Lecture 13

Debugging & Exercises

Gyuhyeon Seo, Doyoun Kim

Overview

- Debugging
  - Debugging techniques
  - Why use debugger

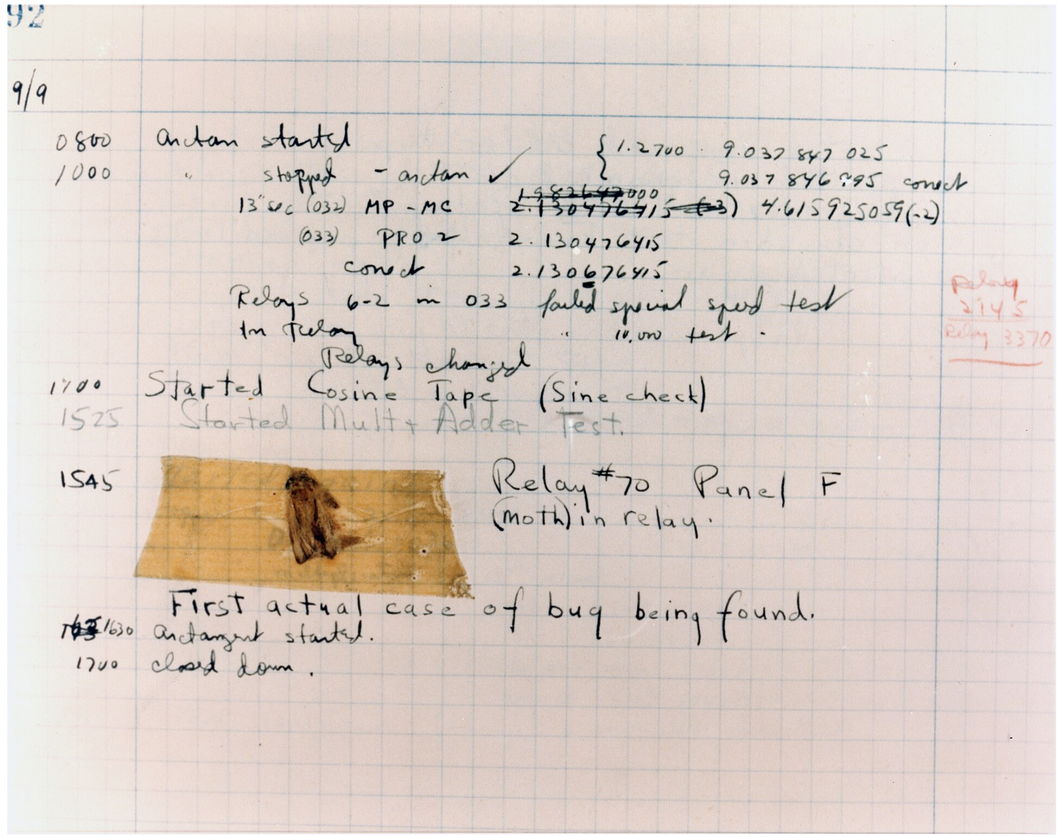
- Exercise
  - 1. list_pop.cpp
  - 2. operator_overloading.cpp
  - 3. simple_vector_bug.cpp

Debugging in Visual Studio Code

Gyuhyeon Seo

Slides from Jongwook Han, Sp24 Computing 1

Debugging



A computer log entry from the Mark2, with a moth taped to the page

- Debugging is the process of finding and resolving bugs (defects or problems that prevent correct operation) within computer programs, software, or systems.

Debugging techniques

Image from:

https://www.reddit.com/r/ProgrammerHumor/comments/ntro76/all_the_print_statements/

- Using std::cout
- Using a debugger
- Testing, logging
- Rubber duck debugging

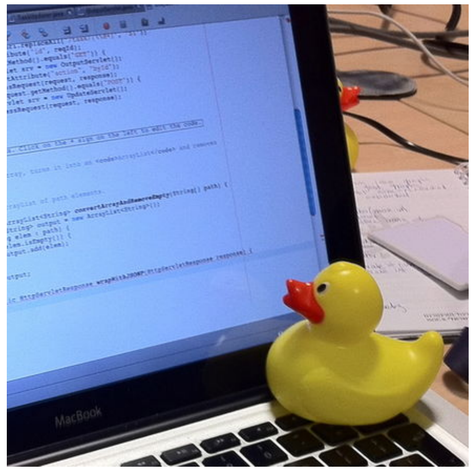


Image from:

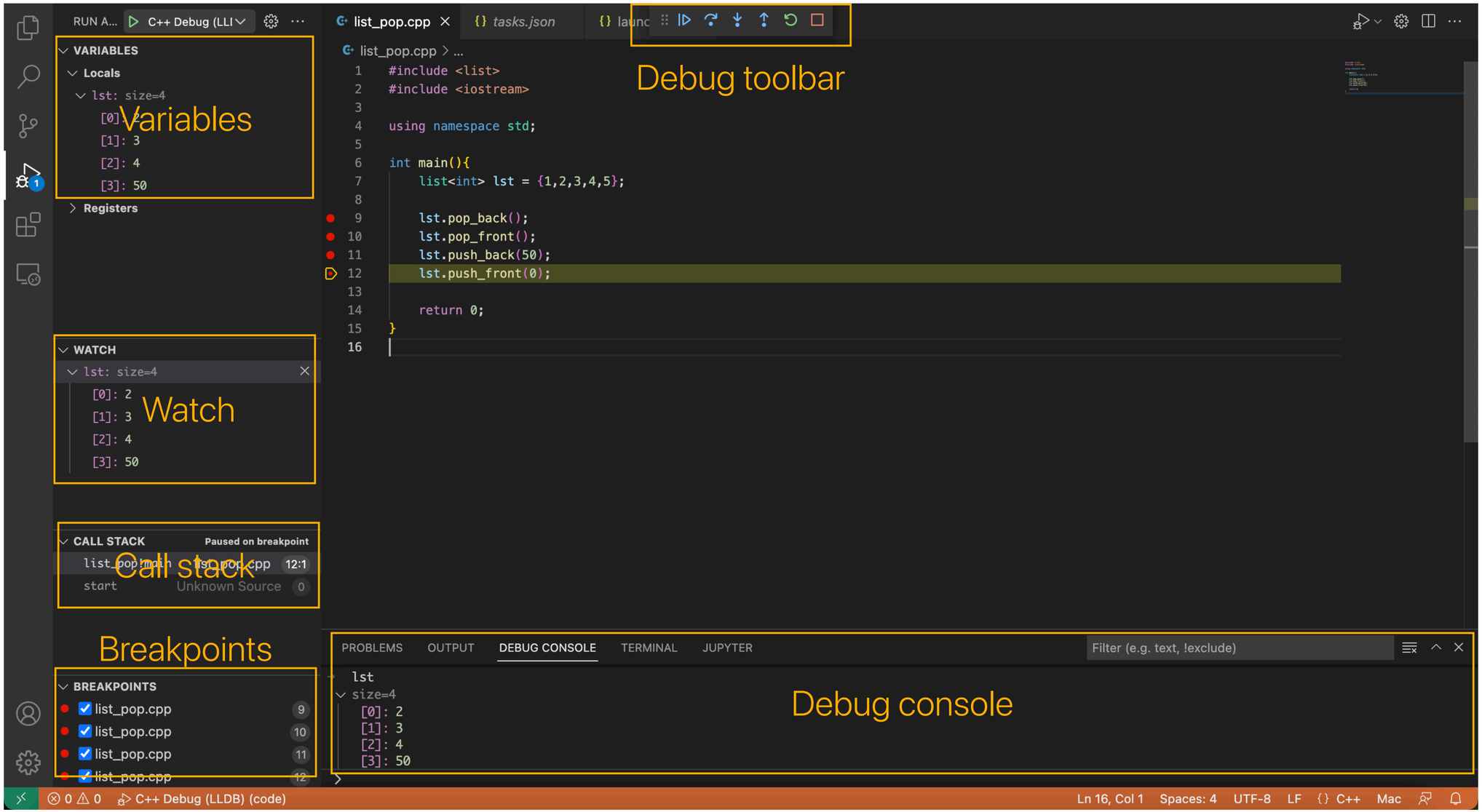
https://en.wikipedia.org/wiki/Rubber_duck_debugging

Why should I use a debugger?

