which are unconditionally required or prohibited. The other commands allow comparison between the reference and result file lines. In all cases if the line is not in the reference file no comparison is made and no error reported. The simple existence test merely determines if the text exists in both files. The match test determines that a particular line exists in both files and that the

remainder of the line is identical. This is not adequate for programs that return lines containing numerical results which may be expected to have some slight difference in value. A simple text comparison would show a difference in the text and incorrectly indicate failure. To solve this problem compareTestOutput provides rules which compare the numeric results within relative or absolute numerical bounds.

The program may output a number of lines with the same leading text when reporting the results of intermediate computations or multiple versions of the test action with similarly named results. If the program is written specifically for testing then the program can be modified to uniquely identify each line that should be compared. This may be inconvenient or even impossible to change if the output comes from some other external code included in the program or the program itself cannot be modified. The use of*search direction, reference marks,* and*instance counting* solves this problem. The direction of search from the top or bottom of the file, the specification of the range of the search for a match in the files, and the number of matches to be made before performing the comparison allow the rules file writer to select any instance of a line.

2 Shortcuts For Casual Users (and suggestions for others)

This section is for program users who will only rarely create new sets of rules for the program and wish to spend very little time reading the manual. Start with a scan of the sections 1 (Concepts), 3 (Running compareResults), 5 (Creating a Rules FIle), and the first parts of 4 (Writing Rules)and 8 (Compare Actions). You can skip anything marked "**Advanced**". Then look at two of the examples, 9.1 (Basic Example) and 9.2 (Match Example)**.** These two examples will give you a good idea of some of the most commonly used features of compareTestOutput. Take a close look at the "compareTestOutput Output" for each example. That will give you an idea of the way errors are flagged. The text in the output is important for you to see what problem occurred. All your test script needs to look for is a return code of 0 which indicates that your program has passed or a SIGUSR1 (30) that indicates that your test program has failed.

If all you wish to do is catch system errors and look for a single result phrase such as "Pass" 9.1 (Basic Example) is enough to read. Even so, look at 9.2 (Match Example). This will show you the most important ways to use the program. The example should give you some idea of how to redesign your test programs to make them easier to write, make their output much more informative, and catch problems that you might have missed in your internal testing code. Because you are comparing the results against a known good result that uses the same input you can write your test program to take a number of different input sets and not write *any* new evaluation code.Just run the good version of the code with the input set, generate a reference result file, and then use that as your reference file when you run the your program with that input. It is easier for you to write and easier for others to see what your code can do – a win-win situation.

3 Running compareTestOutput

compareTestOutput is run from the command line or within a script in this manner:

```
compareTestOutput options result_filename
Options:
-r rules_filename or --rules-file=rules_filename (required)
-m reference_filename or --reference-file=reference_filename (required)
-t return-code-value or --return-code=return-code-value (optional)
-h or --help
```

If the –help or -h option is included on the command line compareTestOutput will print the help information and immediately terminate with no error.

The rules filename, the reference filename, and the result filename must all be defined. If one or more of these are not defined compareTestOutput will print the help message and immediately terminate with an exit code of SIGERR (1). All files must exist and must be readable by the user. If one or more of these files cannot be read compareTestOutput will print the name of the file and the reason that it cannot be read on both stdout and stderr and then terminate with an exit code of SIGIO (23).

The –return-code or -t option is normally given the return code of the test program, a small integer. This option is not required but must be given for a "LEGAL-RETURN-CODE" rule to be applied.

If compareTestOutput finds that the result file matches the reference file in the manner specified by the rules, the program will print the short message "Test passed." on stdout and return with the normal completion code (0). If compareTestOutput finds that the results do not meet the rules the error(s) will be printed on stdout and the program will exit with SIGUSR1 (30).

(Advanced) The value used with the –return-code option need not be simply a small integer that maps to a defined return code. It can be any string without whitespace. This string is compared with all values defined as "legal return codes". If it does not match one of these values it will be identified as a test failure. This method allows a further dynamic "out of band" way to evaluate the test program's success in some abstract manner determined by the author of the rules file in conjunction with the script that first runs the test program and then compareTestOutput.

For backward compatibility the command line "compareTestOutput *rules_filename reference_filename result_filename*" may be used. This usage is strongly deprecated and should not be used by any new script.

4 Writing Rules

The rules file contains one or more instruction lines. Each line is a separate instruction; no instructions may continue across lines. Each instruction has up to three columns separated by white

space (blanks and tabs). The rules file form allows detailed formatting and commenting so that is can serve as a standalone test document. All whitespace at the beginning of any line is ignored. A line which starts with a "#" is a comment and is not processed. Finally lines which are empty or contain only whitespace are also ignored. This allows commenting, indentation, and separation of commands by blank lines.

All keywords may be written in several forms for ease of understanding and for the preference of the writer. Both lower and upper case letters may be used anywhere in the keyword. Other non-letter-characters such as "-", "_", and "." can be used anywhere in the keyword to assist in reading the words. Only whitespace is prohibited in the keyword. For example, the keyword "legalreturncode" is difficult to read. It could be written "LegalReturnCode", "legal_return_code", "Legal-return-code", "LEGAL.RETURN.CODE" or some other variation can be used, but "legal return code" is prohibited. This document and the accompanying examples all use the keyword form with all caps and dashes between words. This highlights the keywords.

Each column has a specific use. The values in the second and third columns are interpreted based upon the context established by the first column. The third column is often empty; the default value of the rule is used based upon the context of the rule. All rules are tested for correctness. This includes the form of the match expression, the existence of the keyword, and the required number of values. Rules which are incorrect are not used and are identified in the program's output. See 9.3 (Rules Syntax Example).

4.1 Column One: Match Expression / Command Keyword

4.1.1 Match expressions

Match expressions are used to identify the lines from the reference and the result files that will be used for the action defined in the second column. The remainder of the line after the match is used for for comparison if the second column contains either the keyword "matches" or a numerical range expression.

Double_quote_form""""

- The text enclosed within the double quotes matches if it exists anywhere within the line.

Single_quote_form"""

- The text enclosed within the single quotes must be at the start of the line after the whitespace is removed.

4.1.2 Regular Expression Match Expressions (Advanced)

Regular_expression_form"RE(regexp)"

- The text within the () is evaluated as a regular expression. The text is in "raw" form; it does not require any escaping to read correctly. This expression must be written in the exact form shown, i.e., enclosed in double quotes, then the phrase "RE" (in either upper or lower case), then parentheses that enclose the actual text of the regular expression.

