Trucov

The True C and C++ Test Coverage Analysis Tool

for Schweitzer Engineering Laboratories

The Problem

If someone were to write a unit test with the goal of 90% coverage over the code, how is this 90% coverage verified? Not reaching test coverage goals can result in occurrences of untested code and may lead to error-prone software. Engineers need a standardized way to check the test coverage of functions, source files, and programs. This standardized process is Trucov.

What is Trucov?

An open source coverage analysis tool that:

- Indicates the areas of a program not exercised by a set of unit tests
- Analyses the branch coverage of unit tests after execution to assure quality
- Displays the control flow of a program and its test coverage information
- Works with the GCC compiler

Why use Trucov?

- Free
- Fast
- Easy to use

Just compile your code with the "-fprofile-arcs -ftest-coverage" flags, run your tests, and run Trucov.

The Trucov Team

Team Members:

- Ye Kyaw : Lead Programmer.
- Matthew Miller : Team Lead and Architect.
- William Reinhardt : Spec Writer and Lead Programmer.

Project Mentors:

- Jack Hagemeister : Instructor
- Nick Terry : Mentor

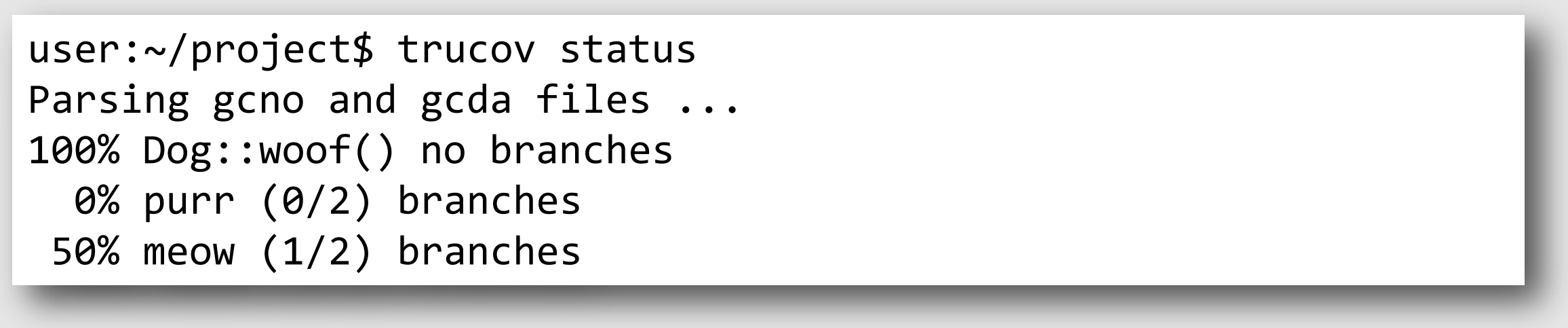
Key Features

- Produces coverage analysis on a per source, per function, and per branch level
- Provides both text and graphical coverage reports that engineers can easily use
- Automatically finds all the source files inside of a project
- Detects how many times any particular block or branch of code has been executed
- Generates the control flow graphs of each function per source file
- Optionally hides coverage information of external libraries to better focus on the product under test
- Allows user to narrow the scope of coverage information by specifying specific functions or files