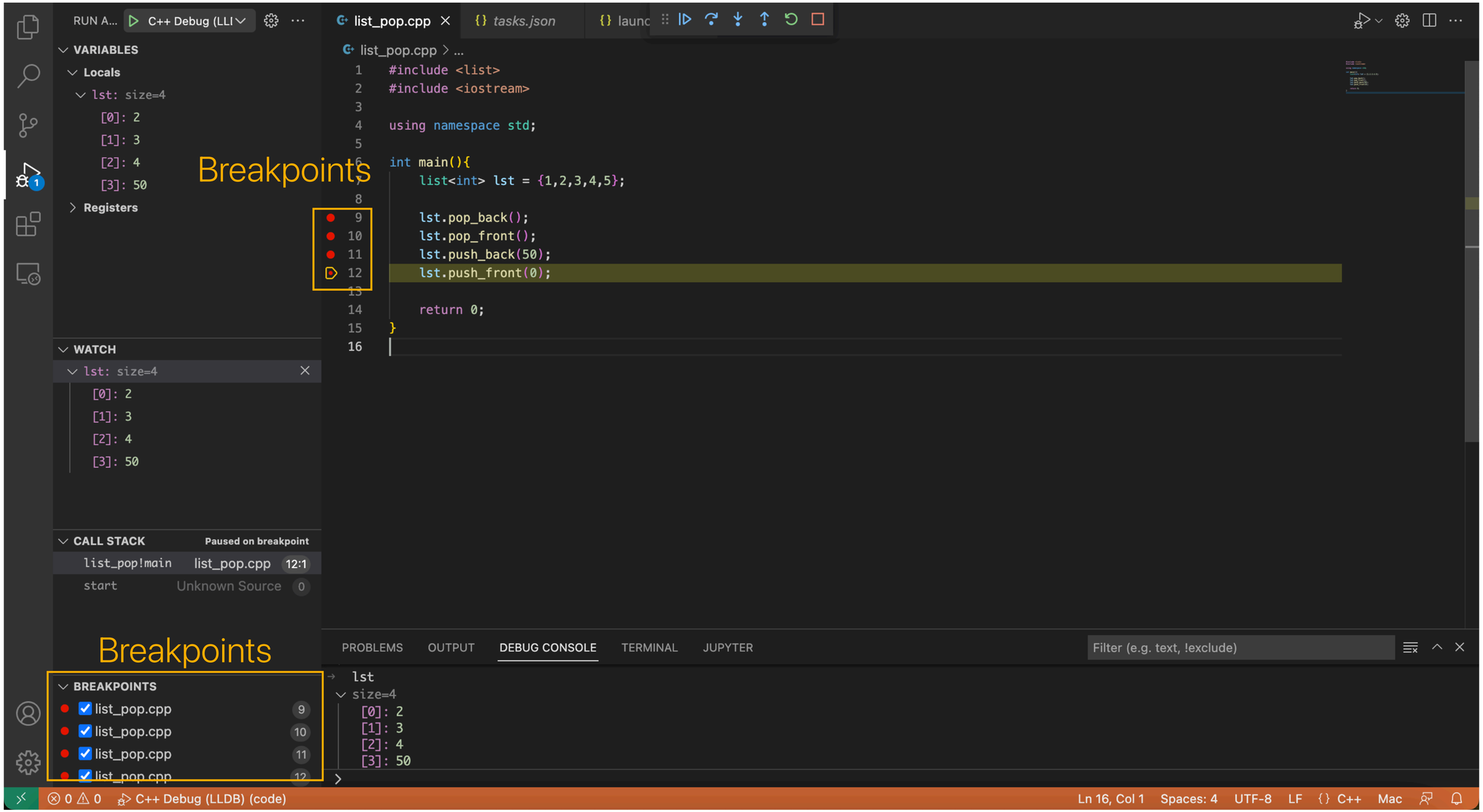
- Understanding unfamiliar code
- Memory exploration
- Efficiency and productivity