Full_regular_expression"FULL-RE(regexp)"

- This expands upon the the regular expression to define not only the value to match but also the value for evaluation. It is written in same manner as the regular expression form but with the phrase "FULL-RE". The expression should return two values, the match text and the text that will be evaluated.

4.1.3 Command keywords

Command keywords are define a top level action for control. They are not enclosed in quotes. They use the second column as a value argument. The second column value is not enclosed in quotes and does contain internal whitespace.

INCLUDE

- Insert lines from another rules file at this point. The second column defines the file name. The include file can only contain tests that are not position dependent: LEGAL-RETURN-CODE, REQUIRED, and PROHIBITED.

LEGAL-RETURN-CODE

- Define an acceptable return code for the test program. This may be defined by either a numeric or a symbolic value as the command argument. The value should not be quoted and cannot contain whitespace. Several acceptable values may be defined by multiple instances of this rule with a different value for each instance. If a rule with this keyword is not defined in the rules file the return value argument is ignored. If no return value argument is included in the command line invoking this run of compareTestOutput the LEGAL-RETURN-CODE rule(s) will be ignored. This is always the first rule which is evaluated.

4.2 Column Two: Control Keyword / Numerical Expression / Command Argument

4.2.1 Control Keywords

Control keywords define specific requirements for the match text in the first column.

REQUIRED

- The text of the first column must be found in the result file without regard to the reference file.

PROHIBITED

- The text of the first column must not exist in the result file.

IN-BOTH

- The text of the first column must exist in the result file only if it exists in the reference file. Other text on the selected lines is ignored.

MATCHES

- The text on the selected lines after the text match expression must be the same in the result file as it is in the text file. For backward compatibility this is the default action if only the first column has a value. New rules files should use this keyword to make the action explicit to other readers.

4.2.2 Mark Control Keywords (Advanced)

SET-TOP-MARK

- Define the top mark.

SET-BOTTOM-MARK

- Define the bottom mark.

CLEAR-TOP-MARK

- Clear the last top mark to revert to the previous top mark. This does nothing if no top marks are set.

CLEAR-BOTTOM-MARK

- Similar action for the bottom mark.

CLEAR-ALL-TOP-MARKS

- Clear all levels of the top mark to set the start of the search range to the beginning of the file.

CLEAR-ALL-BOTTOM-MARKS

- Clear all levels of bottom mark to set the end of the search range to the end of the file.

4.2.3 Numerical Expressions

Numerical expressions are numeric strings that define the range of difference that can exist between the numeric values on the selected line in the reference and result file. The last numeric value on the line is evaluated. There are several means of defining this range. "n" represents any number in these definitions.

n

- A number which does not end with a % sign represents an absolute, non scaling range. This might be used for comparing such things as loop counters but is less flexible for general results, especially those that might vary over a wide range.

n%

- A number which ends with a '%' sign is used to compute the range as a percentage of the reference file's value. For example, if the compare range value is 10% and the reference value is 2000, the range is 200, if the reference value is 2e-6 the range is 2e-7. This is normally the most useful means of comparison because it scales with the reference value.

+-n/n/n%

- A number which begins with neither a '+' or '-' sign or a number which begins with both signs defines a symmetric range that limits both the max and the min. If an asymmetric double sided range is required use two separate rules to define the different upper and lower limits.

+n/+n%

- A number which begins with a '+' sign defines a single ended range. The result file's value can be no larger than the range value above the reference value but may be smaller by any amount.

-n/-n%

- A number which begins with a '-' sign defines the alternate single ended range. The result file's value can be no smaller than the range value below the reference value but may be larger by any amount.

4.2.4 Command Arguments

The command arguments are specific to each command keyword type. See Column One: Command Keywords for usage.

4.3 Column Three: Modifier Keywords / Instance Count

Modifier keywords

Modifier keywords define further action based upon the result of the test.

STOP

- Report failure of this test and process no further actions. This prevents a further listing of errors that are irrelevant based upon this failure. This is especially useful for incorrect return

codes or prohibited keywords like 'SEGFAULT' when anything that follows is certain to be wrong.

CONTINUE

- The default action upon failure of this test. This keyword need not be used; it is available for symmetry with STOP.

FIRST

- The first instance of a line match within the search range in the file. It is the same as an instance count of 1.

LAST

- The last instance of a line match within the search range in the file. It is the same as an instance count of -1.

Instance count (Advanced)

The instance count is the number of match lines found before the control keyword or comparison action is performed. Each match is a single "instance". A positive number indicates the Nth instance down from the current top mark that is still in the marked range. A negative number indicates the Nth instance up from the current bottom mark. The default is 1, the first instance from the top mark.

5 Creating a Rules File

5.1 Initial Design

As you begin to design you rules file consider how your reference file and the result file will differ under various conditions of failure. If there is a single difference that always indicates success or failure you could write a short rules file that checks only that. There may be several different indicators of failure which are in themselves ambiguous but when combined indicate failure. Include a rule to check for each condition. You may wish to include further rules to more precisely identify the failure. This will allow you to look at the output of compareTestOutput and determine not only that there was a failure but the area in which the failure occurred. One example may be a test program that reports not only a summary result but also the results of intermediate tests.

compareTestOutput applies each rule in sequence. Most rules are independent. Their test action does not affect other rules. Since normally all rules will be evaluated the only effect of the order of the rules is the order in which errors will be reported. There are two important exceptions. If a rule has the modifier keyword "STOP" no other rules after this will be evaluated if this rule indicates a failure. The order of rules which create and delete marks is absolutely critical. Setting a mark changes the range of search until it is deleted. Any rules defined between the setting and the deletion of the mark will only be applied against that limited range.

The use of "STOP" in a rule is never critical to the end result of the compareTestOutput run. If the rule indicates failure the test will be reported as failed either with or without the "STOP" in the rule. "STOP" only exists as a convenience to suppress the reporting of further failures once it detects failure. You may not wish to use it during your initial design and test program usage to determine what compareTestOutput reports after that failure is found. You will only want to use it when you can be assured that no further rules after this will generate further useful information. You may also find that moving the rule to a different position within the rules file will allow other useful error reports and suppress those which are not wanted.

5.2 Writing the Rules File

Before you begin writing a rules file create a reference file. Make a copy of this that you will edit for an example result file. After you write the rules file edit your example result file so that compareTestOutput will show a failure for each rule. Leave all other lines in the result file the same. First run compareTestOutput with your rules file and the initial reference file as both reference and result file. compareTestOutput should report no errors. Then make another run with your edited version as the result file. Confirm that each rule reports an error. If a match rule does not work in the manner expected, try changing all possible lines in the result file that could match and run again. You may find that the match has discovered a different line than the one you intended. You will then need to rewrite the rule or, in complicated cases, use marks or instance counts.

