MP-CodeCheck User's Manual (v.0.0.6)

Merly, Inc. (Date: 2022-05-28)

ABSTRACT

Software debugging, the process of identifying and mitigating software defects, has been reported to consume upwards of 50% of all software development time. Software defects can arise from a number of issues. These can include logical errors, temporally or spatially inefficient code, or poorly designed and implemented code, also known as technical debt, to name a few. Some of the most severe software defects are security vulnerabilities, which can compromise the safety of a software system, an organization, or even an entire company. We created MP-CodeCheck (MPCC) to help programmers identify and address such defects (also known as anomalies). In this guide, we walk users through MPCC's general uses as well as the steps required to setup, run, and perform inference against a code base.

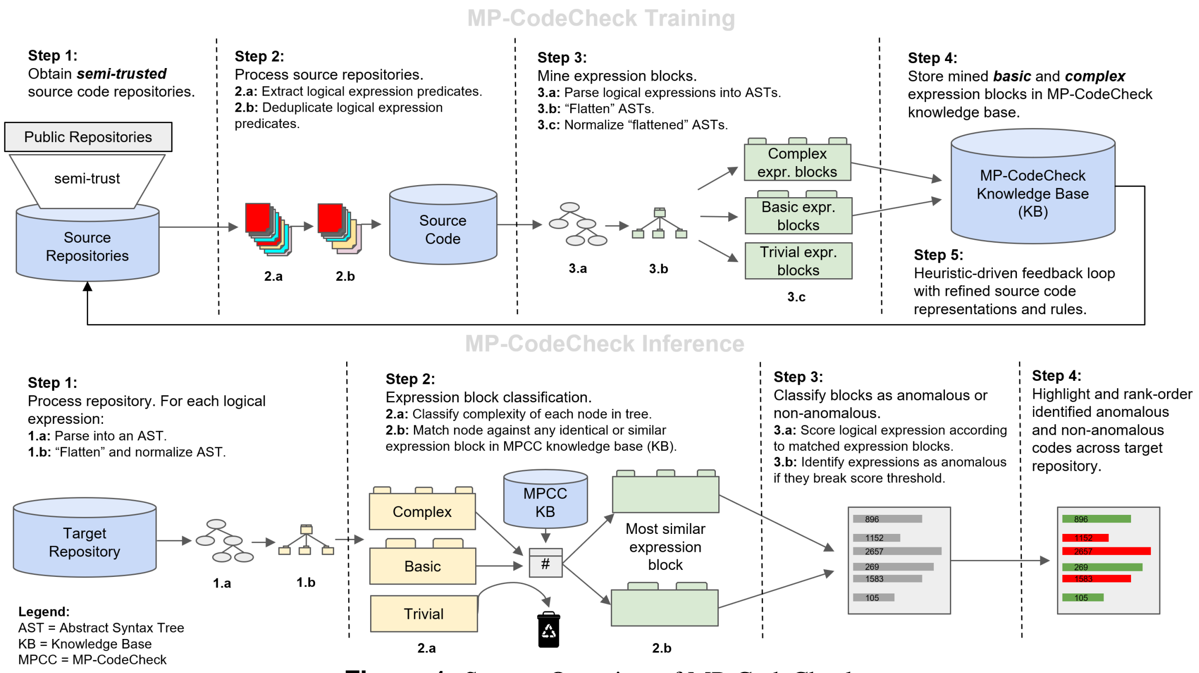
Your code is never sent to us. Ever. Your privacy is our #1 priority.

Contents

Merly's MP-CodeCheck

1 What is MP-CodeCheck?

MP-CodeCheck (MPCC) is an AI-based code anomaly detection system. More specifically, MPCC uses self-supervision, federated learning, and programmatic-guided evolution to detect anomalous code patterns. An overview of its system design is shown in Figure 1. MPCC was designed to learn good and bad code syntax, patterns, and semantics from a large corpora of existing code. Once trained, MPCC's model can be used for a variety of tasks such as: *(i)* detecting potential anomalies in existing code, *(ii)* grading the quality of an existing repository, and *(iii)* guiding programmers through the important aspects of a new code repository, to name a few.



MP-CodeCheck Training
Step 1:
Obtain ***semi-trusted*** source code repositories.
Public Repositories
semi-trust
Source Repositories
Step 2:
Process source repositories.
2.a: Extract logical expression predicates.
2.b: Deduplicate logical expression predicates.
2.a
2.b
Source Code
Step 3:
Mine expression blocks.
3.a: Parse logical expressions into ASTs.
3.b: "Flatten" ASTs.
3.c: Normalize "flattened" ASTs.
3.a
3.b
Complex expr. blocks
Basic expr. blocks
Trivial expr. blocks
3.c
Step 4:
Store mined ***basic*** and ***complex*** expression blocks in MP-CodeCheck knowledge base.
MP-CodeCheck Knowledge Base (KB)
Step 5:
Heuristic-driven feedback loop with refined source code representations and rules.
MP-CodeCheck Inference
Step 1:
Process repository. For each logical expression:
1.a: Parse into an AST.
1.b: "Flatten" and normalize AST.
Target Repository
1.a
1.b
Step 2:
Expression block classification.
2.a: Classify complexity of each node in tree.
2.b: Match node against any identical or similar expression block in MPCC knowledge base (KB).
Complex
Basic
Trivial
MPCC KB
#
Most similar expression block
2.a
2.b
Step 3:
Classify blocks as anomalous or non-anomalous.
3.a: Score logical expression according to matched expression blocks.
3.b: Identify expressions as anomalous if they break score threshold.
896
1152
2657
269
1583
105
Step 4:
Highlight and rank-order identified anomalous and non-anomalous codes across target repository.
896
1152
2657
269
1583
105
Legend:
AST = Abstract Syntax Tree
KB = Knowledge Base
MPCC = MP-CodeCheck

Figure 1. System Overview of MP-CodeCheck.

At its core, MPCC is a machine programming (MP) system that autonomously identifies anomalous logical expressions directly in source code. These anomalous expressions, also known as anomalies, are often latent defects in the existing code that programmers have failed to identify or correct. MPCC helps programmers find these anomalies and correct them, thereby improving the overall quality of the existing software. For this limited release version of MPCC, we only include MPCC the ability to perform inference (i.e., detect good or bad patterns) on code. In subsequent releases of MPCC, we may also include the ability to train new models on other code bases, including users' own proprietary ones.

2 Installation Instructions

Below we list MPCC's installation instructions for the currently supported operating systems (OSes). If you previously installed MPCC without a product key, but now have one, it is safe to run the MerlyInstaller again to register MPCC with a product key. A product key is a 16 character string, separated by hyphens each four characters and is generated when you register MPCC on Merly's website (e.g., 5SA9-HBP2-WRBV-5WA1).

If you encounter any trouble with these steps, please contact `support@merly.ai` for assistance.

2.1 Linux (CentOS, RedHat, SUSE, Ubuntu), MacOS (M1 ARM, x64 Intel)

Launch a command line interface (CLI) and execute the following command:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/merly-ai/MPCC-Universal/main/install.sh)"
```

If you have not registered with a product key, execute the following command from the CLI, where `<key>` is your product key:

```
./MerlyInstaller -k <key> install
```

For MacOS, we recommend running MPCC with iTerm2, due to its support of a broader color scheme than is possible for the default MacOS terminal. You can download it for free here: https://iterm2.com/downloads.html.

2.2 Windows (64-bit)

Launch `cmd.exe` (do not use PowerShell as the below cURL command will not work). Navigate to your user preferred installation directory (e.g., `cd C:\Users\Paul`). Then execute the following commands where `<key>` is your product key:

```
mkdir MPCC
cd MPCC
curl -LO https://github.com/merly-ai/MP-CodeCheckBin-Windows/raw/main/bin/latest/MerlyInstaller.exe
MerlyInstaller -k <key> install
```

2.3 Updating MPCC with the MerlyInstaller

MPCC is constantly being updated. To update your local copy of MPCC to the latest version including updating all of latest programming language models it supports, simply run the MerlyInstaller (installed in the above installation steps) in the following way:

```
MerlyInstaller updateall
```

If you would like to update only certain components, please use the 'usage' command from the MerlyInstaller as shown below for more details on how to update only the components you are interested in.

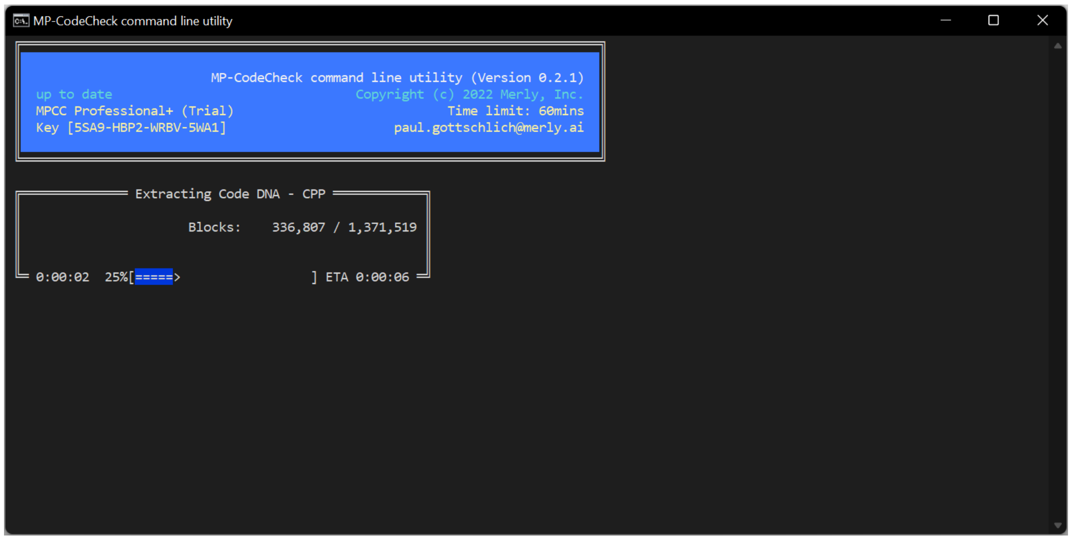
```
MerlyInstaller usage
```

3 Launching MP-CodeCheck

Now that setup is complete, let's launch MPCC to perform inference analysis. From the command line interface (CLI), type the following (where "[code base folder]" is a directory that contains the code you want to analyze):

MacOS, Linux `./MPCC infer -D [code base folder]`

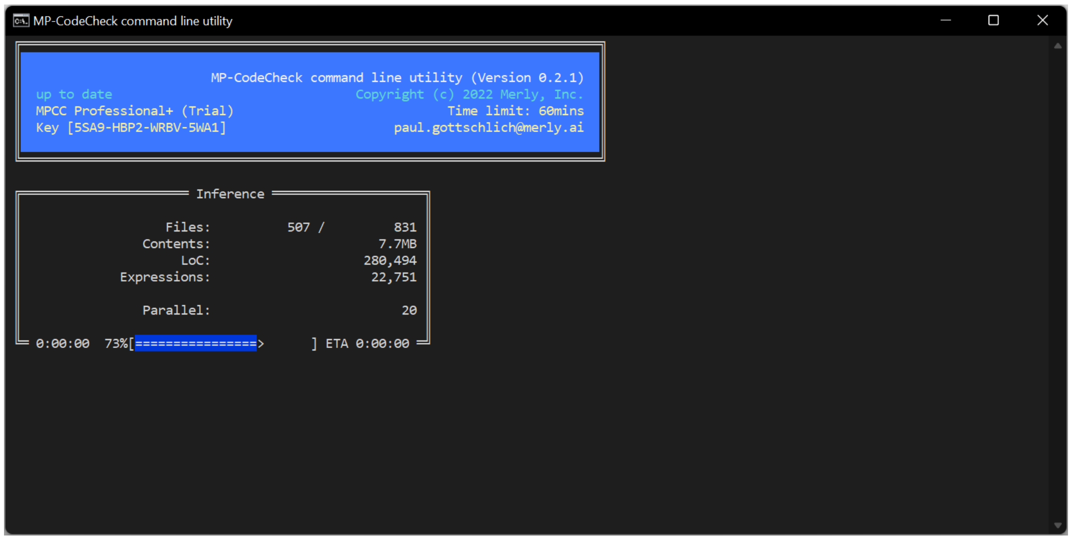
Windows `MPCC.exe infer -D [code base folder]`



```
                          MP-CodeCheck command line utility (Version 0.2.1)
  up to date                                 Copyright (c) 2022 Merly, Inc.
  MPCC Professional+ (Trial)                             Time limit: 60mins
  Key [5SA9-HBP2-WRBV-5WA1]                        paul.gottschlich@merly.ai

╔═══════════════ Extracting Code DNA - CPP ══════════════╗
║                                                        ║
║                     Blocks:    336,807 / 1,371,519     ║
║                                                        ║
╚═ 0:00:02  25%[=====>                    ] ETA 0:00:06 ═╝
```

Figure 2. Launching MP-CodeCheck and Extracting Code DNA.

When run successfully, MPCC will display information that looks similar to the screenshot shown in Figure 2. This shows the progress of MPCC extracting the code DNA from the training model.



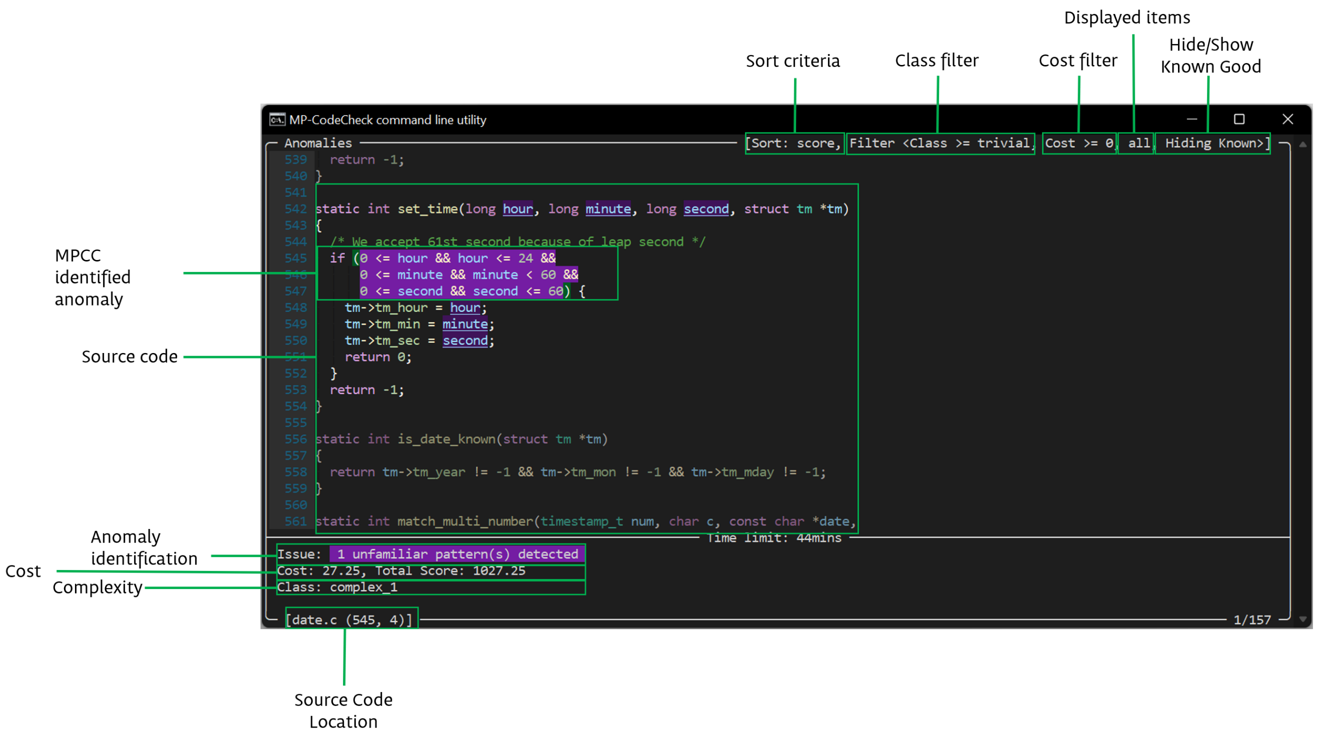
```
                    MP-CodeCheck command line utility (Version 0.2.1)
up to date                             Copyright (c) 2022 Merly, Inc.
MPCC Professional+ (Trial)                         Time limit: 60mins
Key [5SA9-HBP2-WRBV-5WA1]                   paul.gottschlich@merly.ai

═══════════════ Inference ═══════════════

           Files:          507 /         831
        Contents:                      7.7MB
             LoC:                    280,494
     Expressions:                     22,751

