Learning Report – Testing Driven Life Cycle



Test Driven Life Cycle Learning Report

# DETAILS:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared.**  **By** | **Reviewed By** | **To be Approved By** | **Remarks/Revision Details** |
| 1. | 02/11/20 | Sandya Devi Golakoti | Sowmya | Prithvi Shekar |  |
|  |
| 2. | 03/11/20 | Sandya Devi Golakoti | Unmesh | Prithvi Shekar |  |
|  |

**TEAM MEMBERS:**

|  |  |  |
| --- | --- | --- |
| **Name** | **PS Number** | **Email ID** |
| Gunturu Sowmya | 99002539 | [gunturu.sowmya@ltts.com](mailto:gunturu.sowmya@ltts.com) |
| Sandya Devi Golakoti | 99002651 | [sandyadevi.golakoti@ltts.com](mailto:sandyadevi.golakoti@ltts.com) |
| Rahul Rathod | 99002558 | [rahul.rathod@ltts.com](mailto:rahul.rathod@ltts.com) |
| Syed Mohammed Thammem | 99002543 | [syedmohammedthammem@ltts.com](mailto:syedmohammedthammem@ltts.com) |
| Unmesh Deshpande | 99002485 | [unmesh.deshpande@ltts.com](mailto:unmesh.deshpande@ltts.com) |
| Sampreeth Rayadurga | 99002500 | [sampreeth.rayadurga@ltts.com](mailto:sampreeth.rayadurga@ltts.com) |
| Om prakash C | 99002571 | [om.prakashc@ltts.com](mailto:om.prakashc@ltts.com) |

3 of 20

**L&T Technology Services CONFIDENTIAL**

Contents

[DETAILS: 2](#_Toc55299641)

[TABLE OF FIGURES 4](#_Toc55299642)

[TABLE OF TABLES 4](#_Toc55299643)

[ACTIVITY-1 5](#_Toc55299644)

[TESTING DEVELOPMENT LIFE CYCLE 5](#_Toc55299645)

[WHAT IS TDLC? 5](#_Toc55299646)

[WHY IS TDLC USED? 5](#_Toc55299647)

[WHERE IS TDLC USED? 5](#_Toc55299648)

[WHEN IS TDLC USED? 5](#_Toc55299649)

[HOW TO APPLY TDLC? 5](#_Toc55299650)

[ACTIVITY-2 7](#_Toc55299651)

[TEST DRIVEN DEVELOPMENT 7](#_Toc55299652)

[WHAT IS TDD? 7](#_Toc55299653)

[WHERE IS TDD USED? 8](#_Toc55299654)

[WHY IS TDD USED? 8](#_Toc55299655)

[WHEN TO USE TDD? 8](#_Toc55299656)

[HOW TO PERFORM TDD? 9](#_Toc55299657)

[References: 10](#_Toc55299658)

[ACTIVITY-3 11](#_Toc55299659)

[TESTING 11](#_Toc55299660)

[ACTIVITY- 4 12](#_Toc55299661)

[Implementation of TDLC and TDD mini project 12](#_Toc55299662)

[Implementation of TDLC 12](#_Toc55299663)

[Implementation of TDD 14](#_Toc55299664)

[VS CODE CONFIGURATION FOR C++/PYTHON PROJECT 22](#_Toc55299665)

[C++ PROJECTS: 22](#_Toc55299666)

[PYTHON PROJECT: 28](#_Toc55299667)

# TABLE OF FIGURES

[Figure 1 TDD stages 7](#_Toc55314455)

[Figure 2 Steps to perform TDD Tests 9](#_Toc55314456)

[Figure 3 Source Code 12](#_Toc55314457)

[Figure 4 Test Cases execution 13](#_Toc55314458)

[Figure 5 Source code after debug 13](#_Toc55314459)

[Figure 6 Passed Testcases 14](#_Toc55314460)

[Figure 7 Test code 1 (Compile Time Error) 15](#_Toc55314461)

[Figure 8 Test code1 along with Production code (Run Time Error) 16](#_Toc55314462)

[Figure 9 Test Code1 along with Production Code (working successfully) 17](#_Toc55314463)

[Figure 10 Refactored Production Code with Test code 1 (Working Successfully) 18](#_Toc55314464)

[Figure 11 Test Code 2 (Compile Time Error) 19](#_Toc55314465)

[Figure 12 Test Code2 along with Production Code (working successfully) 20](#_Toc55314466)

[Figure 13: Refactored Production Code with Test code 2 (Working Successfully) 21](#_Toc55314467)

# TABLE OF TABLES

[Table 1 Comparison between test objectives, basis and objects 12](#_Toc55314489)

[Table 2 Failed Test Cases before debug 13](#_Toc55314490)

[Table 3 Passed Test Cases 14](#_Toc55314491)

# ACTIVITY-1

# TESTING DEVELOPMENT LIFE CYCLE

WHAT IS TDLC?

TDLC is life cycle of testing. If testing is not planned and not done in a particular process it may not only take time to be completed but also will increase the cost

WHY IS TDLC USED?

TDLC helps make the testing process more sophisticated, consistent and effective. You can include milestones and deliverables for each step of the project. Easy to understand and implement even if the model is expanded to various levels.

WHERE IS TDLC USED?

Availability of requirement Document, acceptance criteria and application architectural document.

WHEN IS TDLC USED?

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However, in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

HOW TO APPLY TDLC?

TDLC is applied in phases as follows:

1. Requirement Analysis:

Requirement Analysis is the first step of Testing Driven Life Cycle (TDLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then quality assurance team meets with the stakeholders to better understand the detail knowledge of requirement.

1. Test Planning:

Test Planning is most efficient phase of software testing life cycle where all testing plans are defined.

In this phase manager of the testing team calculates estimated effort and cost for the testing work. This phase gets started once the requirement gathering phase is completed.

1. Test Case Development:

The test case development phase gets started once the test planning phase is completed. In this phase testing team note down the detailed test cases. Testing team also prepare the required test data for the testing. When the test cases are prepared then they are reviewed by quality assurance team.

1. Test Environment Setup:

Test environment setup is the vital part of the TDLC. Basically, test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process, the testing team is not involved. either the developer or the customer creates the testing environment.

1. Test Execution:

After the test case development and test environment setup test execution phase gets started. In this phase testing team start executing test cases based on prepared test cases in the earlier step.

1. Test Closure:

This is the last stage of TDLC in which the process of testing is analyzed.

# ACTIVITY-2

# TEST DRIVEN DEVELOPMENT

WHAT IS TDD?

* + - TDD stands for Test Driven Development which means letting your tests drive your development.
    - In TDD, developers write a test before writing the production code to fulfill the test.
    - TDD is an iterative development process. It is a rapid cycle of testing, coding and refactoring.
    - TDD is related to test first programming evolved as a part of extreme programming concepts.

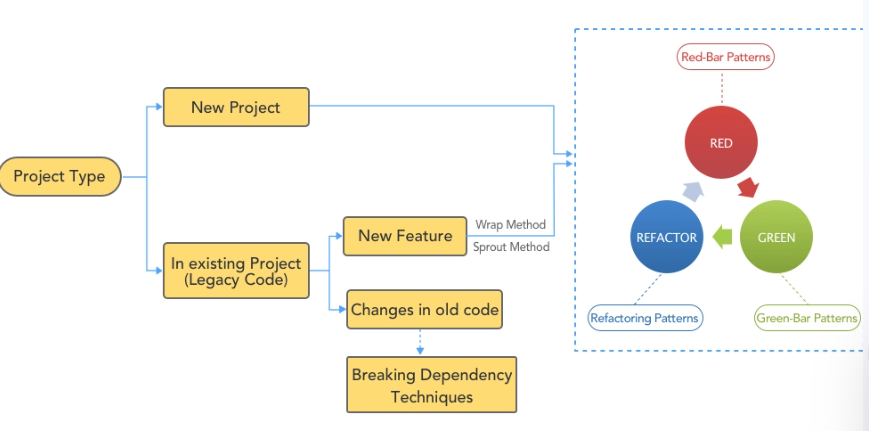


Figure 1 TDD stages

* + - The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication of code as we write a small amount of code at a time to pass tests.

WHERE IS TDD USED?

* + - Test driven development is one of the common practices of Agile core development.
    - It is acquired from the Agile manifesto principles and Extreme programming.
    - TDD does not replace traditional testing, instead it defines a proven way to ensure effective unit testing.
    - In agile, by developing the tests from requirements rather than the code, communication increases. The creator of the requirements, developer and tester must collaborate on the tests and subsequent code.
    - TDD works incredibly well in practice and it is something that all software developers should consider adopting.

WHY IS TDD USED?

* + - TDD encourages the development of simple, clean and meaningful code.
    - TDD helps developers towards simple designs; keeps things typically object oriented structured, pushes developers towards separated components.
    - For example, assume you add some new functional code, compile, and test it. Chances are pretty good that your tests will be broken by defects that exist in the new code.
    - It is much easier to find, and then fix, those defects if you've written two new lines of code than two thousand.
    - Higher test density and test coverage are the default advantages of TDD.

WHEN TO USE TDD?

Before choosing to use TDD in an existing project, you must consider the unique context of your project in detail. Ask yourself these questions and then decide:

* + - How much will TDD increase the cost of implementing your functionality?
    - What kind and how many automated tests do you plan to create?
    - Will TDD be used correctly? Will TDD simplify your production code or make it more complex.
    - TDD lends itself really well to when you have a pure logic function that you need to write. When the work you need to do has a clearly defined set of expected inputs and outputs.

# 

HOW TO PERFORM TDD?