5.3 Reusing the Rules File

If you run your test program with different input sets compare the output of each run with that of your original reference file. If there are differences that will be detected by compareTestOutput with your rules file, you will need to create a different reference file for that input set. Use that new reference file when you run with the new input set. If you are performing numeric tests you should not need to change the rules file because all numeric comparisons are made using the value found in the reference file. If there is text or numbers that exists in only in the output from some input sets, the associated rule may still be safely used because it will be applied only if a line that matches the rule is in the reference file.

The same concepts can be used if you modify your test program. Just use the appropriate reference file and you may be able to use both your old and your new program in testing. This might also be true with several different test programs.

6 Command Actions

6.1 Include

The "INCLUDE" command rule reads another rules file and adds its rules at that point. It is analogous to the "#include" in C. This allows commonly used rules that will apply to virtually all tests to be written in a standard rules file that is then included by the rule file specific to the test program. Only rules which do not depend upon ordering, command rules and the absolute match rules (those which use the control keywords PROHIBITED and REQUIRED) may be defined within an include file. Any number of include statements may be used in a rules file. "INCLUDE" is cascading; one file may include another which is included by yet another in turn. Circular includes (ex, A includes b includes C includes A) are terminated at the point of repetition. Further includes of the same file are ignored. See 9.3 (Rules Syntax Example).

6.2 Legal Return Code

The "LEGAL-RETURN-CODE" command rule defines an acceptable integer value for the return code from the program being tested. The "return code" is the value for the command line argument "–return-code" or "-c". That means that the return code value must have been captured at some point prior to the call to compare test. Any value not defined as a legal return code will be identified as a test failure. This rule may be used several times to define multiple acceptable values; repetitions of the same value are ignored. If no legal return codes have been defined compareTestOutput will not perform any checks. Similarly, if no return code value is given on the command line no checks will be performed even if a "LEGAL-RETURN-CODE" rule is defined. See 9.1 (Basic Example).

7 Search Actions

7.1 Match Expressions

Match rules begin with a match expression text surrounded by single or double quotes. compareTestOutput searches the reference or result file for a line that satisfies the match expression. This *match line* is strictly associated with its file. All searches are performed individually in each file. If the match expression (the first column in the rule's line) is in double quotes " " " the match text may be anywhere within a line. If it is enclosed in single quotes " ' " the match text must be the first text on a line after any leading whitespace. (See 9.2 (Match Example) and 9.3 (Rules Syntax Example) .

Example rules:

Line tested in reference or resource file:

```
Does this one match the rule?
```

Double quote rule that does match because the match text "one match" is in the line:

```
"one match" IN-BOTH
```

Single quote rule that does not match because the match text does not begin the line:

```
'one match' IN-BOTH
```

7.2 Search Control (Simple)

The direction of the search, the range of lines within the files to be searched, and the number of matches to be made prior to comparison can all be specified. By default the search direction is forward, from the *top* of the file downward. The first match instance found will be used. This is the simplest and most obvious way to search for a line that can be identified uniquely, for example, "Final test result:". Test programs often report several lines of interim results. The last of these lines quite often contains the end result that is the only one of interest. If this line is the match instance to be found in the file the modifier keyword "LAST" should be used. This changes the search direction to a backward one from the *bottom* of the file upward to the top. These two actions will suffice in most cases.

7.3 Regular Expression Searches (Advanced)

If neither the double nor the single quote expressions can provide the required selection criteria the match expression may be written in the general regular expression form used in Perl or Python. See 9.4 (Regular Expressions Example).

The basic form of a regular expression is defined in the first column with a double quote, the keyword "RE" (upper or lower case), an open parenthesis, the regular expression in raw string form (no \'s required to escape text), then a close parenthesis and a close quote:**"RE(*regular expression*)".** Parentheses within the regular expression need not be escaped. compareTestOutput determines the end of the match expression as a close parenthesis followed by a double quote followed by whitespace; any parentheses inside this are simply used as part of the regular expression.

The regular expression should be defined to search from the start of the line until the end of the portion of the text to be matched. The leading whitespace is removed prior to applying the regular expression. The matched text should not be the entire line. The text remaining after the match is used for the rule's compare action in a manner identical to a normal match.

The full form of regular expression matching is used to define a specific portion of the remaining text on the line that will be used in the compare action. This is most useful for numeric comparisons where the numeric value to be compared is not the last value in the line or when the text comparison for the match should just use a portion of the line which does not include the end. This form uses a regular expression that returns two match values (two regular expression portions each defined

within parentheses). The complete regular expression is used to select a line for the rule. The second value returned by the regular expression is used by the rule's compare action. This form is defined in a manner similar to the basic form but with the keyword FULL-RE , i.e., **"FULL-RE(*regular expression*)"**.

7.4 Search Control (Advanced)

For instances when the above techniques are not adequate, compareTestOutput provides two further complementary techniques. Reference marks are used to define the portion of both the result and the reference files are to be searched. Instance counts define the number of matches that must be made before selecting the line. See 9.5 (Mark Example).

The search is normally performed over the entire file. If the desired match cannot be readily selected by this global search then marks can be set to limit the range of the search. A top mark defines the starting point of the search portion. The mark is set by a match rule with the action keyword "SET-TOP-MARKER". The mark is set on the matched line. A bottom mark defines the end of search portion. This is set by a match rule with the action keyword "SET-BOTTOM-MARK". The line that a mark is on is not included in the search. If no match line can be found no mark is set. Top and bottom marks are controlled independently, but a top mark must leave at least one line above the bottom mark. The position of each mark is nested; a new top or bottom mark is set within the search space defined by current mark position to more tightly restrict the search space. The mark position reverts to the previous position each time the mark is cleared. The match expression used for the clear commands does not really matter; there is no actual search performed when performing a clear action. "CLEAR-TOP-MARK" deletes the innermost top marker. "CLEAR-BOTTOM-MARK" deletes the innermost bottom mark. "CLEAR-ALL-TOP-MARKS" deletes all top marks; the top limit of the search space reverts to the beginning of the file. "CLEAR-ALL-BOTTOM-MARKS" deletes all bottom marks; the bottom of the search space reverts to the end of the file.