----

Command Line

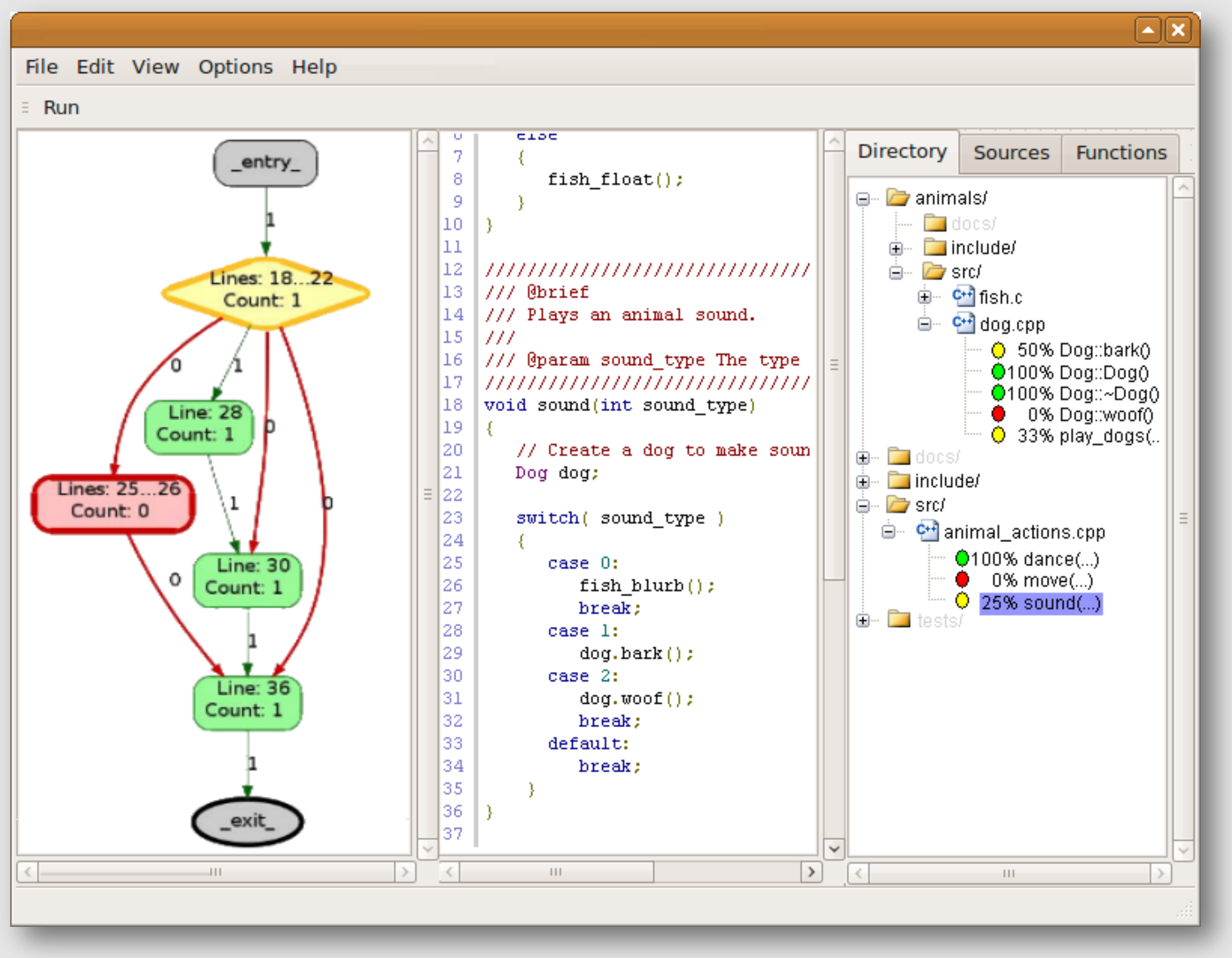
Trucov offers a command line version with over seven different commands for more advanced users.



```
user:~/project$ trucov status
Parsing gcno and gcda files ...
100% Dog::woof() no branches
  0% purr (0/2) branches
 50% meow (1/2) branches
```

Graphical Interface

The Trucov Graphical Interface displays a functions source code and coverage graph side by side. Allowing the user to visual the test coverage over the source code.