        Parallel:                         20

0:00:00  73%[================>      ] ETA 0:00:00
```

Figure 3. MP-CodeCheck Performing Inference Processing.

When MPCC has loaded its trained model and processed the code DNA, it begins inference analysis on all source code that it finds in the files of the directory (or subdirectories) you have supplied when launching it. Figure 3 shows an example of MPCC's inference progress in analyzing a code repository, how much work it has completed, and how much work is remaining. When inference analysis has completed, the *Code View* screen will appear (shown in Figure 4), which will allow a user to analyze the inference results as discussed in the next section.

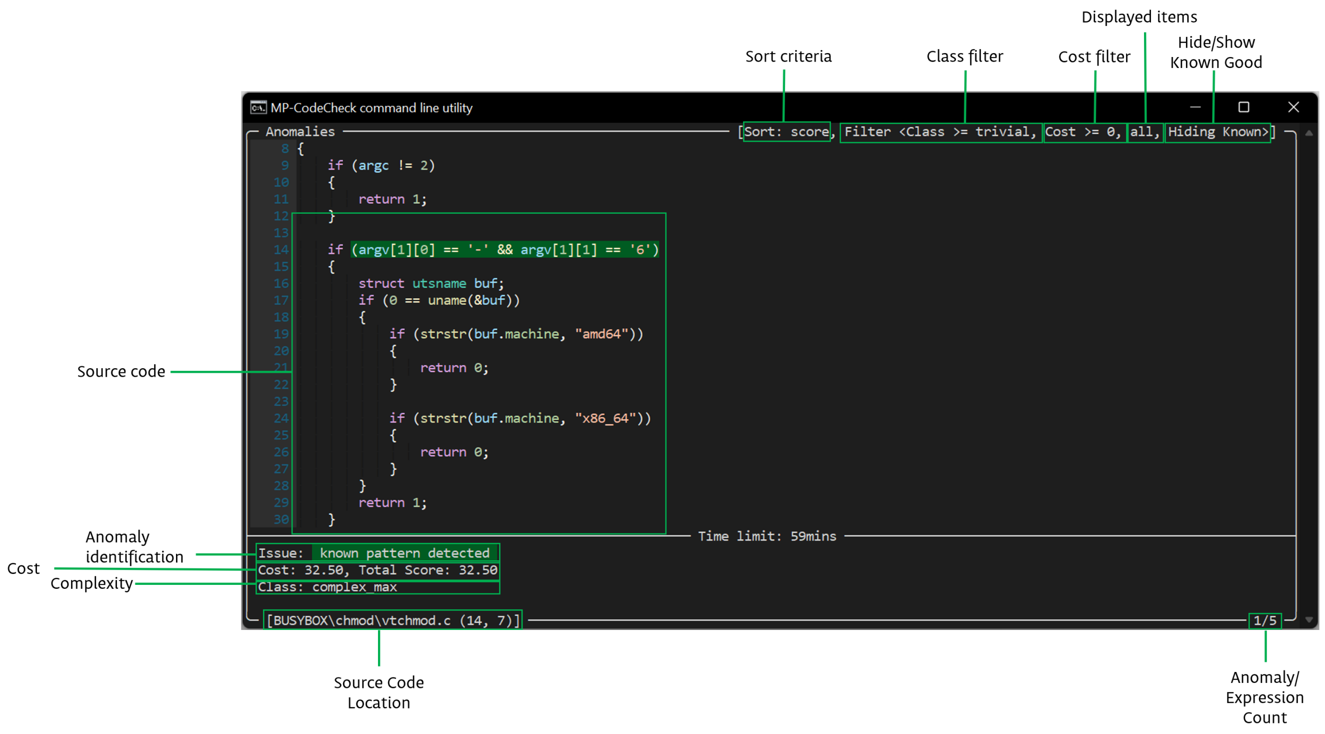
4 Exploring MPCC's Inference Results

After inference analysis is performed, MPCC will show a user interface that includes source code, with an expression highlighted. We call this screen the *Code View*, which will be described in more detail in Views section of this manual. Figure 4 provides an example of an anomalous code example found by MPCC.



```
[Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]
  return -1;
}

static int set_time(long hour, long minute, long second, struct tm *tm)
{
  /* We accept 61st second because of leap second */
  if (0 <= hour && hour <= 24 &&
      0 <= minute && minute < 60 &&
      0 <= second && second <= 60) {
    tm->tm_hour = hour;
    tm->tm_min = minute;
    tm->tm_sec = second;
    return 0;
  }
  return -1;
}

static int is_date_known(struct tm *tm)
{
  return tm->tm_year != -1 && tm->tm_mon != -1 && tm->tm_mday != -1;
}

static int match_multi_number(timestamp_t num, char c, const char *date,
                                                           Time limit: 44mins
Issue:  1 unfamiliar pattern(s) detected
Cost: 27.25, Total Score: 1027.25
Class: complex_1
[date.c (545, 4)]                                                                   1/157
```

Figure 4. An Anomalous Example in MP-CodeCheck's Code View.



```
[Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]
{
    if (argc != 2)
    {
        return 1;
    }

    if (argv[1][0] == '-' && argv[1][1] == '6')
    {
        struct utsname buf;
        if (0 == uname(&buf))
        {
            if (strstr(buf.machine, "amd64"))
            {
                return 0;
            }

            if (strstr(buf.machine, "x86_64"))
            {
                return 0;
            }
        }
        return 1;
    }
                                                           Time limit: 59mins
Issue:  known pattern detected
Cost: 32.50, Total Score: 32.50
Class: complex_max
[BUSYBOX\chmod\vtchmod.c (14, 7)]                                                   1/5
```

Figure 5. A Non-Anomalous Example in MP-CodeCheck's Code View.

Sort Criteria: This refers to how MPCC is sorting the list of expressions it has found. This can be via score (a numeric value assigned by anomaly identification and complexity), or location (sequential code order).

Class Filter: This refers to which class of complexity is being filtered in the current view. This can be set from a minimum value of trivial to a high value of Max complexity.

Cost Filter: This refers to a "mental cost" of an expression. This filter can be set from a minimum value of 0 to a maximum value of 2,000.

Displayed Items: This refers to which items MPCC is displaying. It can be set to all expressions, or only anomalous expressions.

Hide/Show Known Good: This refers to whether or not MPCC displays expressions that have been marked by the user as Known Good.

Anomaly Identification: This displays whether or not MPCC has identified the current expression as an anomaly. Non-anomalous expressions will be classified as "known pattern detected" and highlighted in green. Anomalous expressions will be classified as "unfamiliar pattern(s) detected" and will be highlighted in green.

Cost: This displays the "mental cost" of the current expression.

Complexity: This displays the class of complexity of the current expression.

Source Code Location: This displays the file location of the source code under review.

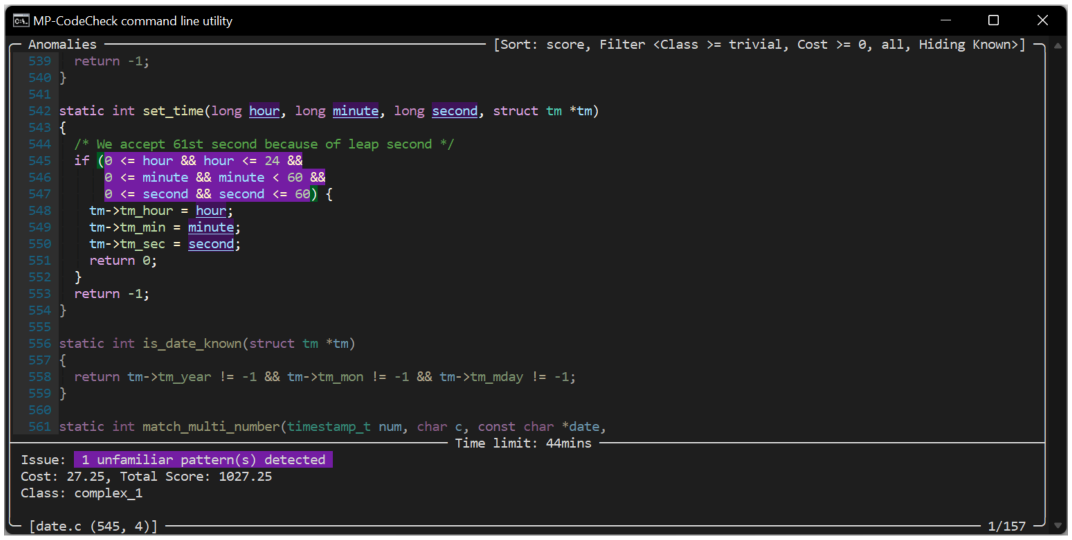
Anomaly/Expression Count: This displays the count of the highlighted expression, as well as the total expressions found in this file. If the user toggles the filter to show only anomalies, this will display the count of highlighted anomaly, and the total anomalies found in the current file.

Walking Through Code: You can move forwards and backwards through the expressions by using the left and right arrow keys, and can page up and page down through the code (by location) using the Page Up and Page Down keys. You can also scroll up and down through the code by hold the Control key while pressing the up or the down arrow, respectively.

5 Basic Commands and Views

In MPCC, there are a number of supported keyboard and mouse commands. In this section we describe those keystrokes and explain mouse behavior. Perhaps the most important initial command to remember is the *help* command which can be launched by pressing the character 'h' on your keyboard. The help command lists all of the keyboard commands, so if you ever find yourself not remembering a keyboard command, just press 'h' and MPCC will launch the keyboard shortcut commands. A screenshot of the help dialogue box is shown in Figure 11.

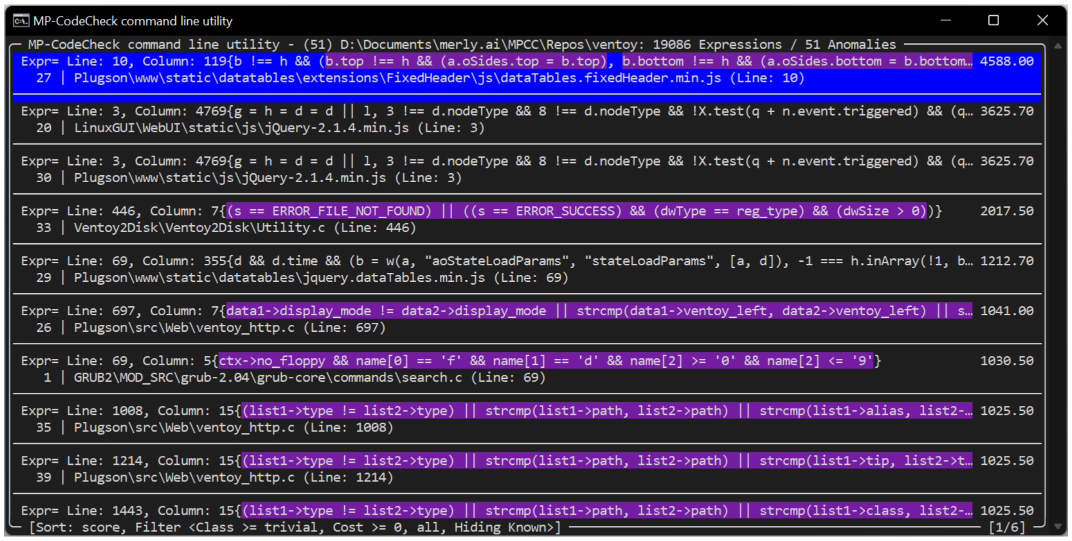
In addition to commands, there are several screens users can utilize to help them gain deeper insights into specific anomalies, general anomaly information, anomalies by file, anomalies per file, and so forth.