TDD can be performed by following the steps below:

* + - ADD TEST: Write enough failing test code (include compile time and runtime failures).
    - RUN TEST: Tests fail as we did not implement the code.
    - WRITE CODE: Implement the production code to pass the failing tests.
    - REFACTOR: Refactor production code and verify the same with existing code.
    - Repeat the process.



Figure 2 Steps to perform TDD Tests

References:

1. TDD in Agile - <http://agiledata.org/essays/tdd.html>
2. Stages of Test-Driven Development - <https://www.guru99.com/test-driven-development.html>

# ACTIVITY-3

# TESTING

|  |  |  |  |
| --- | --- | --- | --- |
|  | Unit Testing | Integration Testing | Acceptance Testing |
| Objectives | 1.The objective of unit tests is to isolate the smallest testable parts of an application and verify that it works in isolation. | 1.The objective of integration test is to verify the integrated components working correctly or not. | 1.The objective of the Acceptance testing is to provide the end-users the confidence that the system will work according to their expectations. |
| Basis | 2.The test basis could be code or component specifications. | 2.The test basis could be sequence diagrams or component level architecture. | 2.The test basis could be user or business requirements, use cases or user stories. |
| Outcomes | 3.In unit testing, object Code is Classes, component. | 3. InIntegration testing, the test object could be Interfaces, micro-services. | 3.In Acceptance testing, the test object could be reports, forms. |

Table 1 Comparison between test objectives, basis and objects

# ACTIVITY- 4

# Implementation of TDLC and TDD mini project

Implementation of TDLC

**Requirements Designed QA Team: -**

1)Extract Data

2)Calculate Debit

3)Calculate Credit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr. No. | Test Cases Description | Expected Output | Actual Output | Result |
| 1. | To Get Customer Id | 1001 | 1000 | Failed |
| 2. | To Get Customer Name | Lippman | Lippma | Failed |
| 3. | To Get Customer After Debit | 3800 | 4800 | Failed |
| 4. | To Get Total Amount After Credit | 8000 | 9000 | Failed |

Table 2 Failed Test Cases before debug

1.Write code according to requirements

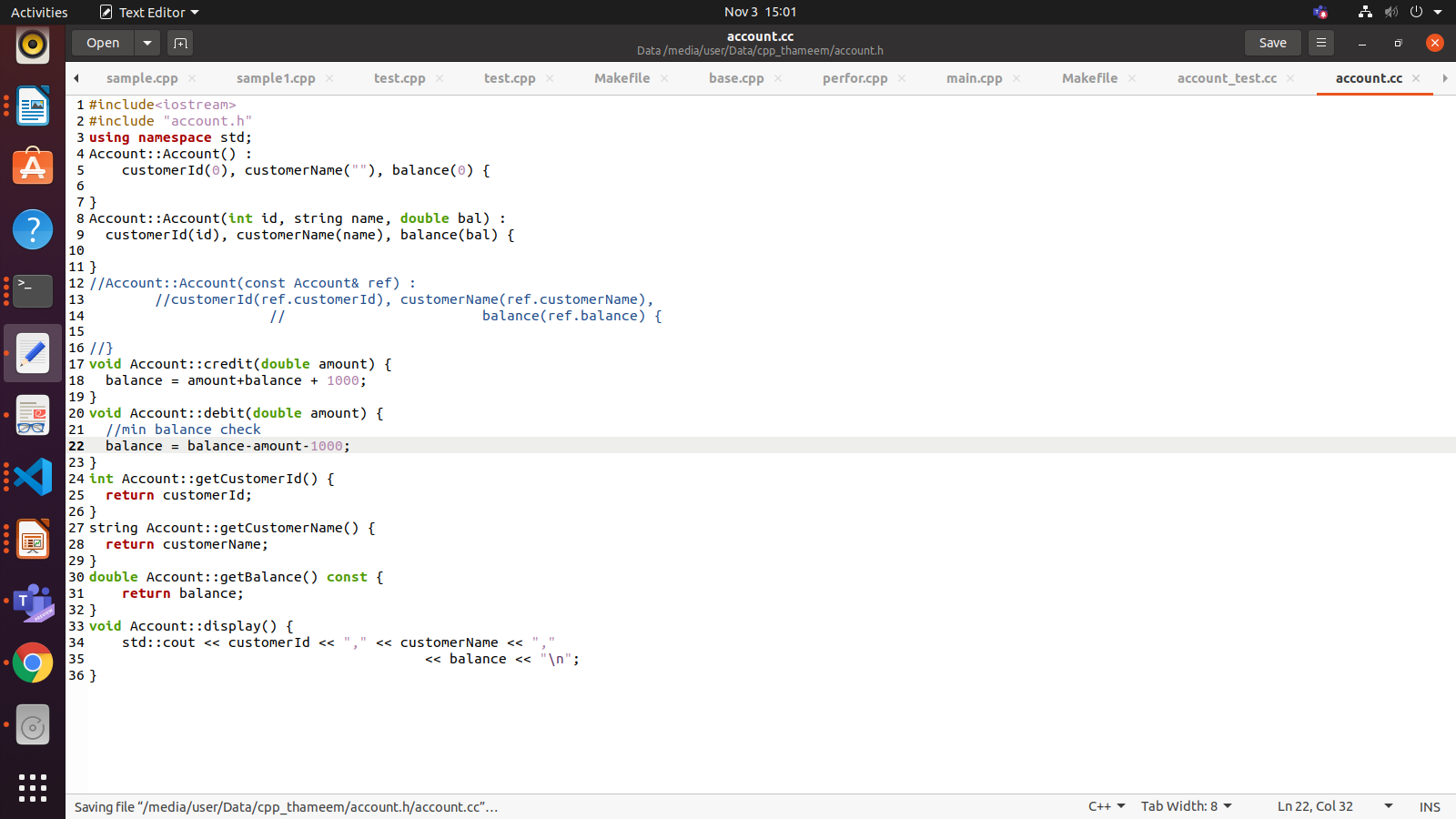
****

Figure 3 Source Code

2.Check the code output, the test case fails

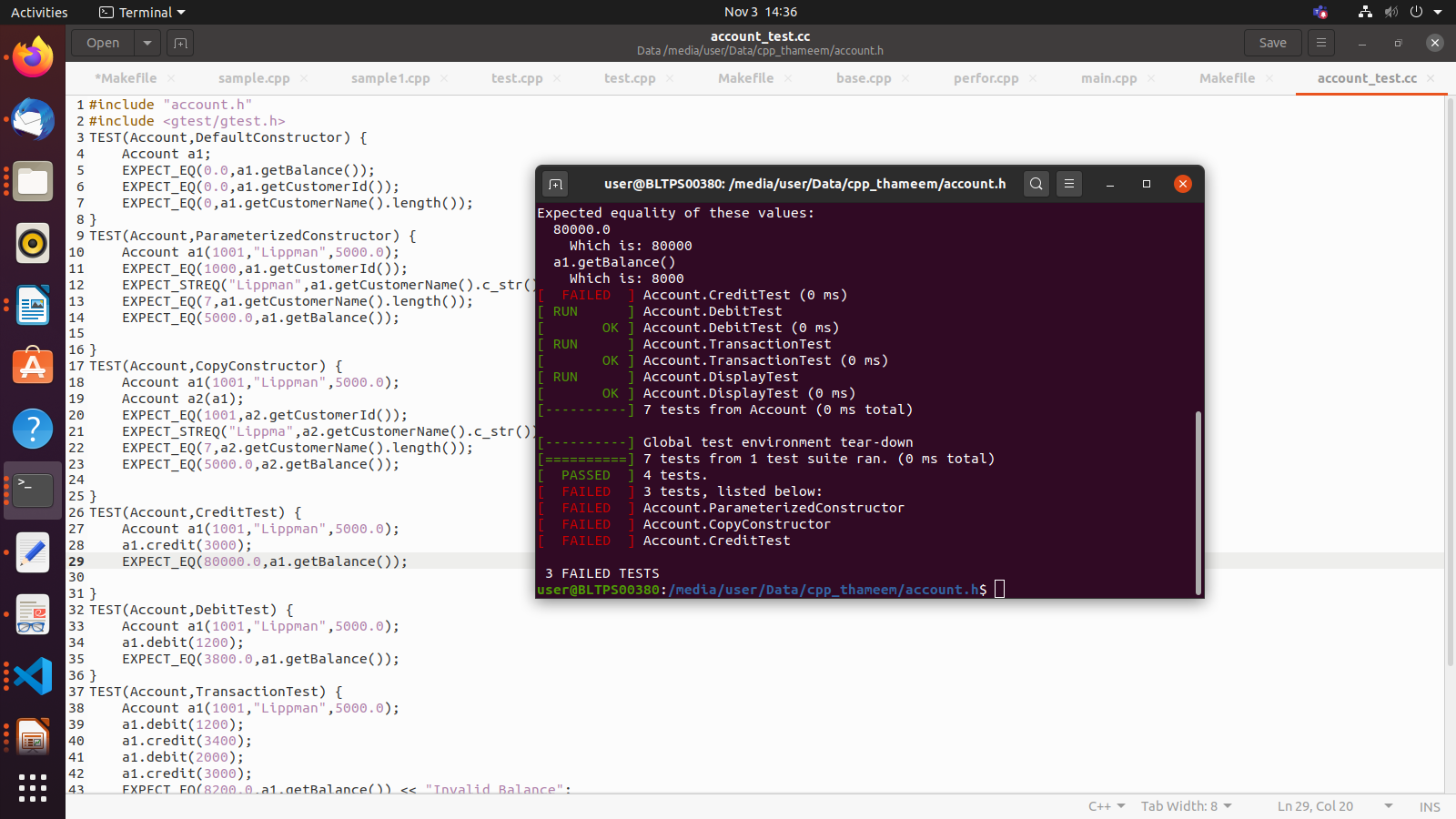
****

Figure 4 Test Cases execution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr. No. | Test Cases Description | Expected Output | Actual Output | Result |
| 1. | To Get Customer Id | 1001 | 1001 | Passed |
| 2. | To Get Customer Name | Lippman | Lippman | Passed |
| 3. | To Get Customer After Debit | 3800 | 3800 | Passed |
| 4. | To Get Total Amount After Credit | 8000 | 8000 | Passes |