- https://news.ycombinator.com/item?id=29923007

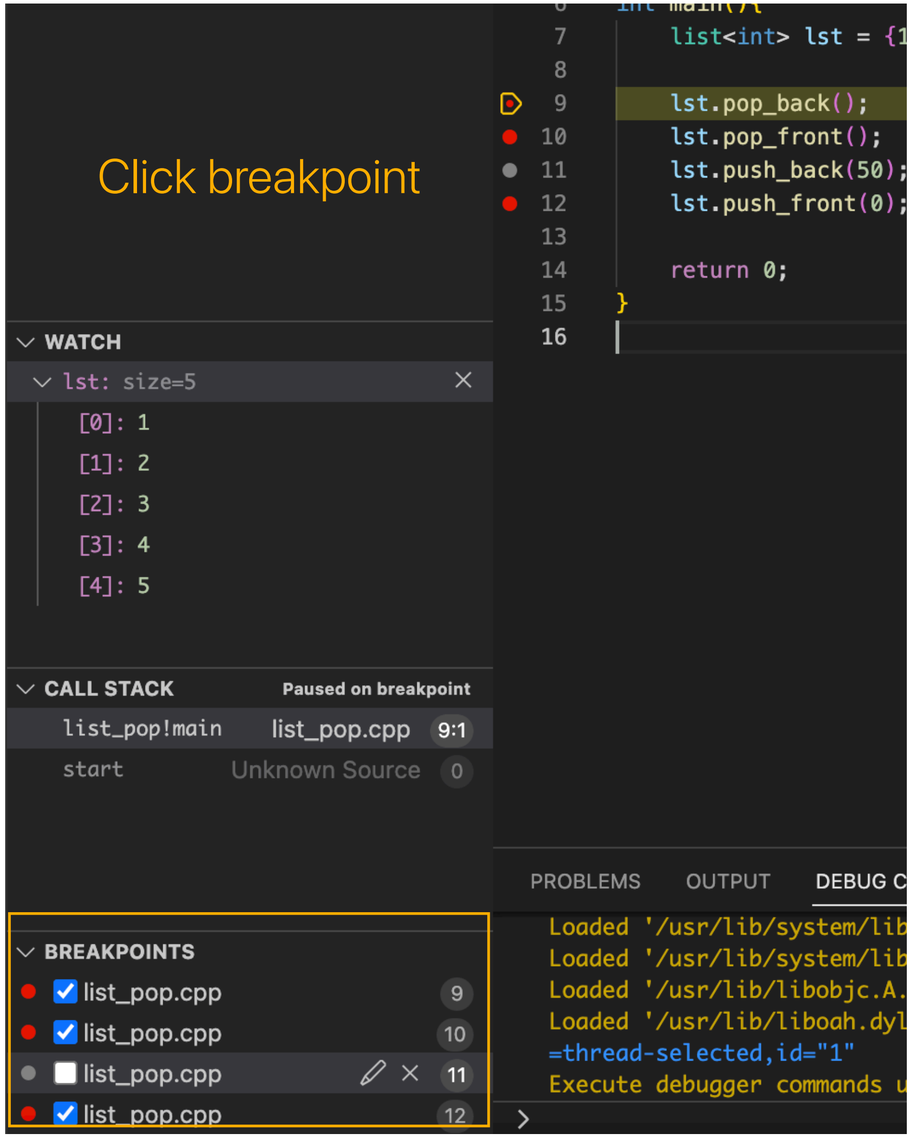
Overview of using a debugger in VSCode



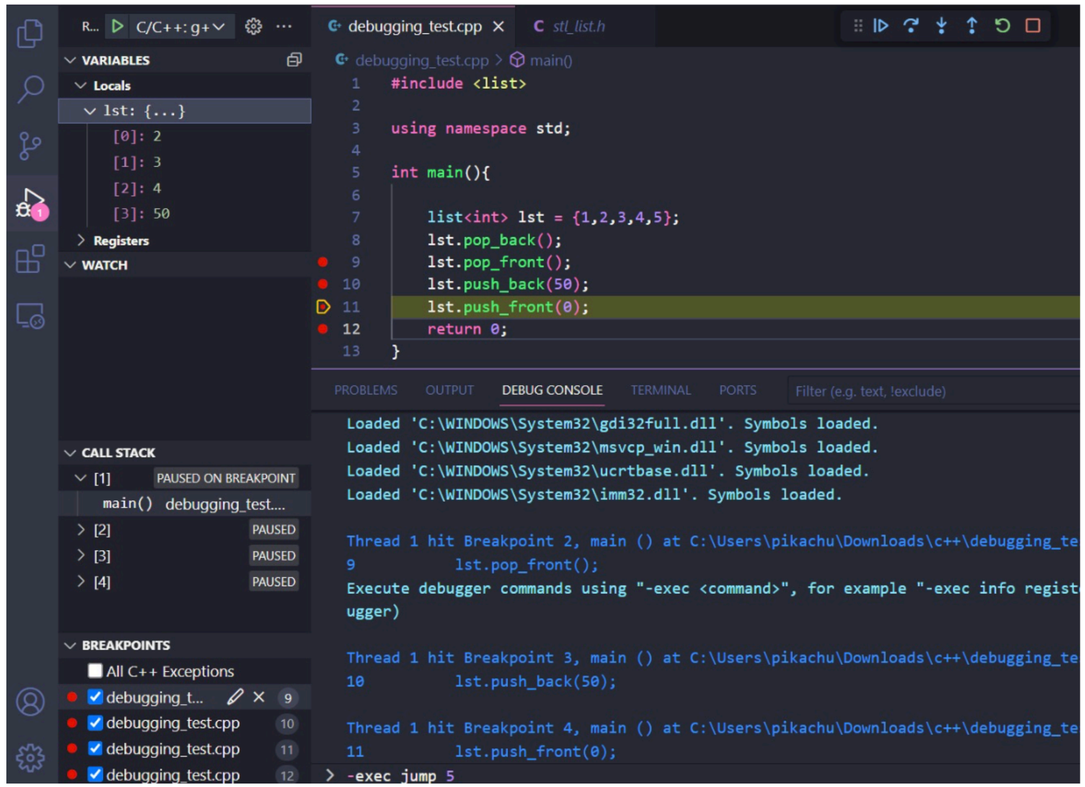
Using multiple breakpoints