```
MP-CodeCheck command line utility
Anomalies                                [Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]
  return -1;
}

static int set_time(long hour, long minute, long second, struct tm *tm)
{
  /* We accept 61st second because of leap second */
  if (0 <= hour && hour <= 24 &&
      0 <= minute && minute < 60 &&
      0 <= second && second <= 60) {
    tm->tm_hour = hour;
    tm->tm_min = minute;
    tm->tm_sec = second;
    return 0;
  }
  return -1;
}

static int is_date_known(struct tm *tm)
{
  return tm->tm_year != -1 && tm->tm_mon != -1 && tm->tm_mday != -1;
}

static int match_multi_number(timestamp_t num, char c, const char *date,
                                               Time limit: 44mins
Issue:  1 unfamiliar pattern(s) detected
Cost: 27.25, Total Score: 1027.25
Class: complex_1

[date.c (545, 4)]                                                                                  1/157
```

Figure 6. MP-CodeCheck's Code View.

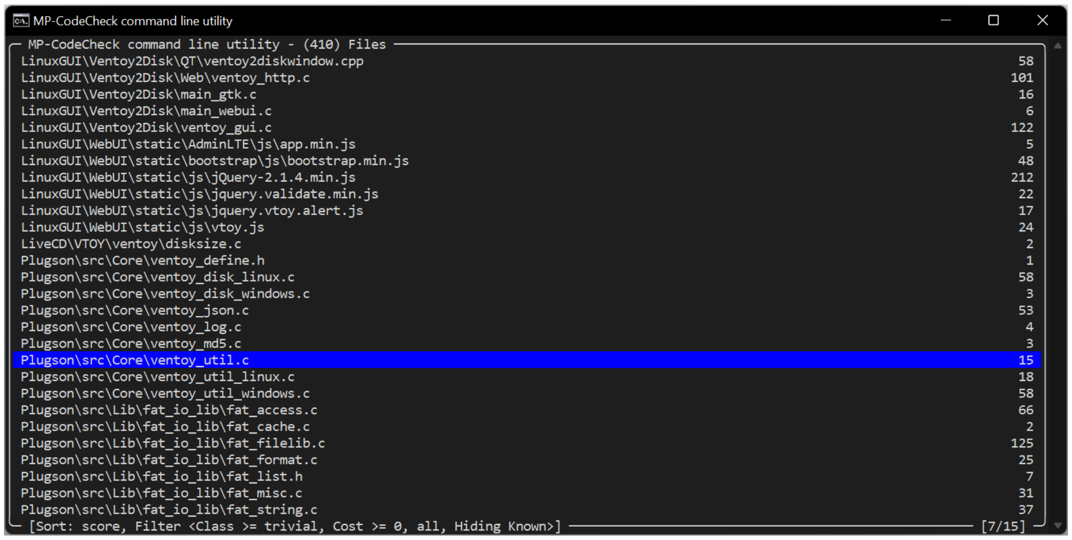
Code View: This is the view of all of the code, with the expressions found highlighted. This view is the default view when MPCC is initially run.



```
MP-CodeCheck command line utility
MP-CodeCheck command line utility - (51) D:\Documents\merly.ai\MPCC\Repos\ventoy: 19086 Expressions / 51 Anomalies
Expr= Line: 10, Column: 119{b !== h && (b.top !== h && (a.oSides.top = b.top), b.bottom !== h && (a.oSides.bottom = b.bottom… 4588.00
  27 | Plugson\www\static\datatables\extensions\FixedHeader\js\dataTables.fixedHeader.min.js (Line: 10)

Expr= Line: 3, Column: 4769{g = h = d = d || 1, 3 !== d.nodeType && 8 !== d.nodeType && !X.test(q + n.event.triggered) && (q… 3625.70
  20 | LinuxGUI\WebUI\static\js\jQuery-2.1.4.min.js (Line: 3)

Expr= Line: 3, Column: 4769{g = h = d = d || 1, 3 !== d.nodeType && 8 !== d.nodeType && !X.test(q + n.event.triggered) && (q… 3625.70
  30 | Plugson\www\static\js\jQuery-2.1.4.min.js (Line: 3)

Expr= Line: 446, Column: 7{(s == ERROR_FILE_NOT_FOUND) || ((s == ERROR_SUCCESS) && (dwType == reg_type) && (dwSize > 0))}      2017.50
  33 | Ventoy2Disk\Ventoy2Disk\Utility.c (Line: 446)

Expr= Line: 69, Column: 355{d && d.time && (b = w(a, "aoStateLoadParams", "stateLoadParams", [a, d]), -1 === h.inArray(!1, b… 1212.70
  29 | Plugson\www\static\datatables\jquery.dataTables.min.js (Line: 69)

Expr= Line: 697, Column: 7{data1->display_mode != data2->display_mode || strcmp(data1->ventoy_left, data2->ventoy_left) || s… 1041.00
  26 | Plugson\src\Web\ventoy_http.c (Line: 697)

Expr= Line: 69, Column: 5{ctx->no_floppy && name[0] == 'f' && name[1] == 'd' && name[2] >= '0' && name[2] <= '9'}             1030.50
   1 | GRUB2\MOD_SRC\grub-2.04\grub-core\commands\search.c (Line: 69)

Expr= Line: 1008, Column: 15{(list1->type != list2->type) || strcmp(list1->path, list2->path) || strcmp(list1->alias, list2-… 1025.50
  35 | Plugson\src\Web\ventoy_http.c (Line: 1008)

Expr= Line: 1214, Column: 15{(list1->type != list2->type) || strcmp(list1->path, list2->path) || strcmp(list1->tip, list2->t… 1025.50
  39 | Plugson\src\Web\ventoy_http.c (Line: 1214)