The instance count defines the direction of search and the number of the match to be used. A count of 1 with no marks set is the same as the most simple default action; it searches for the first match instance in the file. The default for search is, in fact, an instance count of 1 so an empty instance count acts the same way. A instance count of 2 selects the second match instance. An instance count of -2 searches backward for the second match from the bottom. If there are fewer match instances than the absolute value of the instance count, no match is reported. A match rule with an instance count of 0 has no meaning. It is reported as a rule syntax error and ignored. All instance count searches operate within the range set by the innermost top and bottom marks. For example, if a bottom mark is set a match rule with an instance count of -1 would use the first match found in a backward search from the bottom mark. The modifier keyword "FIRST" is a synonym for a match count of 1 and "LAST" is really a synonym for -1. These keywords are included to make the rules more understandable.

8 Compare Actions

8.1 Absolute

Two compare rules depend only upon the text in the result file. These are identified with the action keywords "REQUIRED" and "PROHIBITED". A match rule with the keyword "REQUIRED" means that the match text must exist within the result file. This can be useful for a test that performs internal evaluation of success or failure and reports success with some special phrase, such as "passed". A match rule with the keyword "PROHIBITED" means that the match text must not exist in the file. This is most useful to detect when some system error has occurred. This is often reported with a phrase like "SIGSEGV ...". See 9.1 (Basic Example). The match text should be chosen with some care to assure that it does not inadvertently identify text that isn't really meant to match but at the same time casts a wide enough net to discover all instances that should be matched. If these two types of match rules are the only ones contained within a rules file then the contents of the reference file are unimportant. The reference file must exist, but it can be an empty file.

8.2 Simple Existence

A compare rule with the action keyword "IN-BOTH" searches for the match text in both the reference and the result file. If the text is found in the reference file then it must also exist in the result file. If it is not in the reference file then it need not be in the result file. This is useful when the test program might put out different phrases to indicate success depending upon the input or if the same rules file might be used with several different programs. See 9.2 (Match Example: IN-BOTH rules).

8.3 Text Comparison

A compare rule with the action keyword "MATCHES" compares the text remaining on the files matched line after the match text.

The remaining text in the result file must be identical with that in the reference file. Leading whitespace in the remaining text is ignored. See 9.2 (Match Example – MATCHES).

For example:

line in reference file:

```
First series result: in limit
```

line in result file:

```
First series result: not in limit
```

Rule:

```
'First series result:'   MATCHES
```

This will fail because the text "not in limit" is not identical to "in limit".

8.4 Numerical Comparison

A compare rule with a numerical expression in the second column defines a numerical comparison. This form of a compare action is a truly distinguishing capability of the compareTestOutput program. It is critical to the evaluation of programs that produce numerical results where the values are expected to be quite similar but not exactly the same. This allows comparisons that heretofore had to be performed within the test program itself because no simple text matching of output was meaningful. This also makes test design far simpler because the comparison is made against the matching reference file value so that no internal rules need be defined to handle each of several different inputs. The known good program version need only be run with each set of input values to generate an associated reference file. These reference files are then used when comparing the results computed by a later version of the code with the associated input.

A numerical comparison rule compares the value of the last numerical expression on the selected line in the result file with that in the reference file. Intervening text and other numerical values are ignored as is following text. The value in the rule's second column defines the acceptable range for the result value with respect to that value in the reference value. Any type of range may expressed with these rules. See 9.2 (Match Example – Numerical Match).

A range that is a fraction of the reference file's value is defined by a number with a % following. An absolute range is a number without a % sign. If there is no leading sign the range is symmetrical. A leading '+' or '-' sign defines a single sided range. A minus sign defines the relative minimum value with no upper range limit. A plus sign defines a relative maximum value with no lower range limit. An asymmetric range can be defined with two rules, one which defines a minimum and another which defines a maximum. There can be no whitespace in the numerical expression.

For example, these values in the second column mean:

1.25% – the result value cannot be greater than (reference value + (0.0125 * reference value)) nor less than (reference value - (0.0125 * reference value))

2.0e2 – the result value cannot be greater than (reference value + 200) nor less than (reference value - 200)

-2 – the result value cannot be less than (reference value - 2)

+4.5% – the result value cannot be greater than (reference value + (0.045 * reference value))

9 Examples

The examples in this directory show the use of all important features in the compareTestOutput program. The examples generally build upon each other so they should be examined in the order presented. For each rule look first at the rule, then at the lines selected by the rule in the reference and result files, and then finally at the result of the rule in the compareTestOutput output. All rules that have been successful have no text in the output file; only those rules which indicate a failure in the test program have an entry.

The example rules files are heavily commented to serve as stand-alone training units. Keywords are written in all caps to clearly flag their existence. The reference and result files also have some comments but much of the information is within the lines that are actually operated upon. This means that the files often do not really resemble an actual test output. None of the text in these two files is special; it is made important only when used by the match rules defined in the rules file. There is no need for any explicit comment flags in the reference or result files; they are not parsed.

All of the examples are available as separate files in the accompanying examples directory and are ready to run. The filenames are a concatenation of the example name, the phrase "Example" and the file's use, "RulesFile", "ReferenceFile", or "ResultFile". The easiest way to try an example, for instance the Match Example, is to just change directory to the examples directory and then type "*compareTestOutput -r MatchExampleRulesFile -m MatchExampleReferenceFile MatchExampleResultFile*".

9.1 Basic Example

This is an example of a simple rules file that can be used with a test program that returns a phrase with text "pass" somewhere in it. There are also rules to evaluate the return code and to check for major runtime errors. It uses only absolute rules so it can be used with an empty reference file. The reference file used here is "EmptyFile", a zero length file.

The example uses two different result files, one which passes and one which fails. The examples were run with the additional command line argument "–return-code", the value of the return code of the program being tested. The successful one had a value of 0, the failed one had a value of 11 from the SIGSEGV error.

The compareTestOutput output from the successful test is exactly as will be seen with any compareTestOutput run that does not find any errors. All examples after this basic one will find errors in the result file; only this example has a run that passes.