Table 3 Passed Test Cases

3.Debug the code

****

Figure 5 Source code after debug

4. Confirm the Test Passes

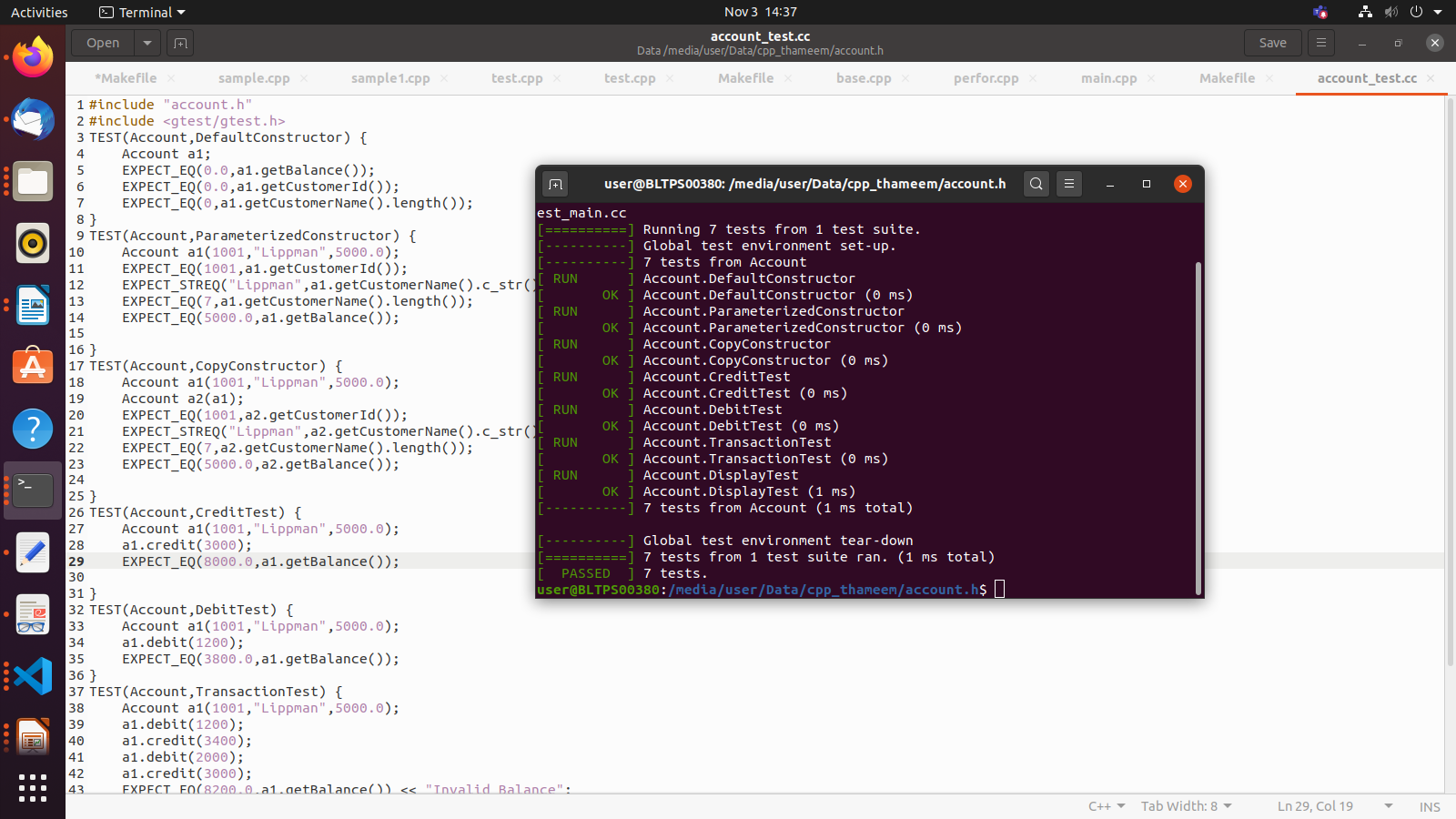
****

Figure 6 Passed Testcases

Implementation of TDD

Problem statement: Create Binary search tree in c++ using TDD

**Steps in TDD (Test Driven Development)**

**Step 1:** Write enough failing test code (Including compile time and runtime)

**Step 2:** Write production code to pass those failing Tests

**Step 3:** Refactor production code and verify the same with the existing test.

**Step 4:** Repeat (Go to Step 1) until all tests are passed and code is refactored.

Now let us write some failing test code:

**Step1:**  Here we write test code first, which is bound to fail as there is no production code.

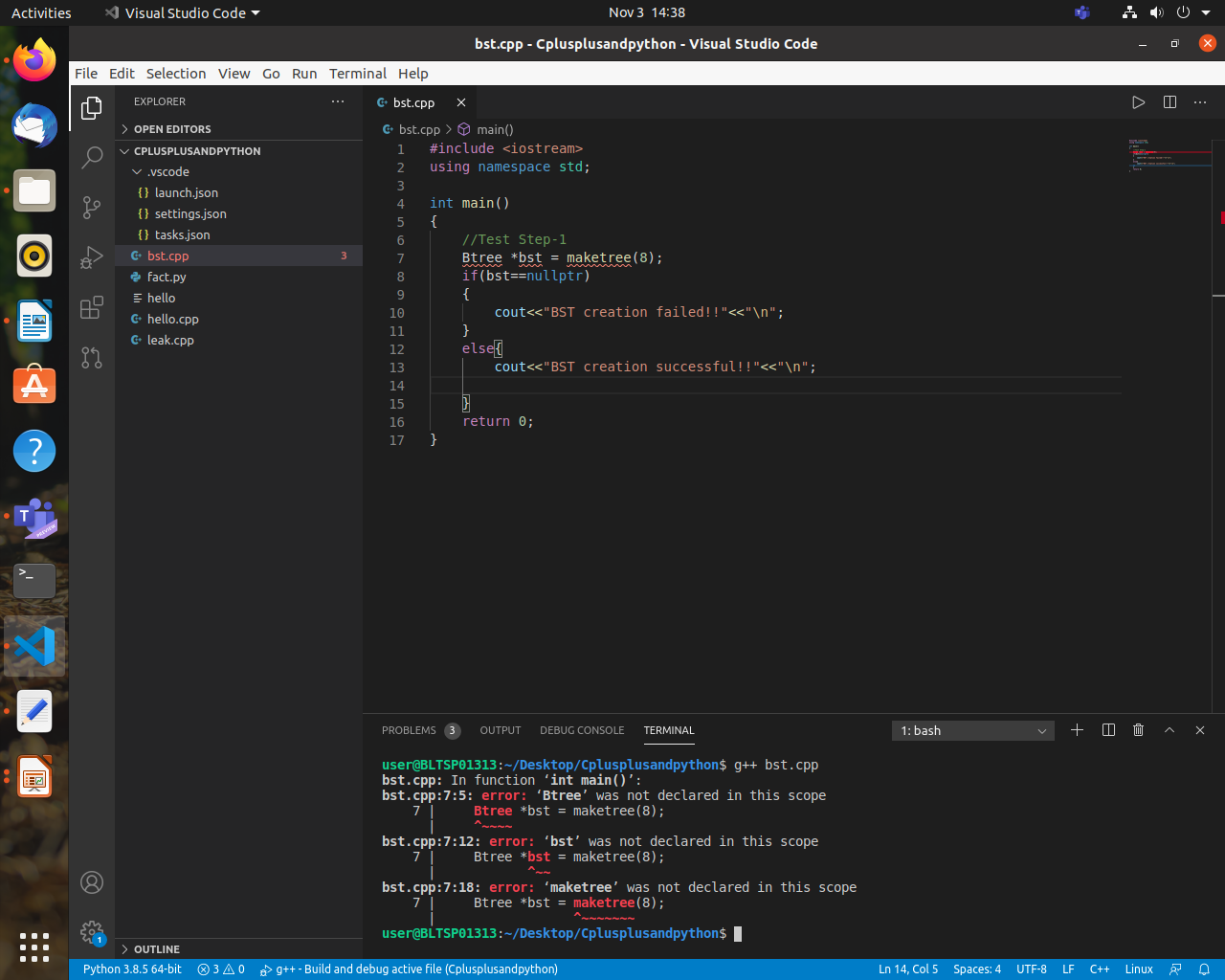
****

Figure 7 Test code 1 (Compile Time Error)

We can see that we have written test code which are failing as Btree and maketree are not defined. Here we are having compile time errors.

We can see that we have written test code which are failing as Btree and maketree are not defined. Here we are having compile time errors.

**Step2:** Now we shall write some production code to make sure we pass test code

****

Figure 8 Test code1 along with Production code (Run Time Error)

We have written sufficient production code to successfully compile the code, but here we have runtime error. So we have come from compile time error to runtime errors.