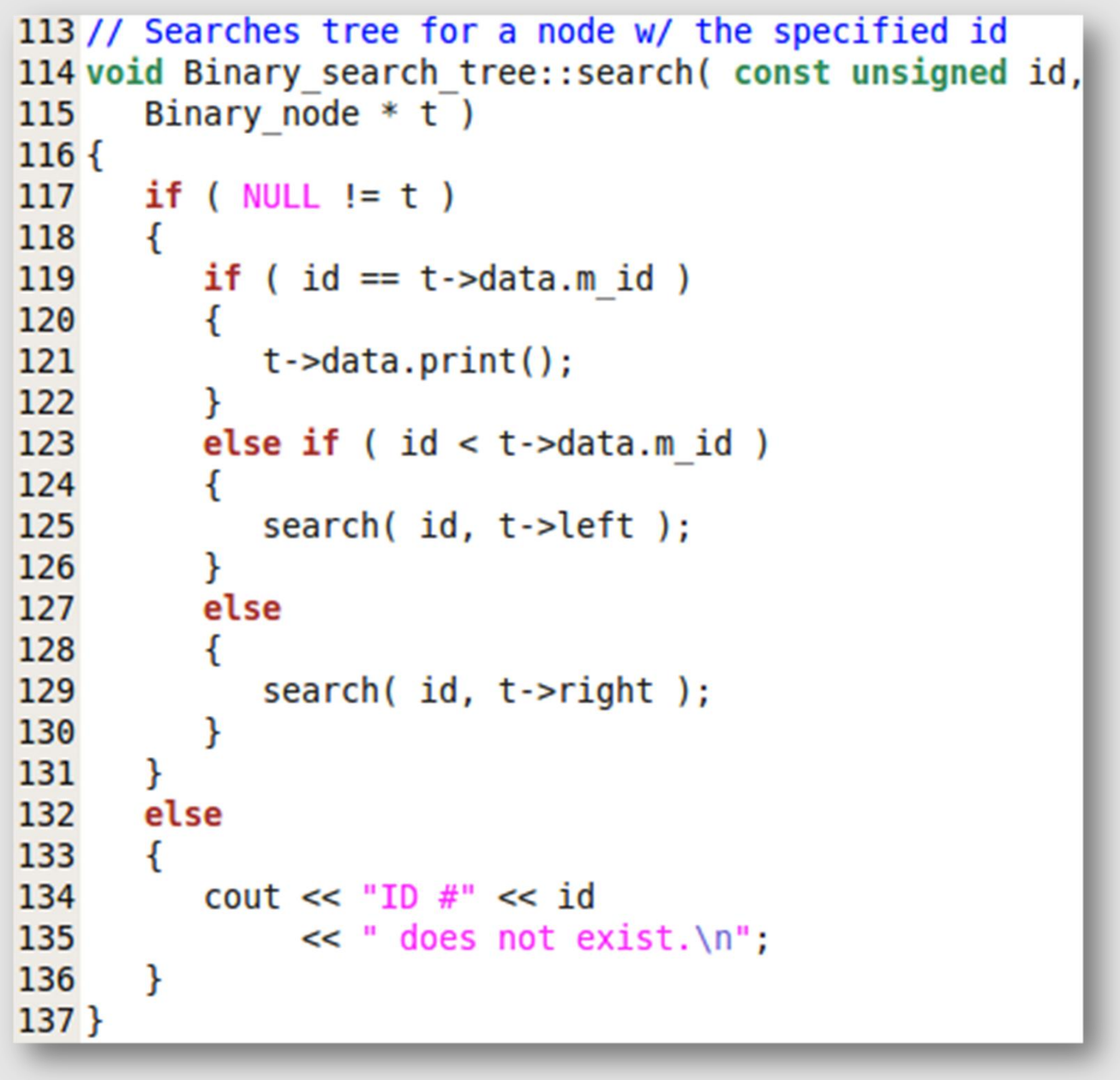


```
    else
    {
        fish_float();
    }
}

//////////////////////////////////
/// @brief
/// Plays an animal sound.
///
/// @param sound_type The type
//////////////////////////////////
void sound(int sound_type)
{
    // Create a dog to make soun
    Dog dog;

    switch( sound_type )
    {
        case 0:
            fish_blurb();
            break;
        case 1:
            dog.bark();
        case 2:
            dog.woof();
            break;
        default:
            break;
    }
}
```