Expr= Line: 1443, Column: 15{(list1->type != list2->type) || strcmp(list1->path, list2->path) || strcmp(list1->class, list2-… 1025.50
[Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]                                         [1/6]
```

Figure 7. MP-CodeCheck's Anomalies View.

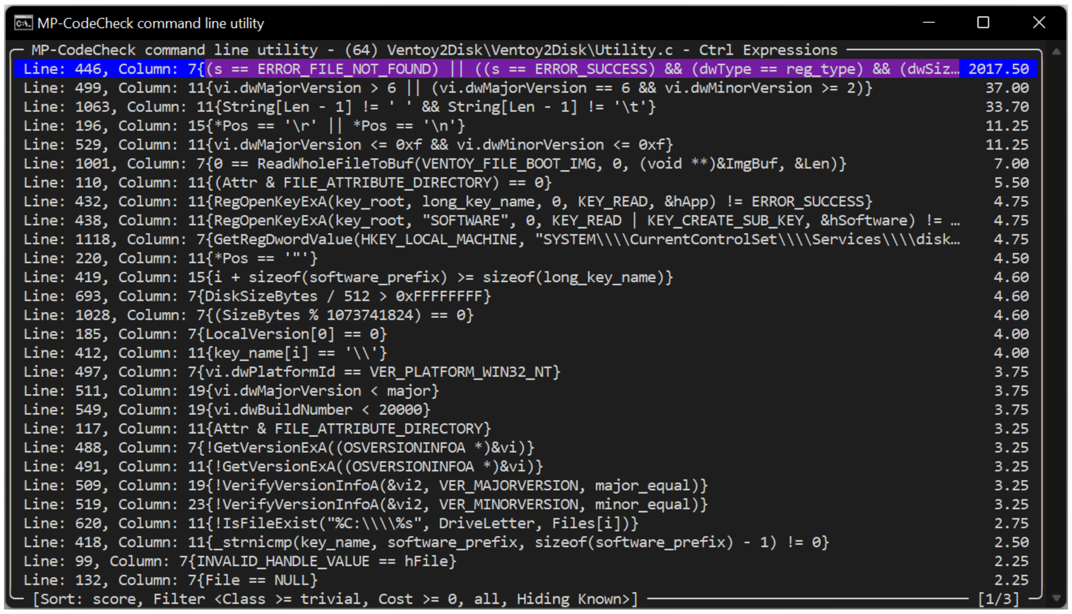
Anomalies View: Press 'a' to switch to the Anomalies view. This view shows all of the expressions in the code (across all files) that MPCC has determined to be an anomaly, sorted by score. You can move up and down the list using the up and down arrows, or the Page Up and Page Down keys. Press Enter with an anomaly highlighted to switch back to the Code View of that specific anomaly.



```
MP-CodeCheck command line utility
MP-CodeCheck command line utility - (410) Files
LinuxGUI\Ventoy2Disk\QT\ventoy2diskwindow.cpp                                58
LinuxGUI\Ventoy2Disk\Web\ventoy_http.c                                      101
LinuxGUI\Ventoy2Disk\main_gtk.c                                              16
LinuxGUI\Ventoy2Disk\main_webui.c                                             6
LinuxGUI\Ventoy2Disk\ventoy_gui.c                                           122
LinuxGUI\WebUI\static\AdminLTE\js\app.min.js                                  5
LinuxGUI\WebUI\static\bootstrap\js\bootstrap.min.js                          48
LinuxGUI\WebUI\static\js\jQuery-2.1.4.min.js                                212
LinuxGUI\WebUI\static\js\jquery.validate.min.js                              22
LinuxGUI\WebUI\static\js\jquery.vtoy.alert.js                                17
LinuxGUI\WebUI\static\js\vtoy.js                                             24
LiveCD\VTOY\ventoy\disksize.c                                                 2
Plugson\src\Core\ventoy_define.h                                              1
Plugson\src\Core\ventoy_disk_linux.c                                         58
Plugson\src\Core\ventoy_disk_windows.c                                        3
Plugson\src\Core\ventoy_json.c                                               53
Plugson\src\Core\ventoy_log.c                                                 4
Plugson\src\Core\ventoy_md5.c                                                 3
Plugson\src\Core\ventoy_util.c                                               15
Plugson\src\Core\ventoy_util_linux.c                                         18
Plugson\src\Core\ventoy_util_windows.c                                       58
Plugson\src\Lib\fat_io_lib\fat_access.c                                      66
Plugson\src\Lib\fat_io_lib\fat_cache.c                                        2
Plugson\src\Lib\fat_io_lib\fat_filelib.c                                    125
Plugson\src\Lib\fat_io_lib\fat_format.c                                      25
Plugson\src\Lib\fat_io_lib\fat_list.h                                         7
Plugson\src\Lib\fat_io_lib\fat_misc.c                                        31
Plugson\src\Lib\fat_io_lib\fat_string.c                                      37
[Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]   [7/15]
```

Figure 8. MP-CodeCheck's Files View.

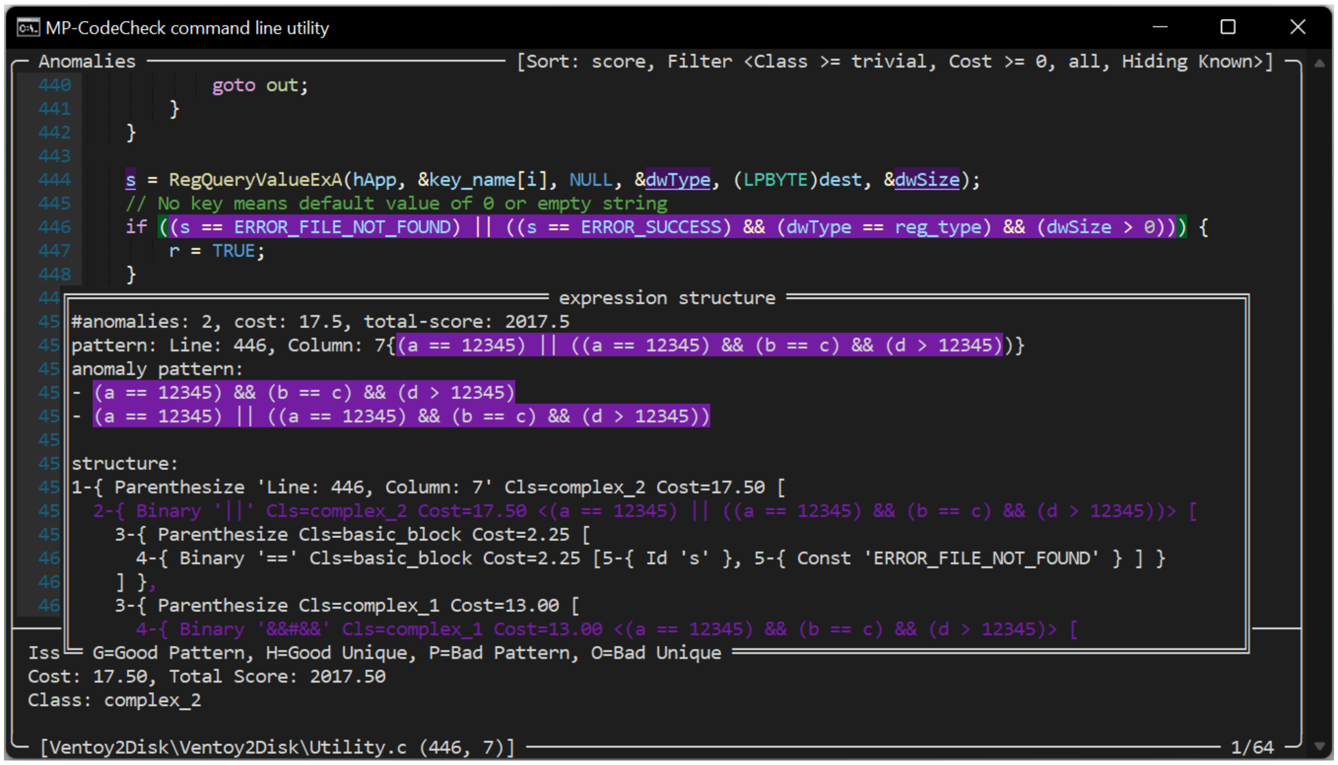
Files View: Press 'f' to switch to the Files view. This view shows all of the source code files, with the total number of expressions MPCC found in each file. You can move up and down the list using the up and down arrows, or the Page Up and Page Down keys. Press Enter with a file highlighted to switch back to the Code View of the expressions within that specific file.