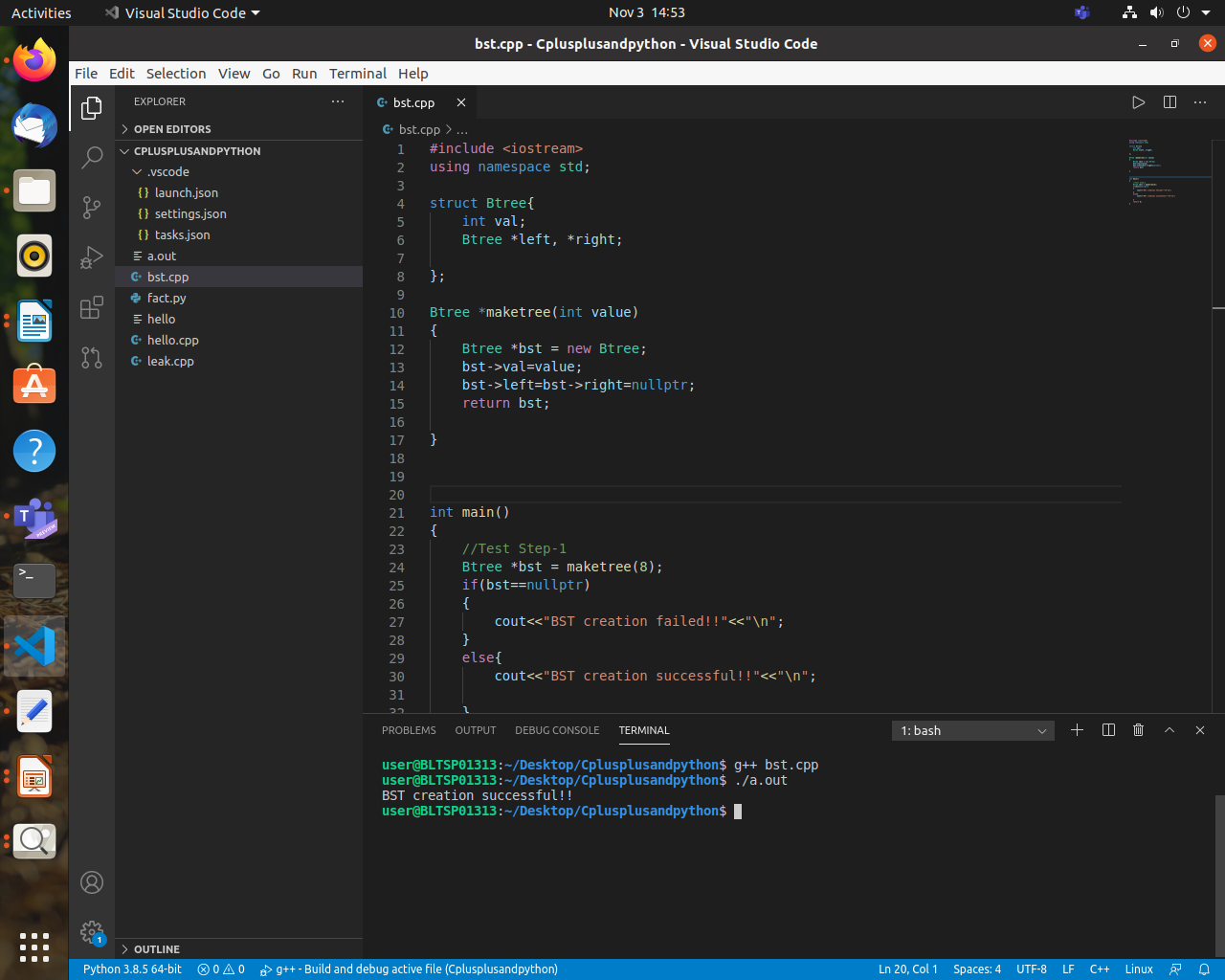
****

Figure 9 Test Code1 along with Production Code (working successfully)

We have executed the code successfully and passed the test code. We have solved runtime errors.

**Step 3:** We have to refactor the code (adding enhancements to the existing code), and we check if code successfully works:

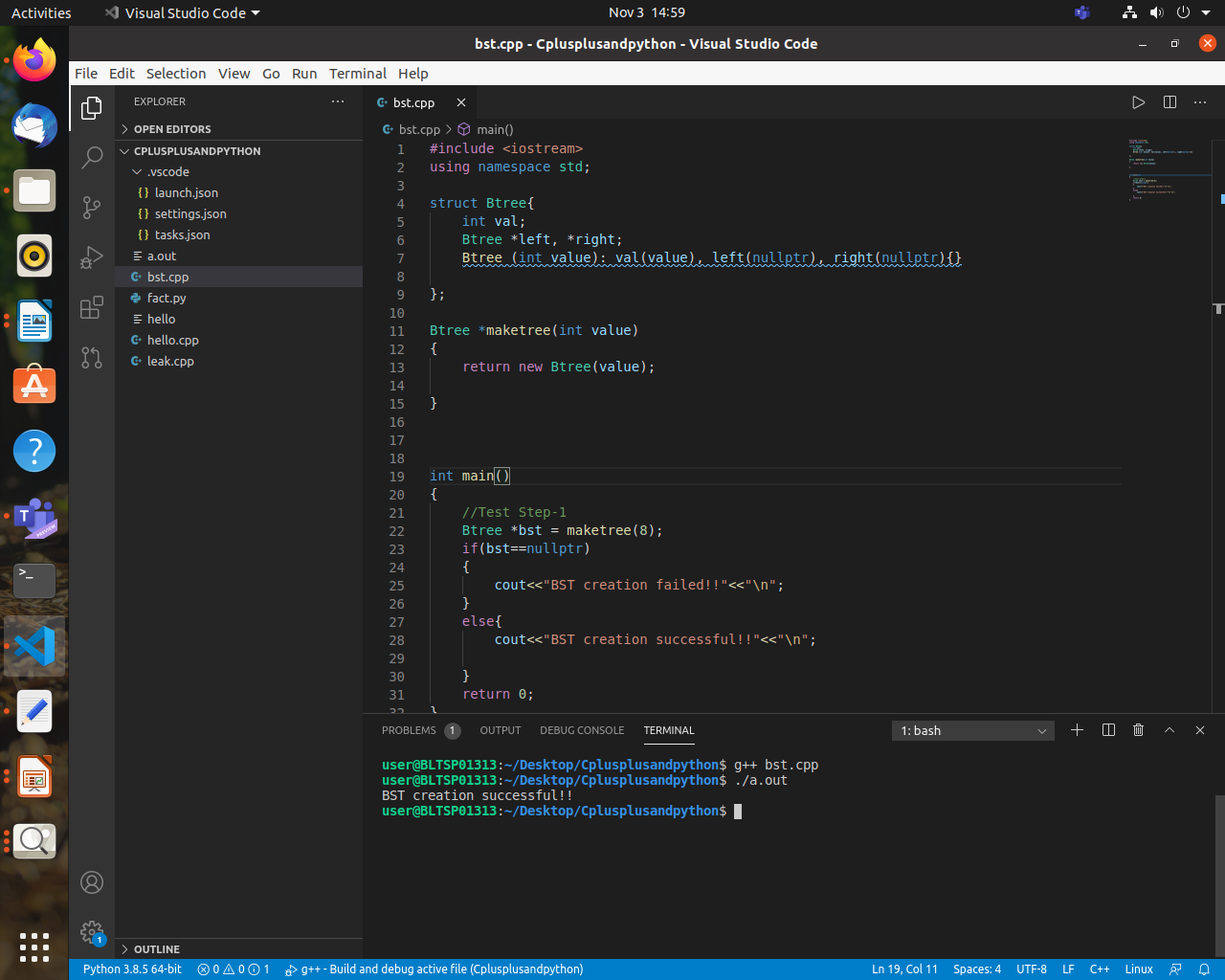
****

Figure 10 Refactored Production Code with Test code 1 (Working Successfully)

We can see that there are some changes made to production code to make it look better and some extra features are added. Code successfully compiled and then executed without any runtime errors.

After 3 steps are over, we are going to repeat these 3 steps again

***Step 1(2nd iteration):***

We have written test code and it has failed because there is no function defined as InsertNode.So we have to write production code to pass the test code.