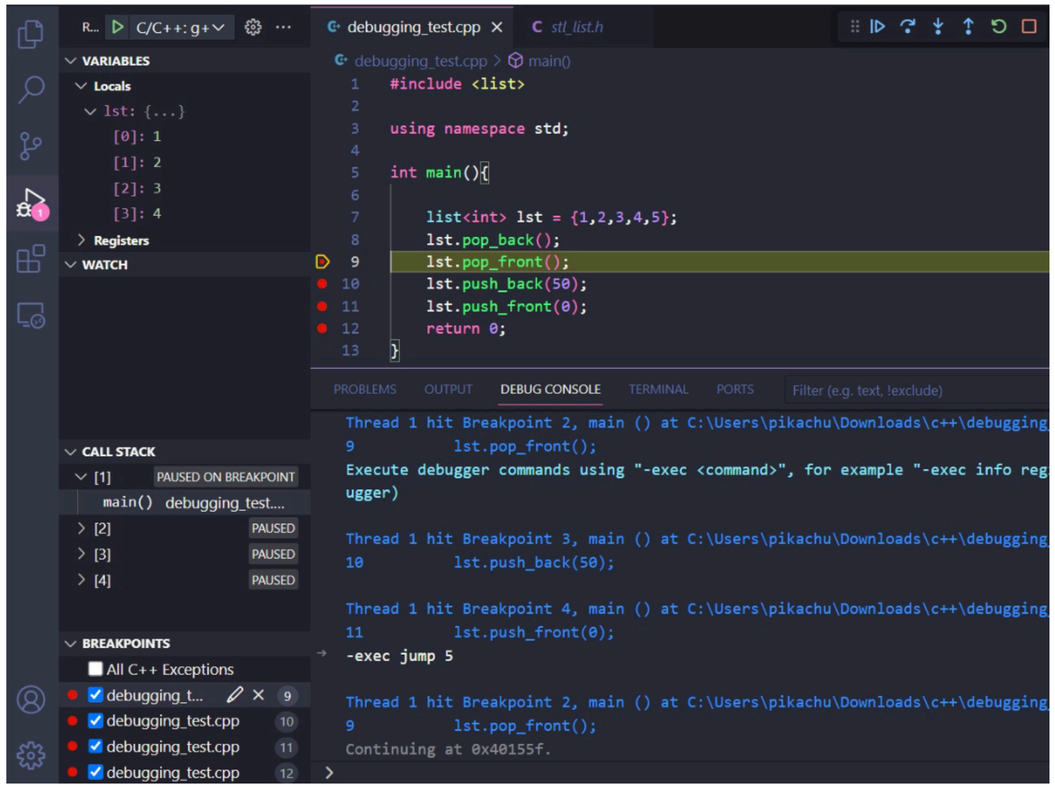


Turning off breakpoints



Jumping lines





- Type *-exec jump {line_number}* in the debug console
- If you use M1/M2(LLDB), type *thread jump --line {line_number}*