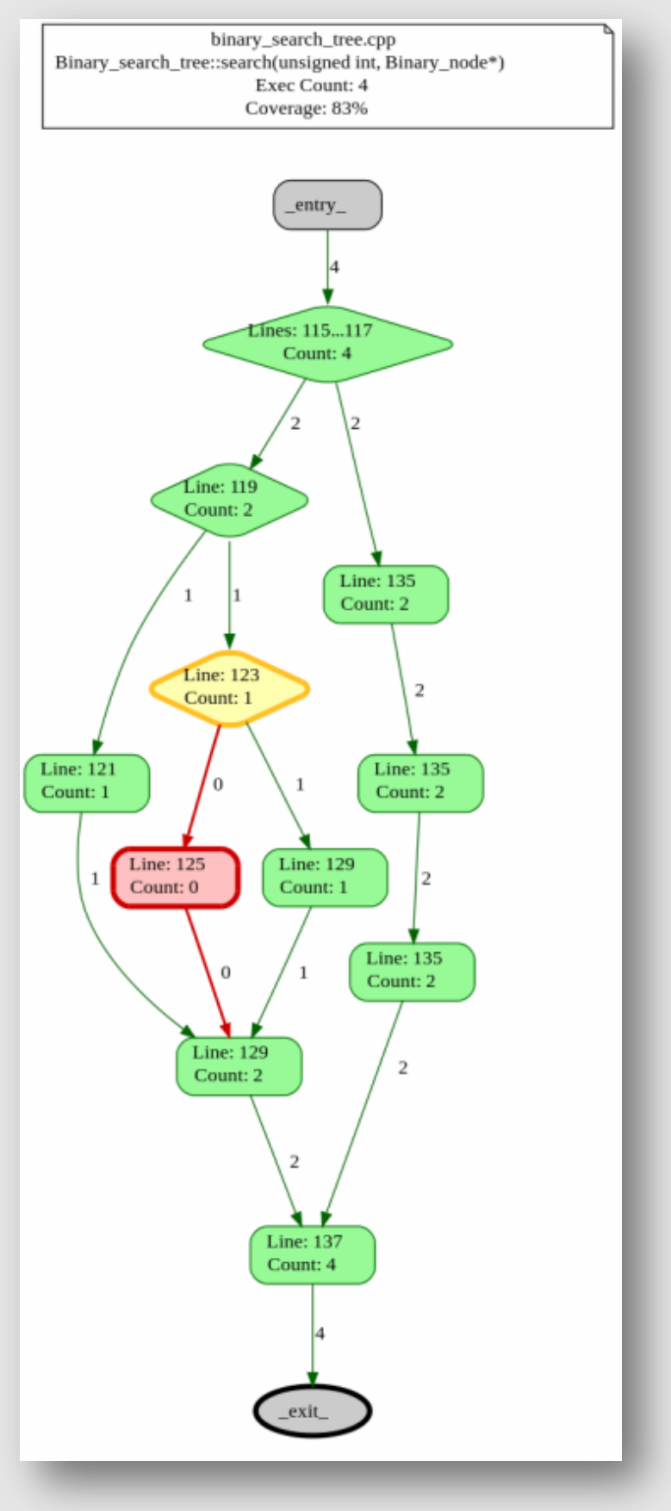
Control Flow Graphs

Trucov can create control flow graphs that represent that paths that might be traversed during the execution of the user's program. This is an excellent way of visualizing both the structure of a source file's functions and also the thoroughness of a project's unit tests.

The user can, at a glance, determine which conditions have never been satisfied and which lines of code have never been executed by use of simple color coding.