9.1.1 Rules File

```
# This is a very simple file that checks for fatal errors and
# the single word "pass" somewhere in the results.
# It does not really compare anything in the reference file
```

```
# so the reference file may be empty.

    # A return code of 0 is allowed from the test. Any other return code
    # will indicate failure.

LEGAL-RETURN-CODE  0

    # If the character sequence "SIG" (as in SIGSEGV, SIGKILL, etc) is found
    # in the result file report failure. Stop further processing because the
    # remainder is probably in error. The "STOP" modifier keyword is here
    # only as example -- it need not be included if you wish processing to
    # continue

"SIG"           PROHIBITED    STOP

    # Somewhere in the file there must be the phrase "pass"

"pass"          REQUIRED
```

9.1.2 Result File (passes)

```
This result file is successful when used with the SimpleExamplesRulesFiles
The test passed!!!
```

9.1.3 compareTestOutput Output

```
Test Passed.
```

9.1.4 Result File (fails)

```
This result file shows errors when used with the SimpleExamplesRulesFiles
interrupt SIGSEGV: 11
test failed
```

9.1.5 compareTestOutput Output

```
------------------------------------------------------------
---------- Comparison test found test failure(s) -----------
------------------------------------------------------------
```

```
The test program return code '11' is not legal.
=====================================================
The prohibited text "SIG" was found in the results file  line 3:
    interrupt SIGSEGV: 11
====Further comparisons stopped upon this failure====
compareTestOutput found test program failure.
```

9.2 Match Example

This example shows how match expressions are formed and used for comparison of critical values in the reference and the result files. The use of numerical comparison is especially emphasized. After review of this example users will be able to create most rules files.

9.2.1 Rules File

```
# This rules file demonstrates the use of the include command
# and the various forms and actions of comparison.
INCLUDE         ./StandardIncludeRules
    # The simplest forms of search. These use only the results file.
    # The keywords here have two variations in capitalization of the keyword form.
    # Any combination of upper and lower case letters are acceptable.
    # Of course the case is critical in the match expressions.

"Pass"                  REQUIRED
"Total Failure"         PROHIBITED

    #All other match actions search the reference file first. If the match is not
    #found in the reference file no error is reported and no search is
    #performed on the result file.

    #This match will not be found in the reference file. Nothing will be reported.

"not found"             IN-BOTH

#   #This match will will be found in the reference but not in the result file.
    #An error will be reported.

"only in reference"             IN-BOTH

    # This match will be found in both files. The remainder of the line is different
```

```
    # but this is OK with the test performed with the keyword "IN-BOTH". This simply
    # tests that the phrase is in both files

"in both files"                 IN-BOTH

    #This uses the single "'" match so it must be at the very start of the line after
    #whitespace. It is at the start in the reference file but not in the result file.
    #An error will be reported.

'at start'                      IN-BOTH

    # Now perform a match using the keyword "MATCHES". This means that everything
    # after the match text must be identical in both files. The first will succeed.

 "This matches"                 MATCHES

    # This match again uses the keyword "MATCHES" but the match text is at
    # different places in the files' lines. This will still succeed because
    # the text remaining after the match text is the same.

"end matches:"                  MATCHES

     #This will fail; the remaining text is different.

"end doesn't match:"            MATCHES

     #Now begin the numerical comparisons. The first is probably the most common.
     #The absolute value of the difference between the reference and the result
     #files must be no greater than the stated fraction of the reference value.
     #The first will succeed, the second fail. Note that all text between the
     #match text and the final numeric expression will be ignored in the comparison.

 'Close enough'                 5%
 'Too different'                5%

     #A number without a percent sign '%' defines an fixed limit. This can be
     #dangerous when the reference value may change in magnitude. The first match
     #will fail, the second will succeed. Probably neither is what the test author
     #intended.

"Fixed comparison that fails"           0.9
```

```
"Fixed comparison that succeeds"        0.9

    #Signed values define a one sided comparison. A '+' sign establishes an upper
    #limit. This means that the result value can be only the defined amount
    #greater than the reference value but can be any amount smaller.

"upper limit that fails"                +10%
"first upper limit that succeeds"       +10%
"second upper limit that succeeds"      +10%

    #A '-' sign defines a lower bound. These examples use a fixed limit merely to
    #show the use.

"Lower limit that fails"                -5
"Lower limit that succeeds"             -5

   #Only the last numerical expression is used in the comparison.
   #The first compare will fail, the second succeed.
"fail ["                10%
"succeed ["             10%

   #If the reference file doesn't have a number then a numeric test becomes simply
   #an "IN-BOTH" test.

'No reference number'           1%

   #If the result file doesn't have a number but the reference file does the
   #compare will fail.

'No result number'      1%
```

9.2.2 Reference File

```
This is the reference file for the Match Example.
These lines at the top of the file don't match
any of the rules so they are ignored. The lines
here are generally in the same order as the rules
file just to make the example easier to understand.
Further instruction only text will be marked with
"><" just to make it more obvious.
>Lines used for text matches<
Line only in reference file
```

```
This is in both files - but the end is different
>note that the "start" of a line is after all whitespace<
          at start in reference file
This matches in both files.
The start is different, but the end matches: the same text
The start is the same, but the end doesn't match: reference
>Lines used for numeric comparison tests<
Close enough (text until number ignored) 1e+6 (text after number ignored)
Too different...  100
Fixed comparison that fails, even though close in value: 10000
Fixed comparison that succeeds: 0.01
The upper limit that fails: 100
The first upper limit that succeeds: 100
The second upper limit that succeeds: 100
Lower limit that fails: 100
Lower limit that succeeds: 100
Try fail [27].2         100
Try succeed [35].9              100
No reference number:
No result number: 100
```

9.2.3 Result File

```
This is the result file for the Match Example.
Further instruction only text will be marked with
"><" just to make it more obvious.
>Here are the phrases for both of the absolute tests<

Oh No, Total Failure!!!
Test didn't pass. Too bad, "pass" is all lower case.

>Lines used for text matches<
This is in both files - but they aren't the same...
        not at start in result file
   This matches in both files.
The start is different, but the end matches: the same text
The start is the same, but the end doesn't match: result
>Lines used for numeric comparison tests<
Close enough:   1000050
Too different...        91
Fixed comparison that fails, even though close in value-- 10001
```

```
Fixed comparison that succeeds: -0.8
The upper limit that fails: 111
The first upper limit that succeeds: 108
The second upper limit that succeeds: 30
Lower limit that fails: 94
Lower limit that succeeds: 400
Try fail [27] .2                111
Try succeed [90] 8.7            105
No reference number: 100
No result number:
```