Exercise 1: list_pop.cpp

list_pop.cpp

```cpp
#include <list>
#include <iostream>

using namespace std;

int main(){
    list<int> lst = {1,2,3,4,5};

    for (int v : lst) {
        lst.pop_back();
        cout << "popped" << endl;
    }

    return 0;
}
```

Exercise 1: Using cout

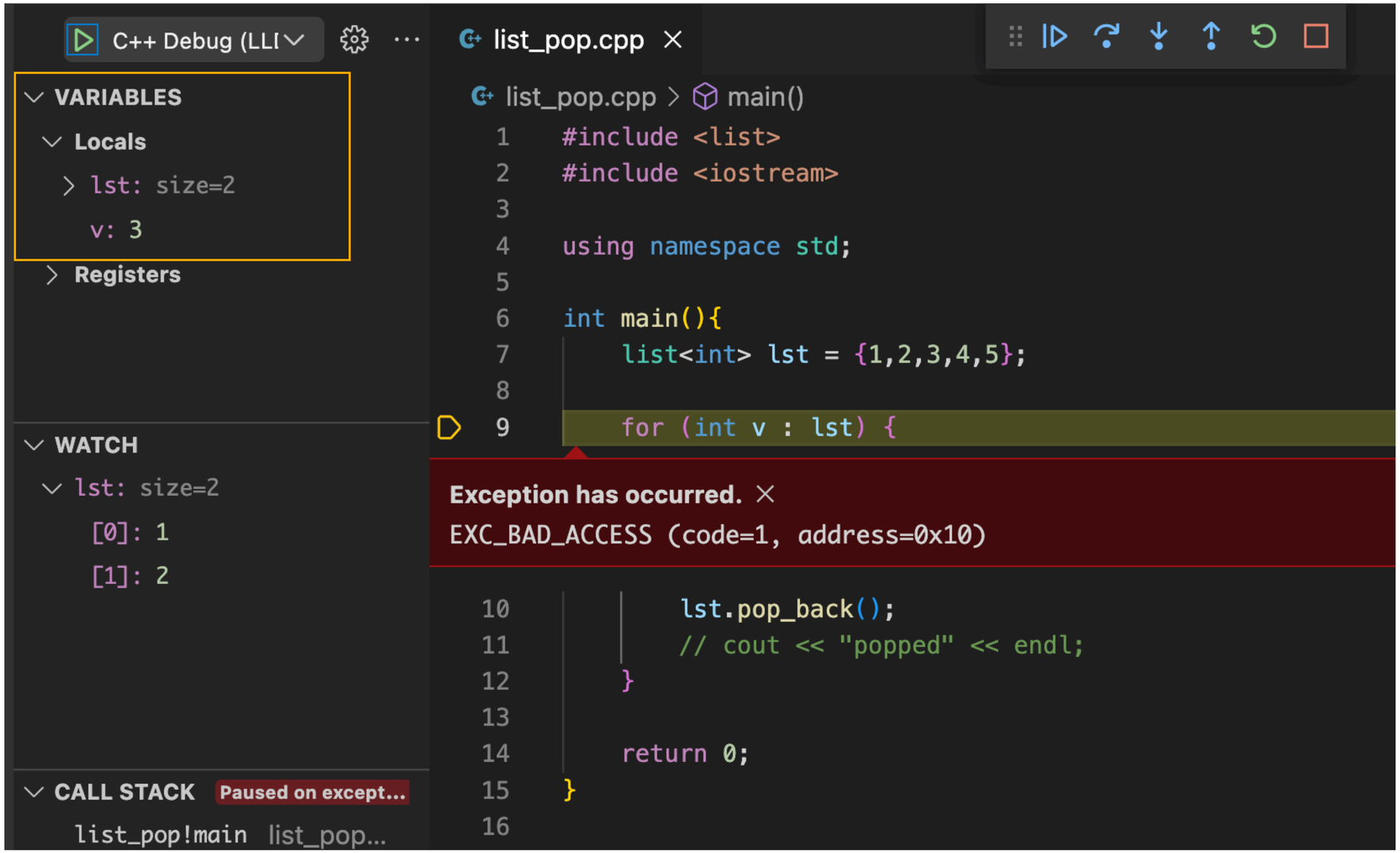
list_pop.cpp

```cpp
#include <list>
#include <iostream>

using namespace std;

int main(){
    list<int> lst = {1,2,3,4,5};

    for (int v : lst) {
        lst.pop_back();
        cout << "popped" << endl;
    }

    return 0;
}
```