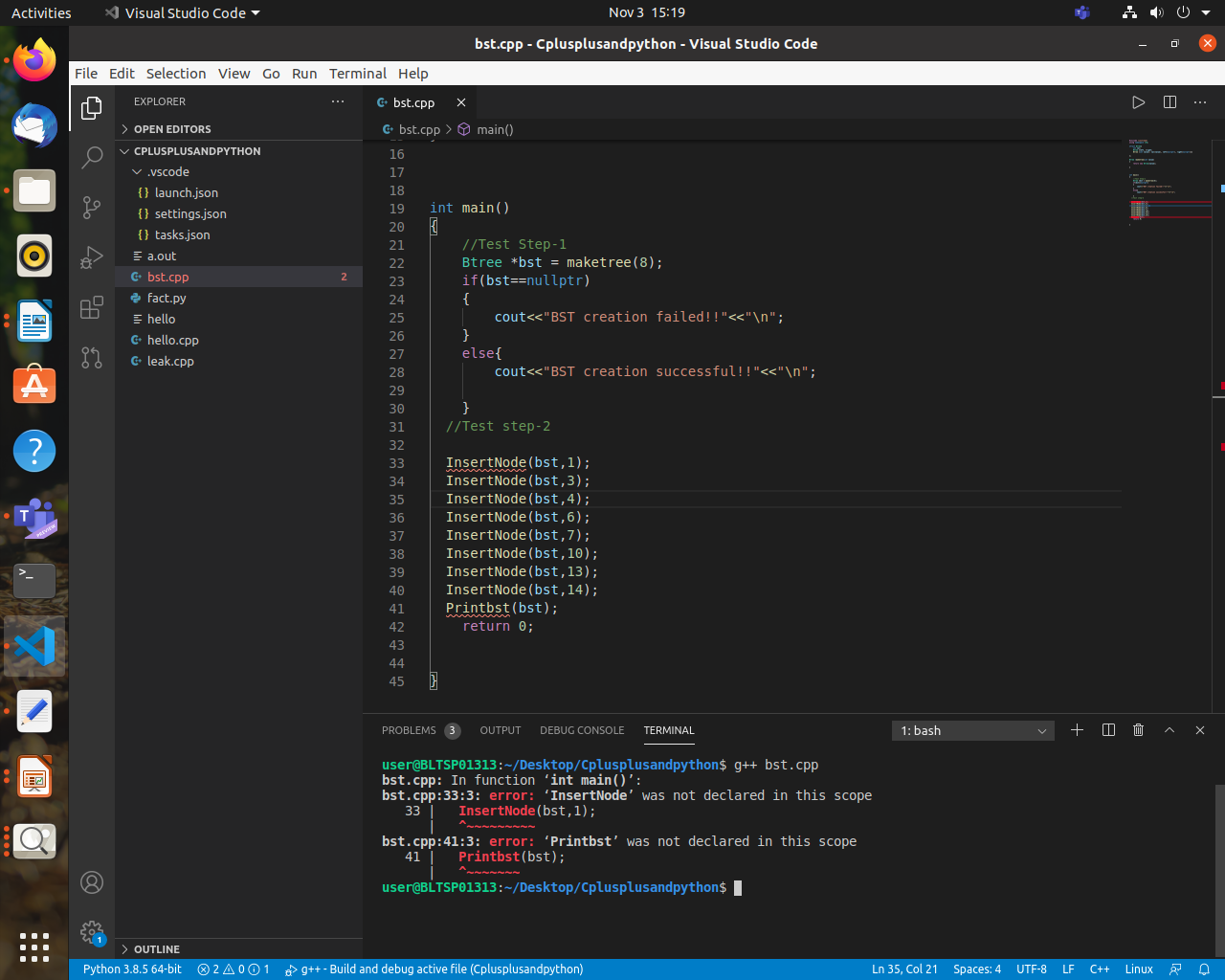
****

Figure 11 Test Code 2 (Compile Time Error)

***Step 2 (2nd iteration):***

Now we will write sufficient production code so that it passes the test code and executes successfully.



Figure 12 Test Code2 along with Production Code (working successfully)

***Step 3 (2nd iteration):*** Now we will refactor the code, i.e add some new features or enhancements to make the code look better and have better functionalities.

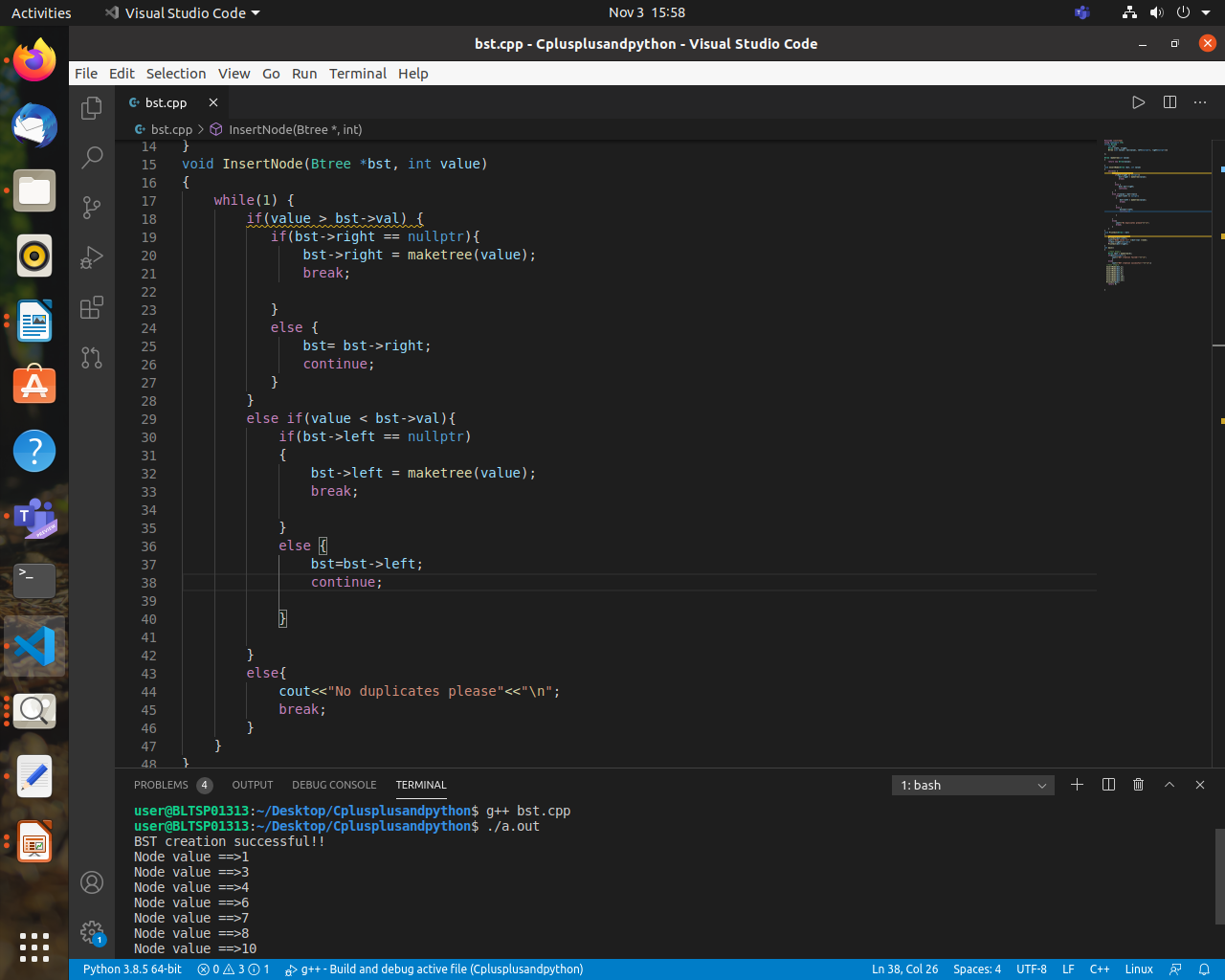


Figure 13: Refactored Production Code with Test code 2 (Working Successfully)

We have refactored the code, and it compiles and then executes successfully.

**Summary:**

This activity demonstrates Test Driven Development, where we write test code first and then fail it, and then write sufficient production code to make the test code work and then we will do refactoring which is basically enhancing the code to make it better and the steps are repeated.

1. TDLC (4W & 1H)

2. TDD (4W & 1H)

3. a. implementing TDLC on mini project (C++ and python project)

3.b Implementing TDD on mini project (C++ and python project)

Problem Statement: Create Binary search tree in C++ using TDD

4. Comparison on test objectives, test basis and test objects (on testing).

5. Configure tool (debugging, auto complete, symbol viewer, build, lint, dB)

# VS CODE CONFIGURATION FOR C++/PYTHON PROJECT

C++ PROJECTS:

1.DEBUGGING

Steps involved in debugging: -

1.Debugger extensions

VS Code has built-in debugging support,can debug any language.For debugging

any languages and runtimes (including PHP, Ruby, C#, Python, C++ and many

others), look for Debuggers extensions in VS Code marketplace or select Install

Additional Debuggers in the top-level Run menu.

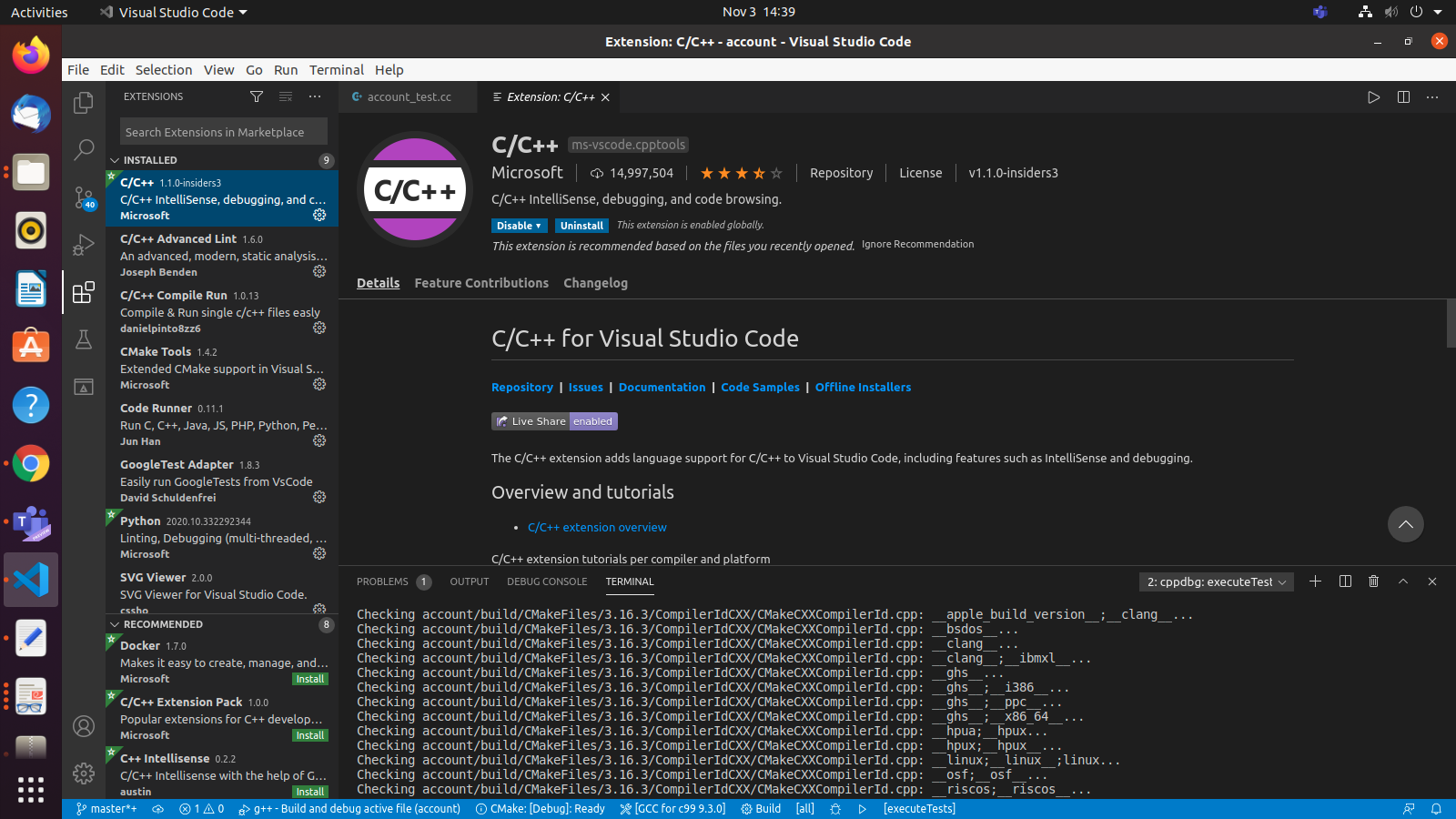


Figure 14 VS Code Extension for Debugger

2.Start Debugging

->To bring up the Run view, select the Run icon in the Activity Bar on the side of VS

Code or use the keyboard shortcut(ctrl+ shift + D).