9.2.4 compareTestOutput Output

```
------------------------------------------------------------
---------- Comparison test found test failure(s) ------------
------------------------------------------------------------
The required text "Pass" could not be found in the result file.
=====================================================
The prohibited text "Total Failure" was found in the results file  line 6:
    Oh No, Total Failure!!!
=====================================================
The rule:
    "only in reference"         IN-BOTH
Found the line in the reference file, line 8:
    Line only in reference file
but no line could be found in the result file.
=====================================================
The rule:
    'at start'                  IN-BOTH
Found the line in the reference file, line 10:
    at start in reference file
but no line could be found in the result file.
=====================================================
The text compare rule:
    "end doesn't match:"                MATCHES
  found lines:
    Reference( 21): The start is the same, but the end doesn't match: reference
    Result( 21): The start is the same, but the end doesn't match: result
  These lines do not match closely enough.
=====================================================
The numeric value compare rule:
    'Too different'             5%
```

```
  found lines:
    Reference( 26): Too different...  100
    Result( 26): Too different...       91
  Numerical comparison determined that the test program's value was too small:
        Reference: 100.0
        Result: 91.0
=====================================================
The numeric value compare rule:
    "Fixed comparison that fails"               0.9
  found lines:
    Reference( 28): Fixed comparison that fails, even though close in value: 10000
    Result( 28): Fixed comparison that fails, even though close in value: 10001
  Numerical comparison determined that the test program's value was too large:
        Reference: 10000.0
        Result: 10001.0
=====================================================
The numeric value compare rule:
    "upper limit that fails"            +10%
  found lines:
    Reference( 31): The upper limit that fails: 100
    Result( 31): The upper limit that fails: 111
  Numerical comparison determined that the test program's value was too large:
        Reference: 100.0
        Result: 111.0
=====================================================
The numeric value compare rule:
    "Lower limit that fails"            -5
  found lines:
    Reference( 35): Lower limit that fails:     100
    Result( 35): Lower limit that fails:        94
  Numerical comparison determined that the test program's value was too small:
        Reference: 100.0
        Result: 94.0
=====================================================
The numeric value compare rule:
    "fail ["            10%
  found lines:
    Reference( 38): Try fail [27].2             100
    Result( 38): Try fail [27] .2               111
  Numerical comparison determined that the test program's value was too large:
        Reference: 100.0
```

```
        Result: 111.0
=====================================================
The numeric value compare rule:
    'No result number'  1%
  found lines:
    Reference( 43): No result number: 100
    Result( 43): No result number
  No numeric value could be found in the result
=====================================================
The numeric value compare rule:
    'Iteration'         2%
  found lines:
    Reference( 48): Iteration [1]               17.6
    Result( 46): Iteration [1]          14.1
  Numerical comparison determined that the test program's value was too small:
        Reference: 17.6
        Result: 14.1
=====================================================
The numeric value compare rule:
    'Iteration'         2%              LAST
  found lines:
    Reference( 51): Iteration [86]              35.55
    Result( 48): Iteration [90]         31.7
  Numerical comparison determined that the test program's value was too small:
        Reference: 35.55
        Result: 31.7
=====================================================
compareTestOutput found test program failure.
```

9.3 Rules Syntax Example

This example shows the basic syntax of the rules. It does not fully explain what the rules mean, that is done in the manual and in later examples.

9.3.1 Rules File

```
# This rules file just demonstrates some of the general rules file syntax.
# Several rules have intentional errors to show how they are detected.
# All errors in rules are nonfatal but they are reported and the compareTestOutput
# test reports an error.
```

```
# Commands are in the first column and are not surrounded by quotes. They
# always take an argument in the second column.
    # INCLUDE loads another rules file. It is a fatal error if the file does
    # not exist. The filename is not in quotes
    # This is good
INCLUDE         ./ExampleIncludeRules

    # This is bad; it has no argument.
INCLUDE
    # This is bad; the file does not exist.
INCLUDE         ./nofile
    #LEGAL-RETURN-CODE expects an integer argument
LEGAL-RETURN-CODE       0
    #LEGAL-RETURN-CODE can accept the third column modifier keyword "STOP"
LEGAL-RETURN-CODE       0       STOP
# All command, action, and modifier keywords can be expressed with any
# combination of upper and lowercase with other non-alphabetic characters except
# whitespace between the keywords characters.
# This lets you write complex keywords as desired. Leading whitespace is
# allowed in rules and any nonzero amount of whitespace may separate columns.
LegalReturnCode         0
  legalreturncode 0
        legal-return-code               0
LEGAL_RETURN_CODE       0
<legal*returN*CoDe>             0
    #These fail because whitespace, abbreviations, and undefined keywords are
    #all errors
Legal return code       0
LGL_RTN_CODE            0
MyBadCommand            0
# Most rules are match rules. Single or double quotes surround the match
# expression in the first column. The type of quote is significant.
# The second column contains an action keyword or a numeric expression.
# The optional third column may contain a modifier keyword or an integer
# All details are in other example files. The examples here only show
# general syntax.
    #These are good.

"end result:"   +5%
'Solution within bounds'        IN-BOTH         LAST
"Run was"       MATCHES
```

```
"Failed"        PROHIBITED      STOP

   # These are not.

        # Anything without quotes is interpreted as a command.
Failed          PROHIBITED
        # Quotes must be closed.
"Failed         PROHIBITED

        # Any word in the second column of a match expression must be an
        # keyword
"Failed"        UNKNOWN-KEYWORD
    # For backward compatibility, a match expression with only one
    # column acts the same as a match expression with the keyword
    # "Matches". This form is deprecated and should not be used in
    # any new scripts.
"Result was"

# Regular expressions are defined by the match phrase "RE" or "FULL-RE" and
# the expression enclosed in parentheses. The details are in examples.
"RE(\D*\d*)"    IN-BOTH
"FULL-RE((\D*)\s*(\d*))"        +2%
   # The regular expression match phrase must have parentheses around the regular
   # expression. If it does not it is used as a simple text match.
   # This error can only be detected with the "REQUIRED" test. The matching text
   # will never be found in either file. Thus any other test (PROHIBITED, IN-BOTH,
   # MATCHES, numeric, etc.) will never report an error.
   # This test fails -- the desired action
"RE(\D*\d*"     REQUIRED

   # These tests always pass.
"RE(\D*\d*"     PROHIBITED
"RE(\D*\d*"     IN-BOTH
```

9.3.2 Include Rules File

```
# This is a rules file that identifies common fatal errors that most tests
# should be compared against. It can be used by adding the rule
# "INCLUDE <file-path>/StandardIncludeRules" where <file-path> is either the
# relative or absolute path to the file
    # A return code of 0 is allowed from the test. Any other return code will
    # indicate failure
```

```
LEGAL-RETURN-CODE   0
    # If the character sequence SIG (as in SIGSEGV, SIGKILL, etc) is found in
    # the result file report failure and stop further processing.
    # The remainder of the test's result is probably in error.
"SIG"             PROHIBITED    STOP
```

9.3.3 Reference File

```
This is a very simple file that contains only lines enough to
match the few successful rules in the RulesSyntaxExampleRulesFile
Test end result:        10.05
Run was successful
Run77
Example loop [110]: 99
```