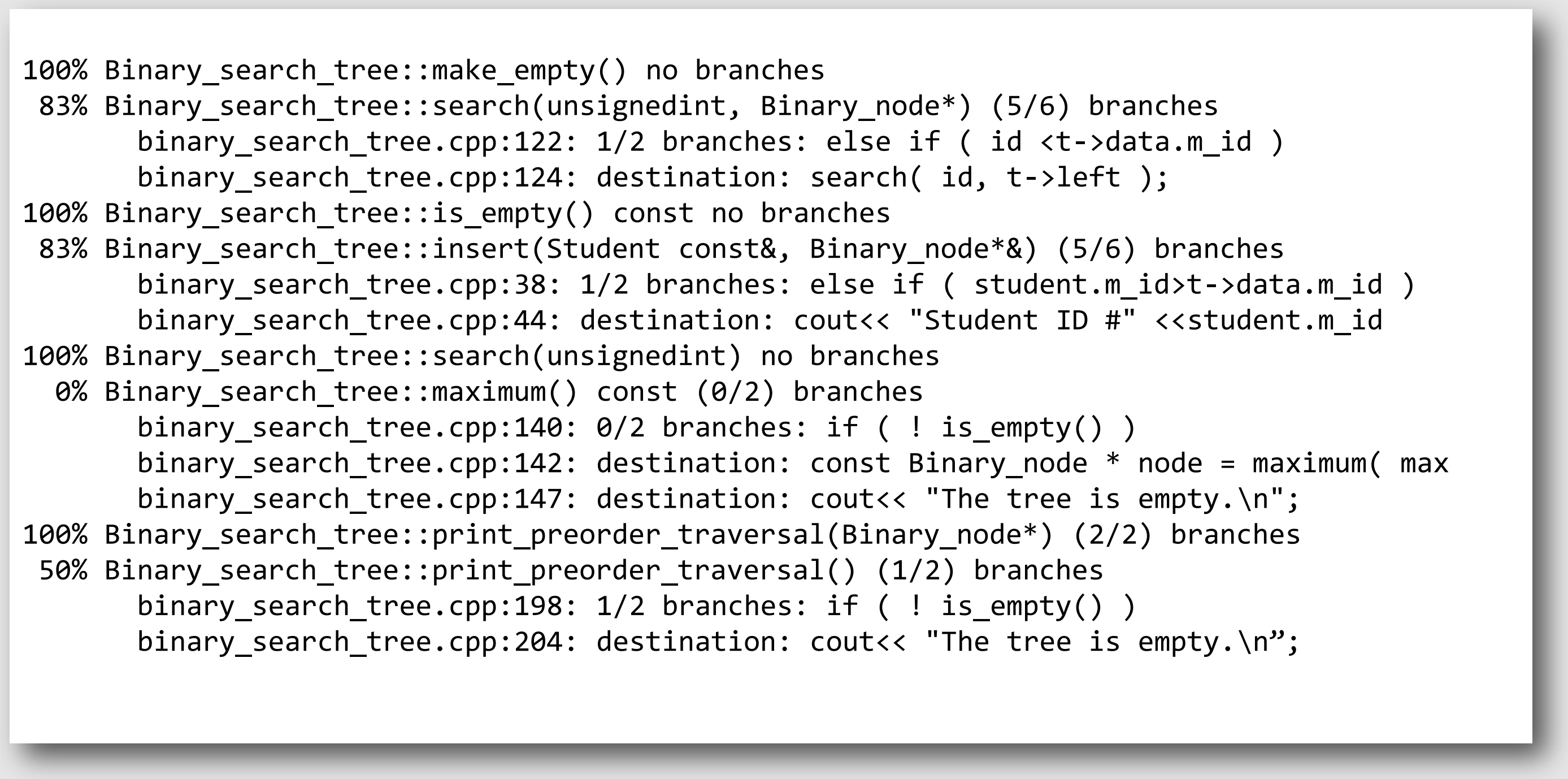
```
113 // Searches tree for a node w/ the specified id
114 void Binary_search_tree::search( const unsigned id,
115    Binary_node * t )
116 {
117    if ( NULL != t )
118    {
119       if ( id == t->data.m_id )
120       {
121          t->data.print();
122       }
123       else if ( id < t->data.m_id )
124       {
125          search( id, t->left );
126       }
127       else
128       {
129          search( id, t->right );
130       }
131    }
132    else
133    {
134       cout << "ID #" << id
135            << " does not exist.\n";
136    }
137 }
```



Code Coverage

Trucov can generate coverage reports with information per source file which offer another way for the user to easily verify the code coverage of the functions contained within.

Besides the summary information of each function, Trucov can additionally alert the user of any branches and lines of code that were never executed. Trucov reports the missing branch and destination information while also providing the exact line number and associated content of the source file.



```
100% Binary_search_tree::make_empty() no branches
 83% Binary_search_tree::search(unsignedint, Binary_node*) (5/6) branches
      binary_search_tree.cpp:122: 1/2 branches: else if ( id <t->data.m_id )
      binary_search_tree.cpp:124: destination: search( id, t->left );
100% Binary_search_tree::is_empty() const no branches
 83% Binary_search_tree::insert(Student const&, Binary_node*&) (5/6) branches
      binary_search_tree.cpp:38: 1/2 branches: else if ( student.m_id>t->data.m_id )
      binary_search_tree.cpp:44: destination: cout<< "Student ID #" <<student.m_id
100% Binary_search_tree::search(unsignedint) no branches
  0% Binary_search_tree::maximum() const (0/2) branches
      binary_search_tree.cpp:140: 0/2 branches: if ( ! is_empty() )
      binary_search_tree.cpp:142: destination: const Binary_node * node = maximum( max
      binary_search_tree.cpp:147: destination: cout<< "The tree is empty.\n";
100% Binary_search_tree::print_preorder_traversal(Binary_node*) (2/2) branches
 50% Binary_search_tree::print_preorder_traversal() (1/2) branches
      binary_search_tree.cpp:198: 1/2 branches: if ( ! is_empty() )
      binary_search_tree.cpp:204: destination: cout<< "The tree is empty.\n”;
```