-> The Run view displays all information related to running and debugging and has

a top bar with debugging commands and configuration settings.

3. Launch Configuration

->To run or debug a code in VS Code, press F5.

->VS Code automatically detect the debug environment, but if it fails, we have to

choose it manually.

->As soon as a debugging session starts, the DEBUG CONSOLE panel is displayed

and shows debugging output, and the Status Bar changes color.

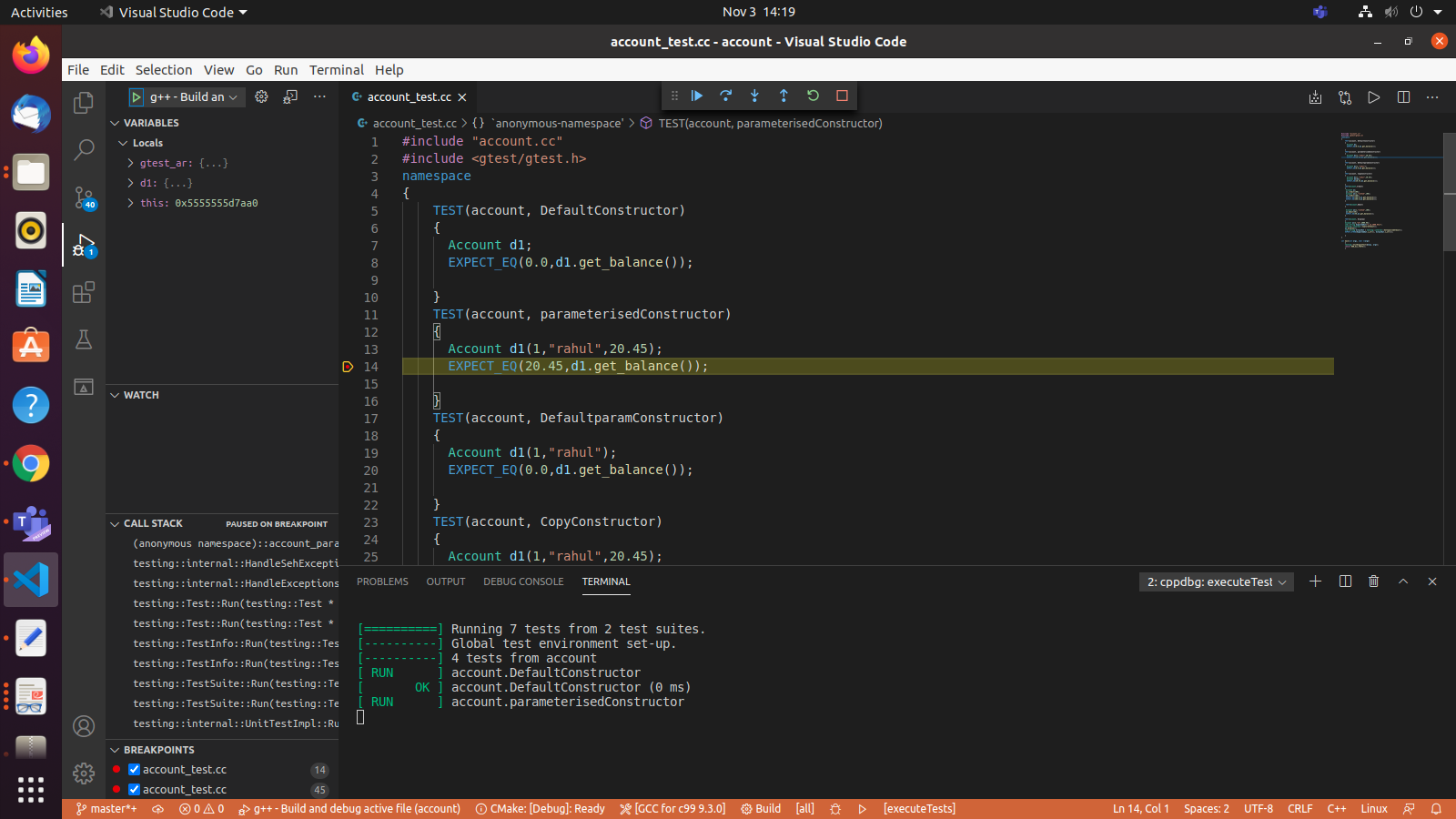


Figure 15 Debugger output

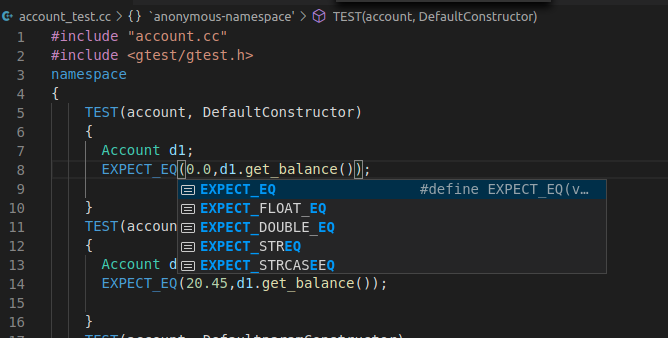
2.AUTO-COMPLETE   
 

Figure 16 Auto Complete

Auto Complete is built-in inside VS code by default.

3.Symbol Viewer

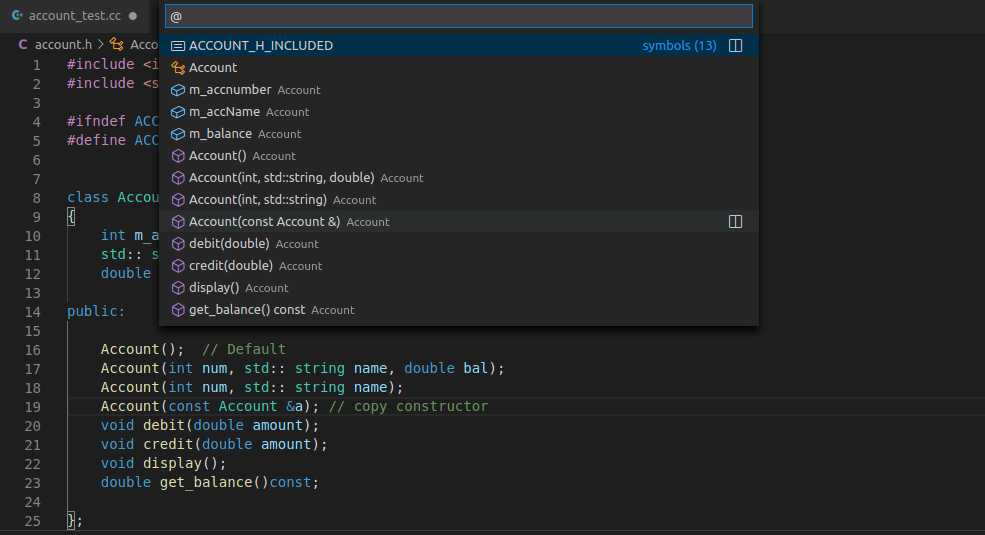


Figure 17 Symbol Viewer

Symbol viewer is built in vs code.We can navigate symbols inside a file with ctrl+shift+o.