9.3.4 Result File

```
This is a very simple file that contains only lines enough to
match the few successful rules in the RulesSyntaxExampleRulesFile
Test end result:        10.1
Run was successful
Run95
Example loop [111]: 99
```

9.3.5 CompareResults Output

```
-----------------------------------------------------
---------- Error in loading include file. ----------
-----------------------------------------------------
    The include file '/test/compareTestOutput/examples/nofile' could not be read:
        [Errno 2] No such file or directory: '/test/compareTestOutput/examples/nofile'
-----------------------------------------------------
---------- Error in rule line(s) ----------
-----------------------------------------------------
    Rule: 'INCLUDE'
    Reason: Rule arguments missing.
=====================================================
    Rule: 'Legal return code    0'
    Reason: Unknown command keyword : Legal
=====================================================
    Rule: 'LGL_RTN_CODE         0'
```

```
    Reason: Unknown command keyword : LGL_RTN_CODE
=====================================================
    Rule: 'MyBadCommand         0'
    Reason: Unknown command keyword : MyBadCommand
=====================================================
    Rule: 'Failed               PROHIBITED'
    Reason: Unknown command keyword : Failed
=====================================================
    Rule: '"Failed              PROHIBITED'
    Reason: End quote missing
=====================================================
    Rule: '"Failed"     UNKNOWN-KEYWORD'
    Reason: Unknown control keyword : UNKNOWN-KEYWORD
=====================================================
-------------------------------------------------------------
---------- Comparison test found test failure(s) ------------
-------------------------------------------------------------
The required text "RE(\D*\d*" could not be found in the result file.
=====================================================
compareTestOutput could not load an include file.
compareTestOutput found rule file parse errors.
compareTestOutput found test program failure.
```

9.4 Regular Expressions Example (Advanced)

compareTestOutput allows the match text to be defined by a regular expression. The regular expressions can be further expanded to precisely select the portion of the matched line that will be used in the comparison action. The use of regular expressions allows virtually any distinct text to be identified and processed. The regular expressions used in the compareTestOutput program are in the form used by Python and Perl.

Regular expressions can be difficult to create and debug. This example is only for those who are familiar with regular expressions.

9.4.1 Rules File

```
# This rule file demonstrates the use of regular expressions for
# generalized match and for explicit definition of the compare
# values. The examples are useful for those who have a good
```

```
# understanding of regular expressions but will be a mystery for
# others.
# Some examples are rather contrived...
# The first set of examples uses the series of lines starting "Result, loop"
# as might be found in an iterative solver. The line without brackets has a
# textual summary, the line with an alphabetical index is a summary.
    # This shows the use of regular expression to search for a line
    # that does not have some text.

"RE(loop\s*^\[)"         MATCHES

    # The second seeks a line with one or more letters as an index

"RE(loop\s*\[[A-Za-z]\])"       5%

# The next examples demonstrate the use of regular expressions to
# control what is evaluated.
   # This pair of examples demonstrate a regular expression to create a
   # match rule that evaluates just the end of the line. Both rules evaluate
   # same line
        # The first rule uses just a normal match expression and doesn't work
        # as desired. It fails because a portion after the match text is
        # different.

"Compare end only"              MATCHES

        #This regular expression rule consumes all of the intervening text
        #so only the last part is compared. It succeeds.

"RE(Compare end only.*:)"       MATCHES

    # The full regular expression option not only performs the match to select
    # the line but also selects the portion to be used for the selected action.
    # In this example only the middle portion of the text is to be matched.
    # The first attempt with a simple regular expression fails because it
    # compares all the way to the end.

"RE(Compare middle only.*:)"    MATCHES

   # This regular expression performs a match with the first portion of
   # the regular expression then selects the portion to be compared and
```

```
   # returns it as the second match value

"FULL-RE((Compare middle only).*:(.*)\s*\()"    MATCHES

   # Finally a full regular expression must be used to compare a numerical
   # value before the last in the line. In this example the compare needs to
   # performed on the number in brackets (a count of loops, maybe)

   # A normal compare uses the last value and succeeds

"Example loop"          +5

   # The full regular expression selects the value inside the square brackets
   # for comparison. It fails.

"FULL-RE((Example loop)\s*\[(\d+)\])"   +5
```

9.4.2 Reference File

```
This is the reference file for the regular expression example.
These lines at the top of the file don't match
any of the rules so they are ignored. The lines
here are generally in the same order as the rules
file just to make the example easier to understand.
Result, loop [1]: 10
Result, loop [3]: 30
Result, loop [A]: 100
Result, loop    success
Compare end only <ref file version>: Good match
Compare middle only <ref>: Good match (reference file)
Example loop [100]: 99
```

9.4.3 Result File

```
This is the reference file for the regular expression example.
These lines at the top of the file don't match
any of the rules so they are ignored. The lines
here are generally in the same order as the rules
file just to make the example easier to understand.
Result, loop [1]: 10
Result, loop [3]: 30
```

```
Result, loop [A]: 110
Result, loop    failure
Compare end only <result file version>: Good match
Compare middle only <res>: Good match (result file)
Example loop [110]: 99
```

9.4.4 compareResult Output

```
-------------------------------------------------------------
---------- Comparison test found test failure(s) ------------
-------------------------------------------------------------
The numeric value compare rule:
    "RE(loop\s*\[[A-Za-z]\])"   5%
  found lines:
    Reference(  9): Result, loop [A]: 100
    Result( 10): Result, loop [A]: 110
  Numerical comparison determined that the test program's value was too large:
        Reference: 100.0
        Result: 110.0
=====================================================
The text compare rule:
    "Compare end only"          MATCHES
  found lines:
    Reference( 13): Compare end only <ref file version>: Good match
    Result( 14): Compare end only <result file version>: Good match
  These lines do not match closely enough.
=====================================================
The text compare rule:
    "RE(Compare middle only.*:)"        MATCHES
  found lines:
    Reference( 15): Compare middle only <ref>: Good match (reference file)
    Result( 16): Compare middle only <res>: Good match (result file)
  These lines do not match closely enough.
=====================================================
The numeric value compare rule:
    "FULL-RE((Example loop)\s*\[(\d+)\])"       +5
  found lines:
    Reference( 17): Example loop [100]: 99
    Result( 18): Example loop [110]: 99
  Numerical comparison determined that the test program's value was too large:
        Reference: 100.0
```

```
        Result: 110.0
=====================================================
compareTestOutput found test program failure.
```

9.5 Mark Example (Advanced)

None of the techniques for search in the previous examples allow one to select a specific line for comparison if there are several lines with the same text. This is a fairly common occurrence for programs which report intermediate results or which perform several different tests with the same result phrase. This example demonstrates the use of the search direction, marks to limit the search range within the text, and offset count to select a specific instance other than the first or last.