```
MP-CodeCheck command line utility
MP-CodeCheck command line utility - (64) Ventoy2Disk\Ventoy2Disk\Utility.c - Ctrl Expressions
Line: 446, Column: 7{(s == ERROR_FILE_NOT_FOUND) || ((s == ERROR_SUCCESS) && (dwType == reg_type) && (dwSiz… 2017.50
Line: 499, Column: 11{vi.dwMajorVersion > 6 || (vi.dwMajorVersion == 6 && vi.dwMinorVersion >= 2)}     37.00
Line: 1063, Column: 11{String[Len - 1] != ' ' && String[Len - 1] != '\t'}                          33.70
Line: 196, Column: 15{*Pos == '\r' || *Pos == '\n'}                                                 11.25
Line: 529, Column: 11{vi.dwMajorVersion <= 0xf && vi.dwMinorVersion <= 0xf}                         11.25
Line: 1001, Column: 7{0 == ReadWholeFileToBuf(VENTOY_FILE_BOOT_IMG, 0, (void **)&ImgBuf, &Len)}      7.00
Line: 110, Column: 11{(Attr & FILE_ATTRIBUTE_DIRECTORY) == 0}                                        5.50
Line: 432, Column: 11{RegOpenKeyExA(key_root, long_key_name, 0, KEY_READ, &hApp) != ERROR_SUCCESS}   4.75
Line: 438, Column: 11{RegOpenKeyExA(key_root, "SOFTWARE", 0, KEY_READ | KEY_CREATE_SUB_KEY, &hSoftware) != …  4.75
Line: 1118, Column: 7{GetRegDwordValue(HKEY_LOCAL_MACHINE, "SYSTEM\\\\CurrentControlSet\\\\Services\\\\disk…  4.75
Line: 220, Column: 11{*Pos == '"'}                                                                   4.50
Line: 419, Column: 15{i + sizeof(software_prefix) >= sizeof(long_key_name)}                          4.60
Line: 693, Column: 7{DiskSizeBytes / 512 > 0xFFFFFFFF}                                               4.60
Line: 1028, Column: 7{(SizeBytes % 1073741824) == 0}                                                 4.60
Line: 185, Column: 7{LocalVersion[0] == 0}                                                           4.00
Line: 412, Column: 11{key_name[i] == '\\'}                                                           4.00
Line: 497, Column: 7{vi.dwPlatformId == VER_PLATFORM_WIN32_NT}                                       3.75
Line: 511, Column: 19{vi.dwMajorVersion < major}                                                     3.75
Line: 549, Column: 19{vi.dwBuildNumber < 20000}                                                      3.75
Line: 117, Column: 11{Attr & FILE_ATTRIBUTE_DIRECTORY}                                               3.25
Line: 488, Column: 7{!GetVersionExA((OSVERSIONINFOA *)&vi)}                                          3.25
Line: 491, Column: 11{!GetVersionExA((OSVERSIONINFOA *)&vi)}                                         3.25
Line: 509, Column: 19{!VerifyVersionInfoA(&vi2, VER_MAJORVERSION, major_equal)}                      3.25
Line: 519, Column: 23{!VerifyVersionInfoA(&vi2, VER_MINORVERSION, minor_equal)}                      3.25
Line: 620, Column: 11{!IsFileExist("%C:\\\\%s", DriveLetter, Files[i])}                              2.75
Line: 418, Column: 11{_strnicmp(key_name, software_prefix, sizeof(software_prefix) - 1) != 0}        2.50
Line: 99, Column: 7{INVALID_HANDLE_VALUE == hFile}                                                   2.25
Line: 132, Column: 7{File == NULL}                                                                   2.25
[Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]   [1/3]
```

Figure 9. MP-CodeCheck's Expressions View.

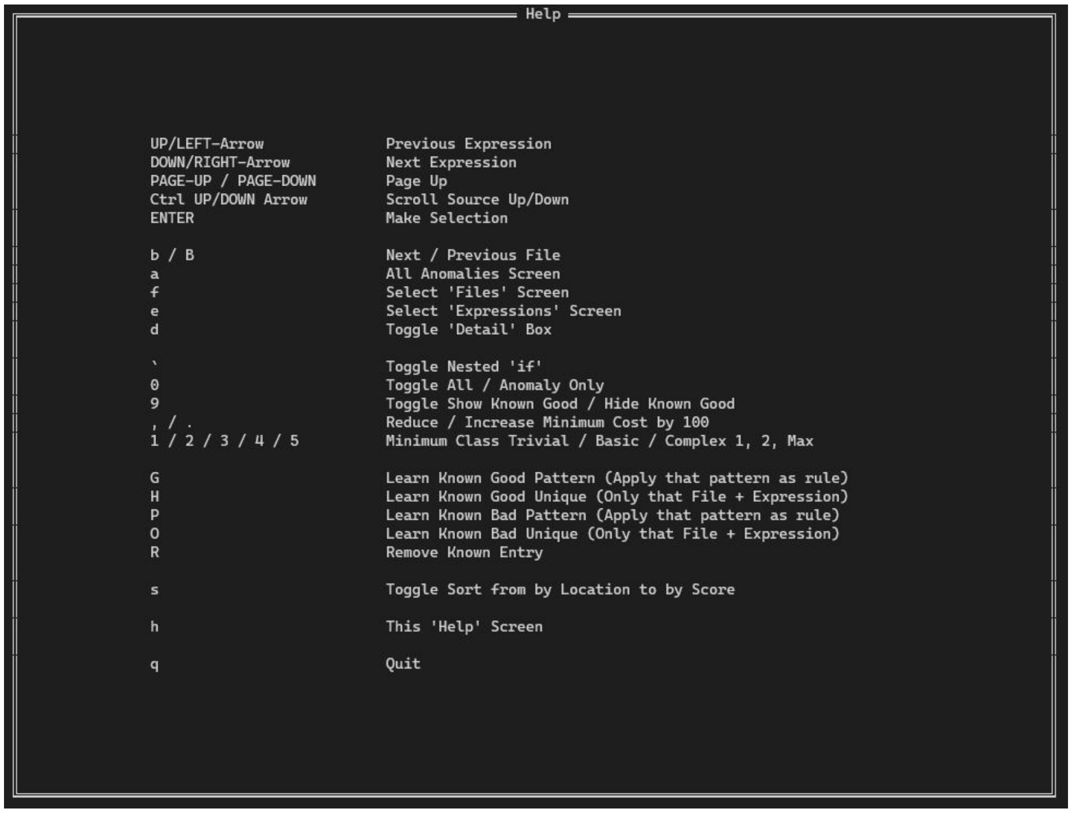
Expressions View: Press 'e' to switch to the Expressions view. This view shows all of the expressions in the current file, sorted by score. You can move up and down the list using the up and down arrows, or the Page Up and Page Down keys. Also note that you can toggle the sort between code location and score by pressing the 's' key. Press Enter with an expression highlighted to switch back to the Code View with that specific expression highlighted.



```
MP-CodeCheck command line utility
Anomalies ──────────────── [Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]
            goto out;
        }
    }

    s = RegQueryValueExA(hApp, &key_name[i], NULL, &dwType, (LPBYTE)dest, &dwSize);
    // No key means default value of 0 or empty string
    if ((s == ERROR_FILE_NOT_FOUND) || ((s == ERROR_SUCCESS) && (dwType == reg_type) && (dwSize > 0))) {
        r = TRUE;
    }
═══════════════════════ expression structure ═══════════════════════
#anomalies: 2, cost: 17.5, total-score: 2017.5
pattern: Line: 446, Column: 7{(a == 12345) || ((a == 12345) && (b == c) && (d > 12345))}
anomaly pattern:
- (a == 12345) && (b == c) && (d > 12345)
- (a == 12345) || ((a == 12345) && (b == c) && (d > 12345))

structure:
1-{ Parenthesize 'Line: 446, Column: 7' Cls=complex_2 Cost=17.50 [
  2-{ Binary '||' Cls=complex_2 Cost=17.50 <(a == 12345) || ((a == 12345) && (b == c) && (d > 12345))> [
    3-{ Parenthesize Cls=basic_block Cost=2.25 [
      4-{ Binary '==' Cls=basic_block Cost=2.25 [5-{ Id 's' }, 5-{ Const 'ERROR_FILE_NOT_FOUND' } ] }
    ] },
    3-{ Parenthesize Cls=complex_1 Cost=13.00 [
      4-{ Binary '&&#&&' Cls=complex_1 Cost=13.00 <(a == 12345) && (b == c) && (d > 12345)> [
Iss═ G=Good Pattern, H=Good Unique, P=Bad Pattern, O=Bad Unique ═══
Cost: 17.50, Total Score: 2017.50
Class: complex_2

[Ventoy2Disk\Ventoy2Disk\Utility.c (446, 7)] ──────────────── 1/64
```

Figure 10. MP-CodeCheck's Details View.

Details View: Press 'd' to switch to the Details view. This view shows the detail of the currently selected expression. The detail lets you know how many anomalies MPCC identified within the expression, the cost, and the total score. Press 'd' to return to Code View.