4.Build

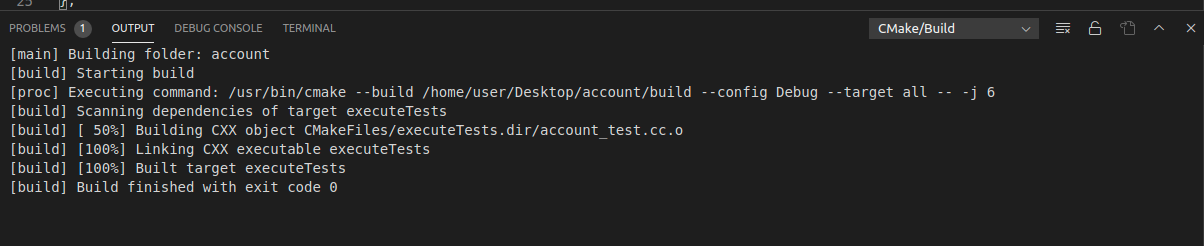
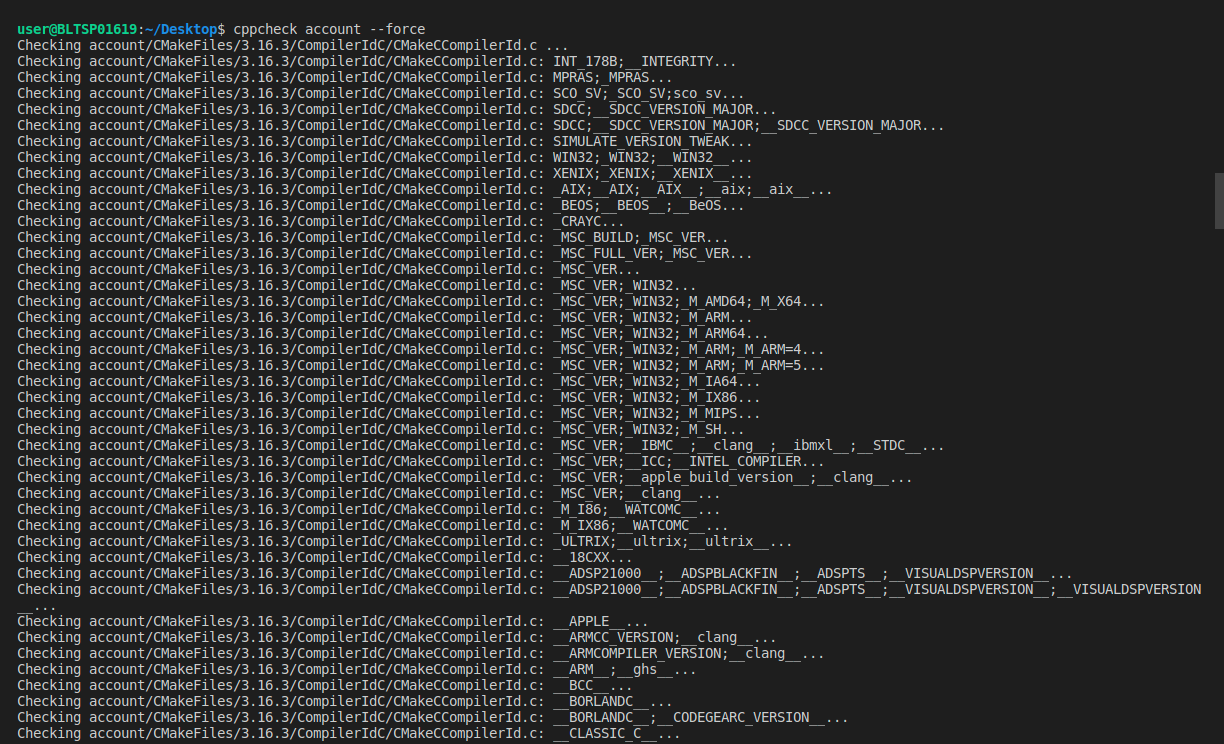


Figure 18 Build

5.CPP CHECK

The package used for C++ compilation & debugging is “c/c++ advanced lint”. It is used for static code analysis, unused variables, code quality, coding rules, etc.



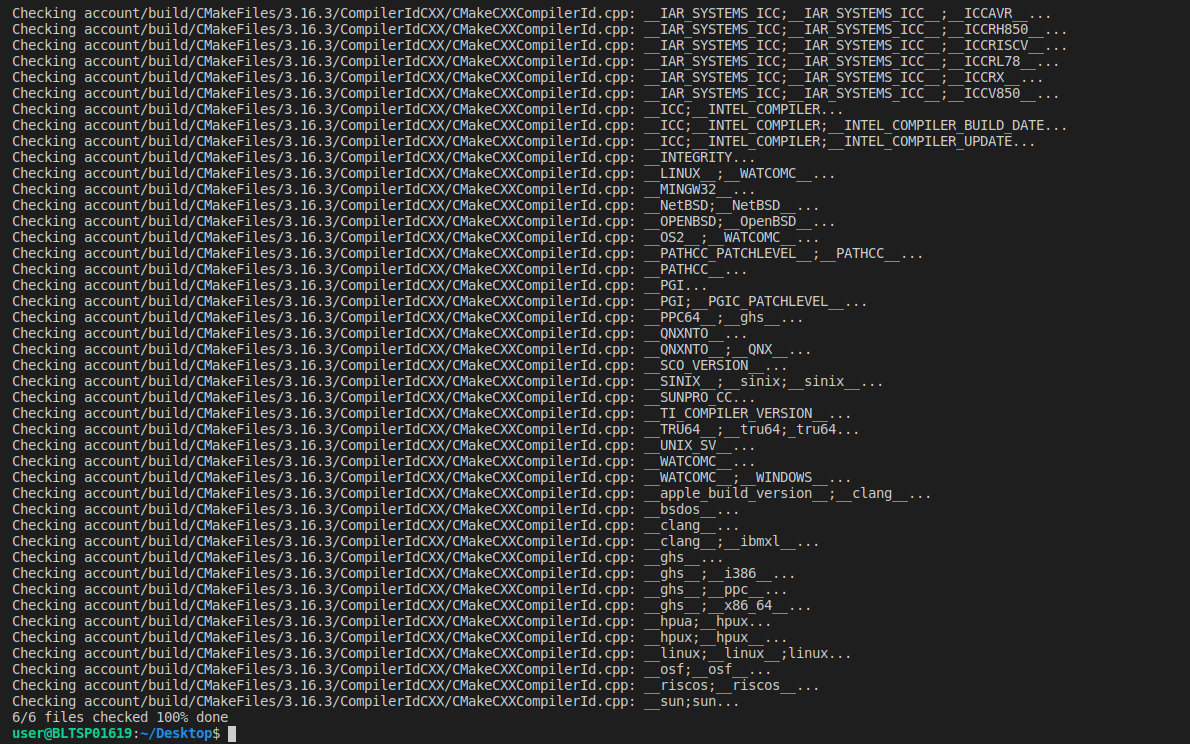


Figure 19 CppCheck

6.Valgrind

Valgrind is a programming tool for memory debugging, memory leak detection, and profiling.The package used for memory debugging is “vs code valgrind”

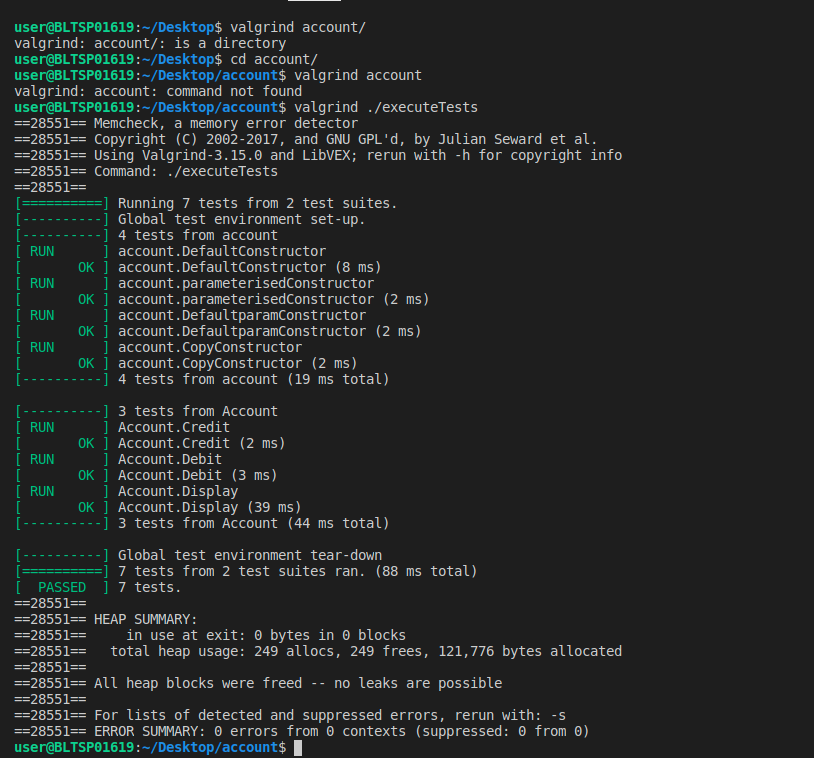


Figure 20Valgrind

REFERENCES:

1.<https://marketplace.visualstudio.com/items?itemName=jbenden.c-cpp-flylint>

2.<https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools>

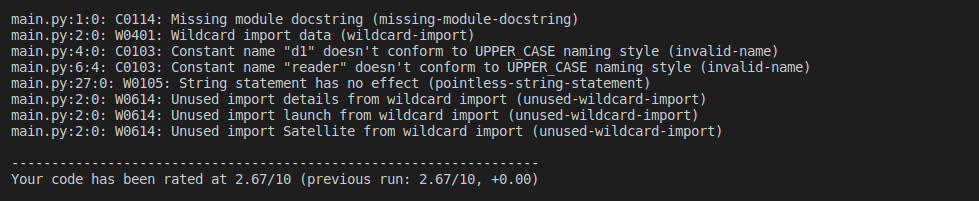
3.<https://marketplace.visualstudio.com/items?itemName=krosf.vscode-valgrind>

PYTHON PROJECT:

1.PYLINT

Pylint is a Python static code analysis tool which looks for programming errors, helps enforcing a coding standard, sniffs for code smells and offers simple refactoring suggestions.

It’s highly configurable, having special pragmas to control its errors and warnings from within your code, as well as from an extensive configuration file.



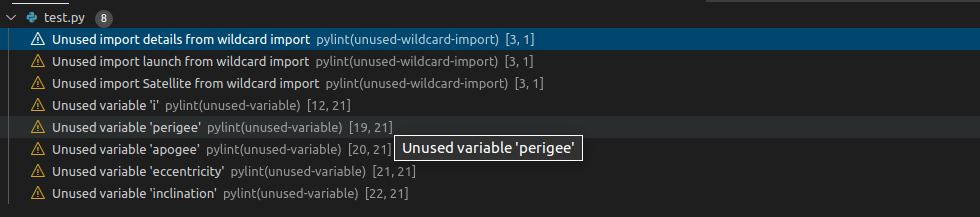


Figure 21Pylint

REFERENCE

1.<https://code.visualstudio.com/docs/python/linting>.