The first technique, search direction, is simple to use and the most common. Many times it is merely the last value of an iteration series that is of interest. That is the first rule in this example. The second technique, using marks, is demonstrated by the next several rules. The third technique, instance counting, is demonstrated in the final rules.

The example uses almost all numerical matches. This allows further text at the end of each match to readily identify which lines are selected by each rule. The sequence of several of the rules is quite important. The rules which use marks build upon the result of the previous rules.

9.5.1 Rules File

```
# This is the example rules file to show how search direction,
# markers, and index counting works. The position after the rule
# has been performed is identified by the text within "<>".
    # Read last "Final" (the real end one) <match A>
"Final" 1%      LAST
    # Set first top marker to assure that we really are in
    # the summary results <top marker A>
"Summary"       SET-TOP-MARK
    # Now set the top mark just above the write test result<top marker B>
"Write"         SET-TOP-MARK
    # Read the Write Test final result <match B>
"Final"         1%
    # Set the top mark just above the compute test result <top marker C>
"Compute"       SET-TOP-MARK
    # Read the Compute Test final result
 "Final"        1%
    # Remove all top marks so that a bottom marker can be set higher than
    # the top marks. Note that the match expression isn't really used

"unused"        CLEAR-ALL-TOP-MARKS
```

```
    # Set bottom mark to be able to read the lines at the end of the
    # Write test <bottom marker A>
    # When setting the bottom mark the search proceeds from the bottom
    # so we need to get the second instance from the bottom

"Compute Test"  SET-BOTTOM-MARK -2
    #Read the last instance of loop in the write test, the first one
    # above "Compute Test <match D>

"loop"          1%      -1
    # Set the top mark to be able to read the first instance of the
    # write test loop. Note that the bottom marker did not need to
    # be removed first. <top marker D>

"Write"         SET-TOP-MARK
    # Read the first instance of loop in write test <match E> with
    # an empty third column for the default action.

"loop"          1%
    # Read the second instance of loop in write test <match f> with
    # an instance count of 2

"loop"          1%      2
    # Attempt to read Final in the "Compute Test" block,
    # the second instance of final after the top mark, to compare text.
    # This will find nothing because the line is below the bottom
    # mark.
"Final"         Matches 2
    # Remove bottom mark to try again
"unused"        CLEAR-BOTTOM-MARK
     # Now retry the read for "Final"
"Final"         Matches 2
```

9.5.2 Reference File

```
This is the reference file for the search direction, mark, and
instance count example.
The text in the <>'s are comments for the example. They do not
affect the results of the match rules because numeric match
rules are used which test only numbers and ignore following
```

```
text.
Read Test
        loop 2: 12
        loop 5: 14
        loop 7: 20
        Final: pass
Write Test              <top marker D>
        loop 1: 8       <match E>
        loop 3: 90      <match F>
        loop 5: 17
        loop 6: 88      <match D>
        Final: pass
Compute Test            <bottom marker A>
        loop 1: 77
        loop 2: 20
        loop 3: 30
        loop 4: 81
        loop 5: 91
        loop 6: 98
        Final: pass     <match  G>
Summary Results         <top marker A>
  Read Test
    Final       50

  Write Test            <top marker B>
    Final       88      <match B>

  Compute Test          <top marker C>
    Final       100     <match C>
Final   1050            <match A>
Total Compute Time 5.2
```

9.5.3 Result File

```
This is the result file for the search direction, mark, and
instance count example.
The text in the <>'s are comments for the example. They do not
affect the results of the match rules because numeric match
rules are used which test only numbers and ignore following
text.
Read Test
```

```
        loop 2: 9
        loop 3: 14
        loop 12: 20
        Final: pass
Write Test              <top marker D>
        loop 1: 13      <match E>
        loop 3: 100     <match F>
        loop 5: 17
        loop 6: 82      <match D>
        Final: pass
Compute Test            <bottom marker A>
        loop 1: 77
        loop 2: 20
        loop 5: 91
        loop 6: 98
        Final: warning  <match G>
Summary Results         <top marker A>
  Read Test
    Final       50

  Write Test            <top marker B>
    Final       95      <match B>

  Compute Test          <top marker C>
    Final       20      <match C>
Final   1130            <match A>
Total Compute Time 5.2
```

9.5.4 compareTestOutput Output

```
-------------------------------------------------------------
---------- Comparison test found test failure(s) ------------
-------------------------------------------------------------
The numeric value compare rule:
    "Final"     1%      LAST
  found lines:
    Reference( 37): Final       1050            <match A>
    Result( 34): Final  1130            <match A>
  Numerical comparison determined that the test program's value was too large:
        Reference: 1050.0
```

```
        Result: 1130.0
=====================================================
The numeric value compare rule:
    "Final"             1%
  found lines:
    Reference( 33): Final       88      <match B>
    Result( 30): Final  95      <match B>
  Numerical comparison determined that the test program's value was too large:
        Reference: 88.0
        Result: 95.0
=====================================================
The numeric value compare rule:
    "Final"     1%
  found lines:
    Reference( 36): Final       100     <match C>
    Result( 33): Final  20      <match C>
  Numerical comparison determined that the test program's value was too small:
        Reference: 100.0
        Result: 20.0
=====================================================
The numeric value compare rule:
    "loop"              1%      -1
  found lines:
    Reference( 18): loop 6: 88  <match D>
    Result( 17): loop 6: 82     <match D>
  Numerical comparison determined that the test program's value was too small:
        Reference: 88.0
        Result: 82.0
=====================================================
The numeric value compare rule:
    "loop"              1%
  found lines:
    Reference( 15): loop 1: 8   <match E>
    Result( 14): loop 1: 13     <match E>
  Numerical comparison determined that the test program's value was too large:
        Reference: 8.0
        Result: 13.0
=====================================================
The numeric value compare rule:
    "loop"              1%      2
  found lines:
```

```
    Reference( 16): loop 3: 90  <match F>
    Result( 15): loop 3: 100    <match F>
  Numerical comparison determined that the test program's value was too large:
        Reference: 90.0
        Result: 100.0
=====================================================
The text compare rule:
    "Final"             Matches 2
  found lines:
    Reference( 27): Final: pass <match  G>
    Result( 24): Final: warning <match G>
  These lines do not match closely enough.
=====================================================
compareTestOutput found test program failure.
```