```
                                    Help

UP/LEFT-Arrow              Previous Expression
DOWN/RIGHT-Arrow           Next Expression
PAGE-UP / PAGE-DOWN        Page Up
Ctrl UP/DOWN Arrow         Scroll Source Up/Down
ENTER                      Make Selection

b / B                      Next / Previous File
a                          All Anomalies Screen
f                          Select 'Files' Screen
e                          Select 'Expressions' Screen
d                          Toggle 'Detail' Box

`                          Toggle Nested 'if'
0                          Toggle All / Anomaly Only
9                          Toggle Show Known Good / Hide Known Good
, / .                      Reduce / Increase Minimum Cost by 100
1 / 2 / 3 / 4 / 5          Minimum Class Trivial / Basic / Complex 1, 2, Max

G                          Learn Known Good Pattern (Apply that pattern as rule)
H                          Learn Known Good Unique (Only that File + Expression)
P                          Learn Known Bad Pattern (Apply that pattern as rule)
O                          Learn Known Bad Unique (Only that File + Expression)
R                          Remove Known Entry

s                          Toggle Sort from by Location to by Score

h                          This 'Help' Screen

q                          Quit
```

Figure 11. MP-CodeCheck's Help Dialogue Box.

Help Pop-up: In addition to the above views, you can press the 'h' key in any view to bring up the help screen which will show you all of the hot keys and their functions.

6 Sorting/Filtering Inference Results

The following lists the ways MPCC's inference results on source code data can be sorted and/or filtered.

Sort Criteria:

Options:

- Score (numeric value assigned by anomaly identification and complexity)
- Location (sequential code order)

Default: Score
Toggle: 's' key

Class filter:

Options:

- Trivial (minimum)
- Basic

- Complex 1
- Complex 2
- Max

Default: Trivial
Adjust: ‘1’, ‘2’, ‘3’, ‘4’, ‘5’ keys

Cost filter:

Options:

- 0 (minimum) to 2,000

Default: 0
Adjust: ‘,’ to decrease, ‘.’ to increase, ‘m’ to reset to 0 (minimum)

Displayed items:

Options:

- All expressions
- Anomalies Only

Default: All expressions
Toggle: ‘0’ key

Hide/Show Known Good:

Options:

- Hide Known Good
- Show Known Good

Default: Hide Known Good
Toggle: ‘9’ key

MPCC Generated Files

In addition to the live (online) user interface, you can also review the inference results offline through four MPCC generated files. These files are re-generated each time inference is run successfully. These files will be created in the same folder that the MPCC executable was launched and have the following naming structure.

[Code Repo].by_file.txt: This file lists all anomalous expressions (that are not nested if’s) found by MPCC. This human readable file lists the original anomalous source and its normalized version.

[Code Repo].by_file_nested_if.txt: This file lists all nested if expressions that are found by MPCC to be anomalous. This human readable file lists the original anomalous source and its normalized version.

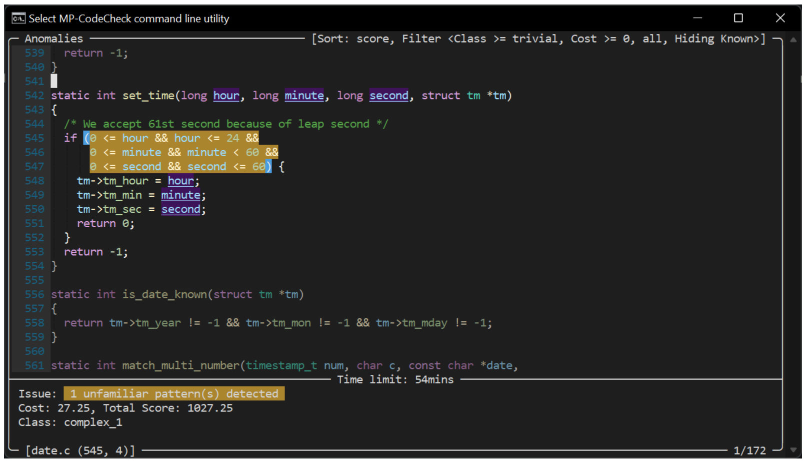
[Code Repo].mpcc.anomaly_list.json: This file lists all expressions that are found by MPCC to be anomalous, in a machine-readable format.

[Code Repo].mpcc.summary.json: This file contains a summary of all of the files, size, and lines of code reviewed by MPCC. It also provides a summarized report of the number of expressions, anomalies, and scores found in the source code that inference was performed on, in a machine-readable format.

7 Advanced Usage

In this section, we describe how to use some of the advanced features of MP-CodeCheck.

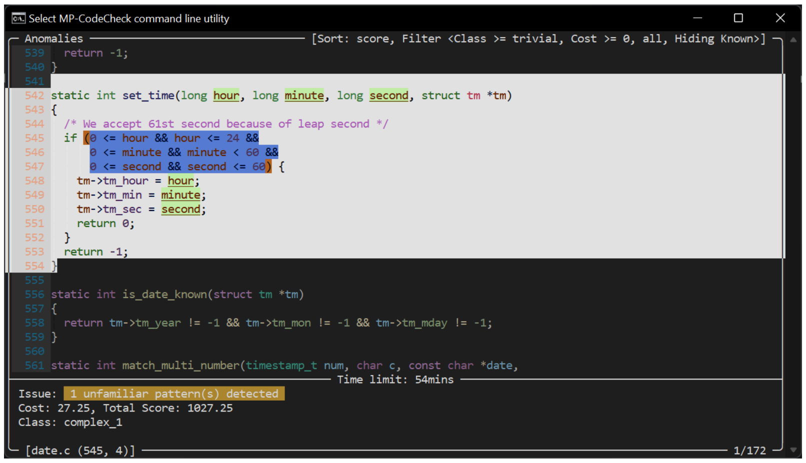
7.1 Copy/Paste Functionality in MPCC



```
Select MP-CodeCheck command line utility
Anomalies                      [Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]
  return -1;
}

static int set_time(long hour, long minute, long second, struct tm *tm)
{
  /* We accept 61st second because of leap second */
  if (0 <= hour && hour <= 24 &&
      0 <= minute && minute < 60 &&
      0 <= second && second <= 60) {
    tm->tm_hour = hour;
    tm->tm_min = minute;
    tm->tm_sec = second;
    return 0;
  }
  return -1;
}

static int is_date_known(struct tm *tm)
{
  return tm->tm_year != -1 && tm->tm_mon != -1 && tm->tm_mday != -1;
}

static int match_multi_number(timestamp_t num, char c, const char *date,
                                          Time limit: 54mins
Issue:  1 unfamiliar pattern(s) detected
Cost: 27.25, Total Score: 1027.25
Class: complex_1

[date.c (545, 4)]                                                                      1/172
```

Figure 12. A white cursor appears to mark the start of your copy area.

Left-click on the beginning of the area you'd like to copy. You'll see a white cursor where the copy will begin (Figure 12). Next, hold down the left mouse button and drag the cursor over the area you'd like to copy, as shown in Figure 13). You can also hold down the Shift key and use the cursor keys (up/down, left/right) to select the area you'd like to copy. When you have the selection highlighted, you can either right-click on your mouse or press the Enter key to complete the copy.