PYTHON PROJECT

* DEBUGGING



The package used for C++ compilation & debugging is “dgb-gdb”

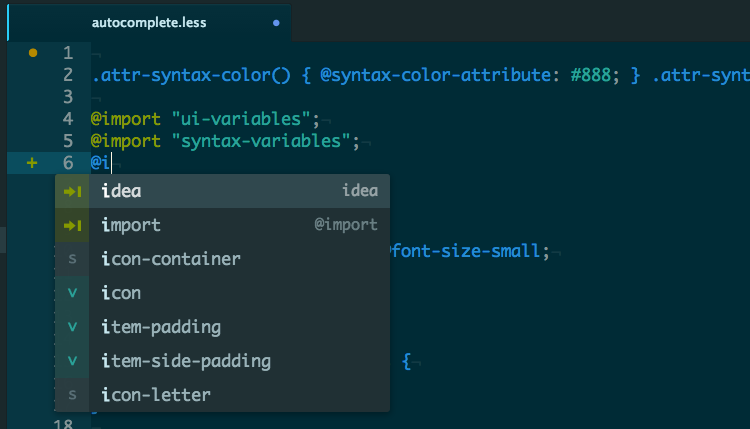
Compiling steps in Atom:

1. Write the code and save it
2. Right click and select “Compile Run” or press F6

Debugging steps in Atom:

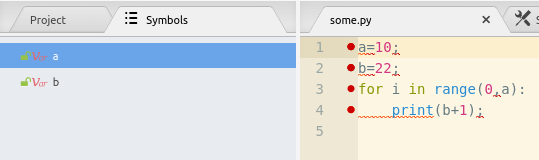
1. Right click on an executable in the treeview, select “Debug this file”, and click Save
2. Toggle breakpoints by clicking beside line numbers or pressing F9
3. Press F5, and select the executable [1]

* AUTO-COMPLETE



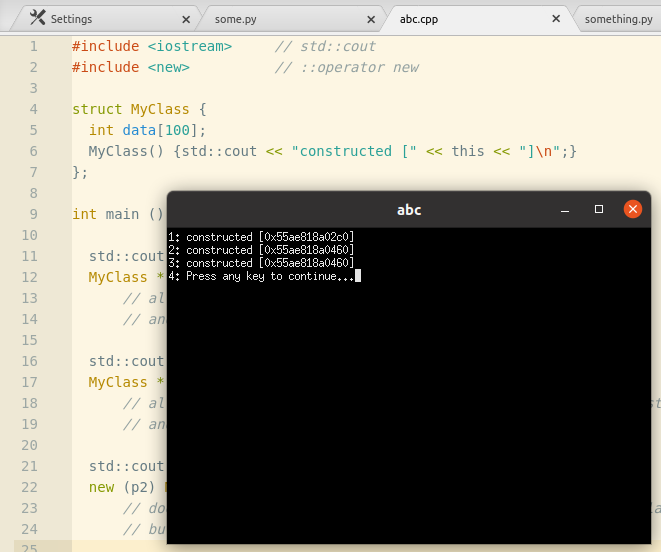
Auto-Complete is built-in inside Atom by default. The autocomplete system lets you view and insert possible completions in the editor using Tab or Enter [2]

* SYMBOL VIEWER



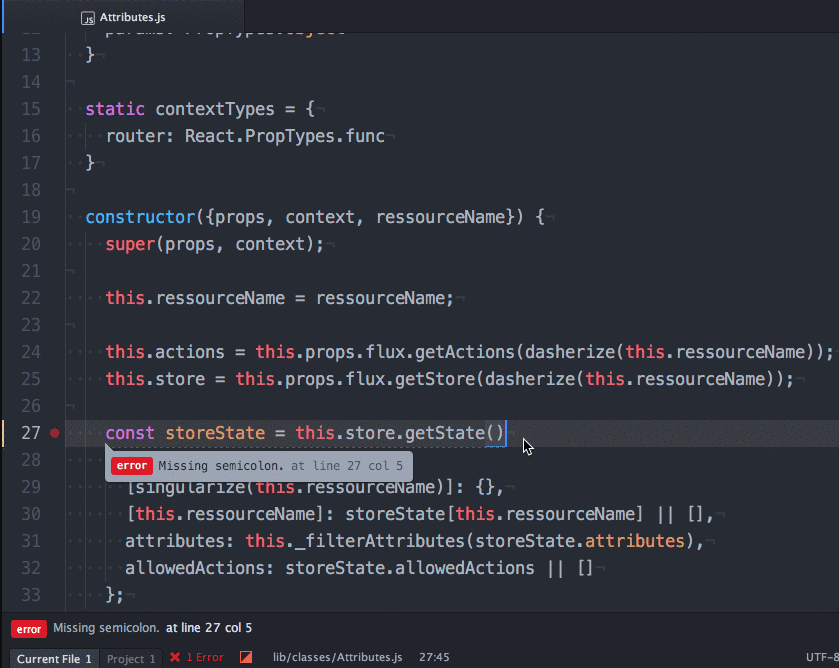
The package used for viewing symbols is symbols-navigator [3]

* BUILD



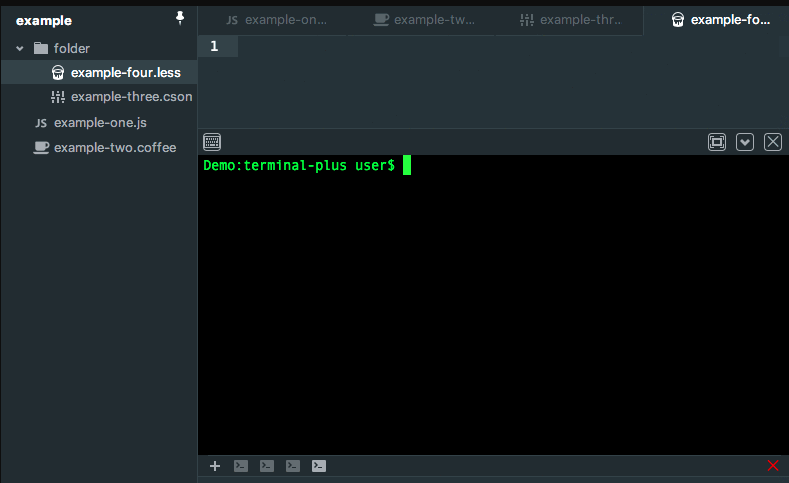
With “dgb-gdb” package being installed into Atom, building can be done done by saving the code, and then right click to select “Compile Run” or press F5. The above image shows that the cpp code is built and displays the output in XTerm terminal [1]

* CPPCHECK



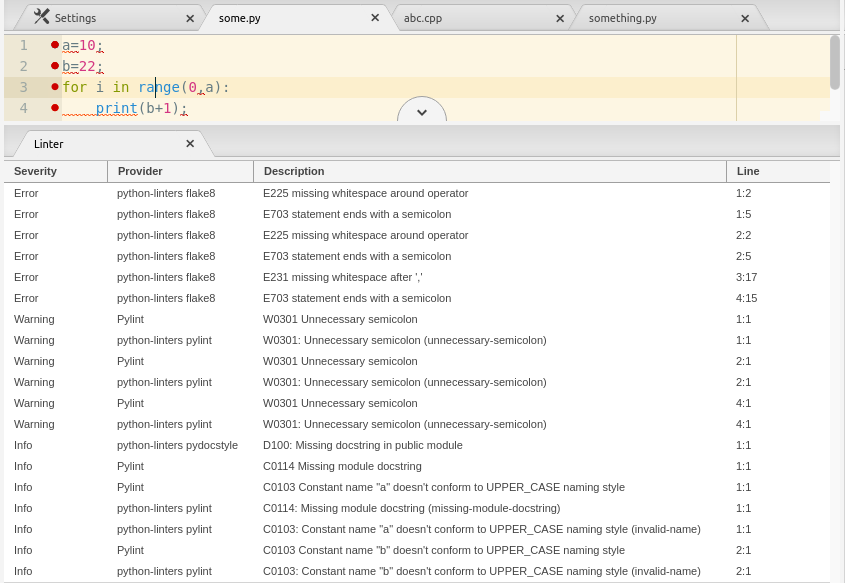
The package used for C++ cppcheck is “linter-cppcheck”. It is used for static code analysis, unused variables, code quality, coding rules, etc. In the case of Atom, linter-cppcheck continuously checks the code in real time as code is being typed [4]

* TERMINAL IN ATOM



The package used for installing the external terminal is “atom-ide-terminal”. You can use it [5]

* PYLINT



The package used for Python linter is “linter-pylint”. It is used for static code analysis, unused variables, code quality, coding rules, etc. In the case of Atom, linter-pylint continuously checks the code in real time as code is being typed [6]

REFERENCES:

1. <https://atom.io/packages/dbg-gdb>
2. <https://flight-manual.atom.io/using-atom/sections/autocomplete/>
3. [https://atom.io/packages/symbols-navigator#:~:text=Press%20ctrl%20%2B%20alt%20%2B%20o%20to,Install%20Packages%20from%20Atom%20Settings](https://atom.io/packages/symbols-navigator" \l ":~:text=Press ctrl %2B alt %2B o to,Install Packages from Atom Settings).
4. <https://atom.io/packages/linter-cppcheck>
5. <https://atom.io/packages/atom-ide-terminal>
6. https://atom.io/packages/linter-pylint