```
Select MP-CodeCheck command line utility
Anomalies                      [Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]
  return -1;
}

static int set_time(long hour, long minute, long second, struct tm *tm)
{
  /* We accept 61st second because of leap second */
  if (0 <= hour && hour <= 24 &&
      0 <= minute && minute < 60 &&
      0 <= second && second <= 60) {
    tm->tm_hour = hour;
    tm->tm_min = minute;
    tm->tm_sec = second;
    return 0;
  }
  return -1;
}

static int is_date_known(struct tm *tm)
{
  return tm->tm_year != -1 && tm->tm_mon != -1 && tm->tm_mday != -1;
}

static int match_multi_number(timestamp_t num, char c, const char *date,
                                          Time limit: 54mins
Issue:  1 unfamiliar pattern(s) detected
Cost: 27.25, Total Score: 1027.25
Class: complex_1

[date.c (545, 4)]                                                                      1/172
```

Figure 13. Highlighted text is ready to be copied.

7.2 MPCC Configuration

For users who wish to customize their MPCC experience, a JSON file is available to configure MPCC to fit your preferences.

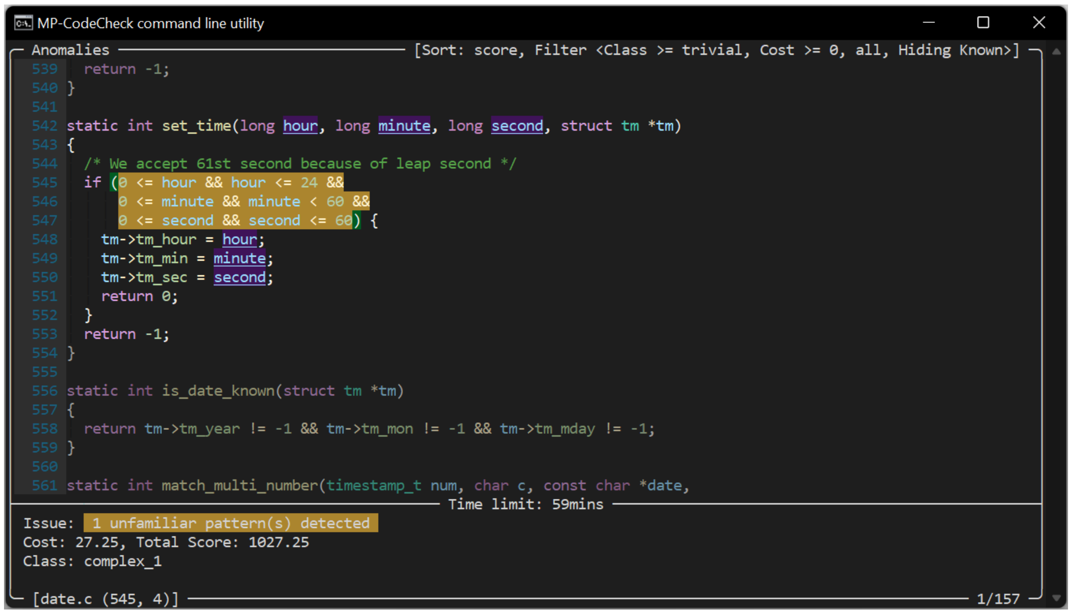
The JSON file is located at the following location:

```
%appdata%\..\local\merly.ai\debugging\MP-CodeCheck\config.json
```

You can use any text editor to modify the colors, log file locations, and settings. Let's take a closer look.

Colors: These are stored in the JSON file in hexadecimal (HEX) RGB; simply use your favorite color picker to find the hex value of the color you'd like, and change the value of the associated item.

For example, you can set `anomaly_background` to RGB `ab852e` to change the highlight color of the anomalous expressions to dark orange.



```
MP-CodeCheck command line utility
Anomalies ──────────────── [Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>]
539   return -1;
540 }
541
542 static int set_time(long hour, long minute, long second, struct tm *tm)
543 {
544   /* We accept 61st second because of leap second */
545   if (0 <= hour && hour <= 24 &&
546       0 <= minute && minute < 60 &&
547       0 <= second && second <= 60) {
548     tm->tm_hour = hour;
549     tm->tm_min = minute;
550     tm->tm_sec = second;
551     return 0;
552   }
553   return -1;
554 }
555
556 static int is_date_known(struct tm *tm)
557 {
558   return tm->tm_year != -1 && tm->tm_mon != -1 && tm->tm_mday != -1;
559 }
560
561 static int match_multi_number(timestamp_t num, char c, const char *date,
──────────────────────────────────── Time limit: 59mins ──────────────
Issue:  1 unfamiliar pattern(s) detected
Cost: 27.25, Total Score: 1027.25
Class: complex_1

[date.c (545, 4)] ─────────────────────────────────────────── 1/157
```

Figure 14. An example of changing the anomaly highlight color in MP-CodeCheck's Code View.

Or, set `highlight_background` to RGB `4a9de0` to change the highlight color of the non-anomalous expressions to light blue.

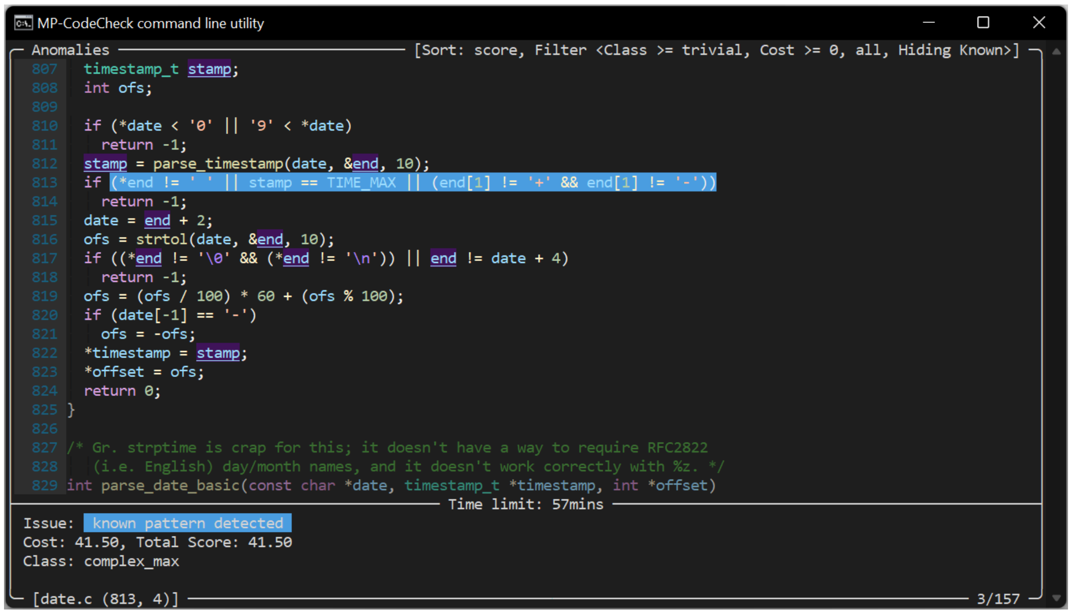
Models Path: The models path can be modified in the JSON file, and is initially set to:

```
"models_path": "[initial install directory]\\models"
```

Log Path: The log path can be modified in the JSON file, and is initially set to:

```
"log_path": "[initial install directory]\\logs"
```

Settings:
"run_training" – Determines whether or not training should be run before inference on the source code (defaults to true).
"filter" – Determines if items (such as nested_ifs) are extracted (defaults to true).



```
─ Anomalies ──────────────────────── [Sort: score, Filter <Class >= trivial, Cost >= 0, all, Hiding Known>] ─
807   timestamp_t stamp;
808   int ofs;
809
810   if (*date < '0' || '9' < *date)
811     return -1;
812   stamp = parse_timestamp(date, &end, 10);
813   if (*end != ' ' || stamp == TIME_MAX || (end[1] != '+' && end[1] != '-'))
814     return -1;
815   date = end + 2;
816   ofs = strtol(date, &end, 10);
817   if ((*end != '\0' && (*end != '\n')) || end != date + 4)
818     return -1;
819   ofs = (ofs / 100) * 60 + (ofs % 100);
820   if (date[-1] == '-')
821     ofs = -ofs;
822   *timestamp = stamp;
823   *offset = ofs;
824   return 0;
825 }
826
827 /* Gr. strptime is crap for this; it doesn't have a way to require RFC2822
828    (i.e. English) day/month names, and it doesn't work correctly with %z. */
829 int parse_date_basic(const char *date, timestamp_t *timestamp, int *offset)
──────────────────────────────────────────── Time limit: 57mins ─────────────
 Issue:  known pattern detected
 Cost: 41.50, Total Score: 41.50
 Class: complex_max

─ [date.c (813, 4)] ─────────────────────────────────────────────── 3/157 ─
```

Figure 15. An example of changing the expression highlight color in MP-CodeCheck's Code View.

7.3 File and Folder Exclusion

If you wish to omit files or folders from MP-CodeCheck's inference, you can use MPCC's file and folder exclusion command. This feature is run during the inference step and is triggered by the -X command for directories (uppercase X) and -x for files (lowercase X). As an example, one might trigger it using the following format:

MacOS, Linux
```
./MPCC infer -D [code directory] -X [directories to exclude] -x [files to exclude]
```

Windows
```
MPCC.exe infer -D [code directory] -X [directories to exclude] -x [files to exclude]
```

MPCC's file and directory exclusion command uses wildcards. The following are some example usages:

- `-x *api*` (lowercase x, excludes all files with 'api' in the name)
- `-X */out/*` (uppercase X, excludes all folders that contain a folder 'out' in the name)
- `-x */internal` (lowercase x, excludes all files with 'internal' as the last folder name)

For more details launch MPCC with the 'usage' command.

THE MACHINE PROGRAMMING COMPANY