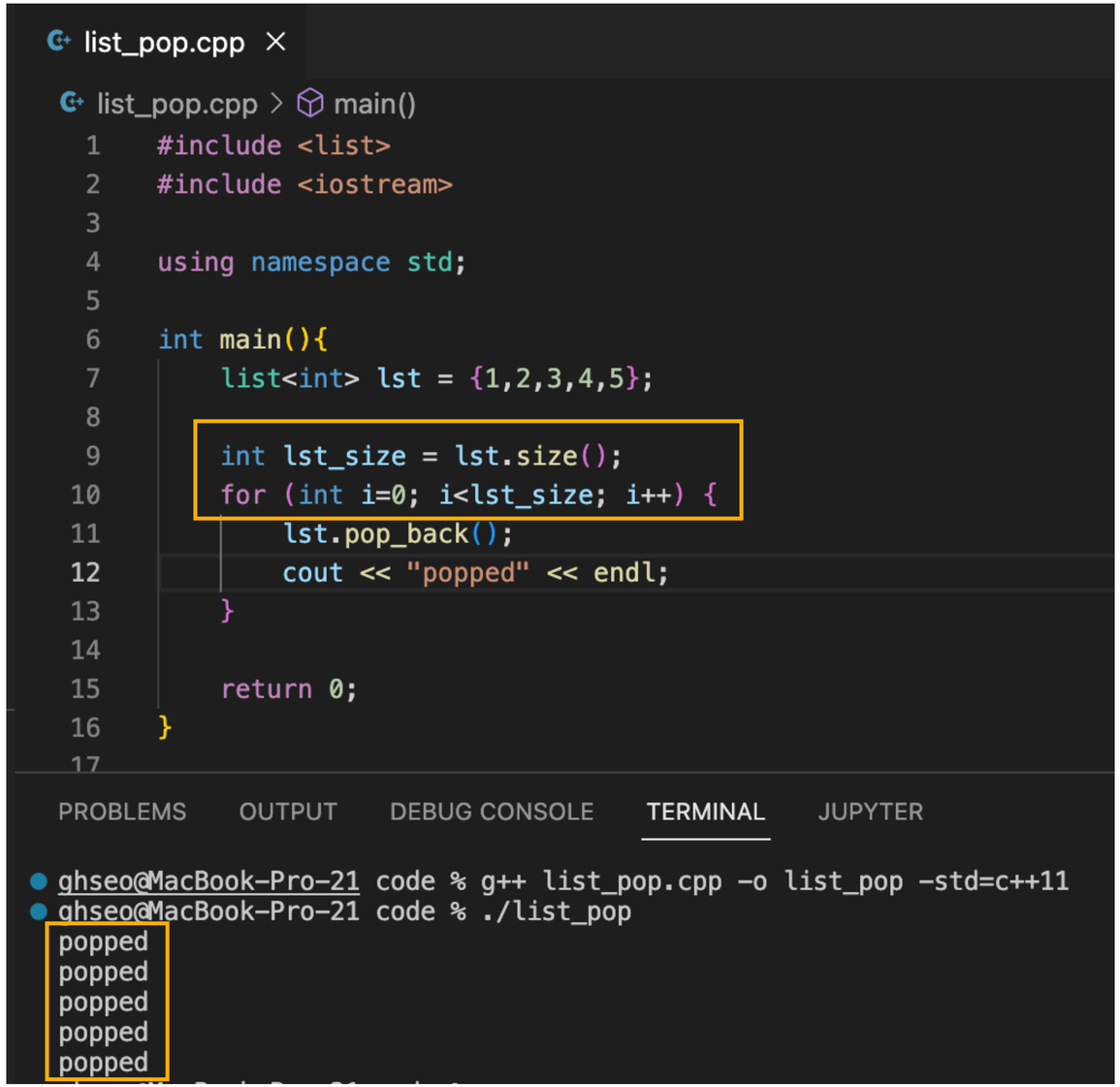
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
ghseo@MacBook-Pro-21 code % g++ list_pop.cpp -o list_pop -std=c++11
ghseo@MacBook-Pro-21 code % ./list_pop
popped
popped
popped
zsh: segmentation fault  ./list_pop
```

Exercise 1: Using debugger



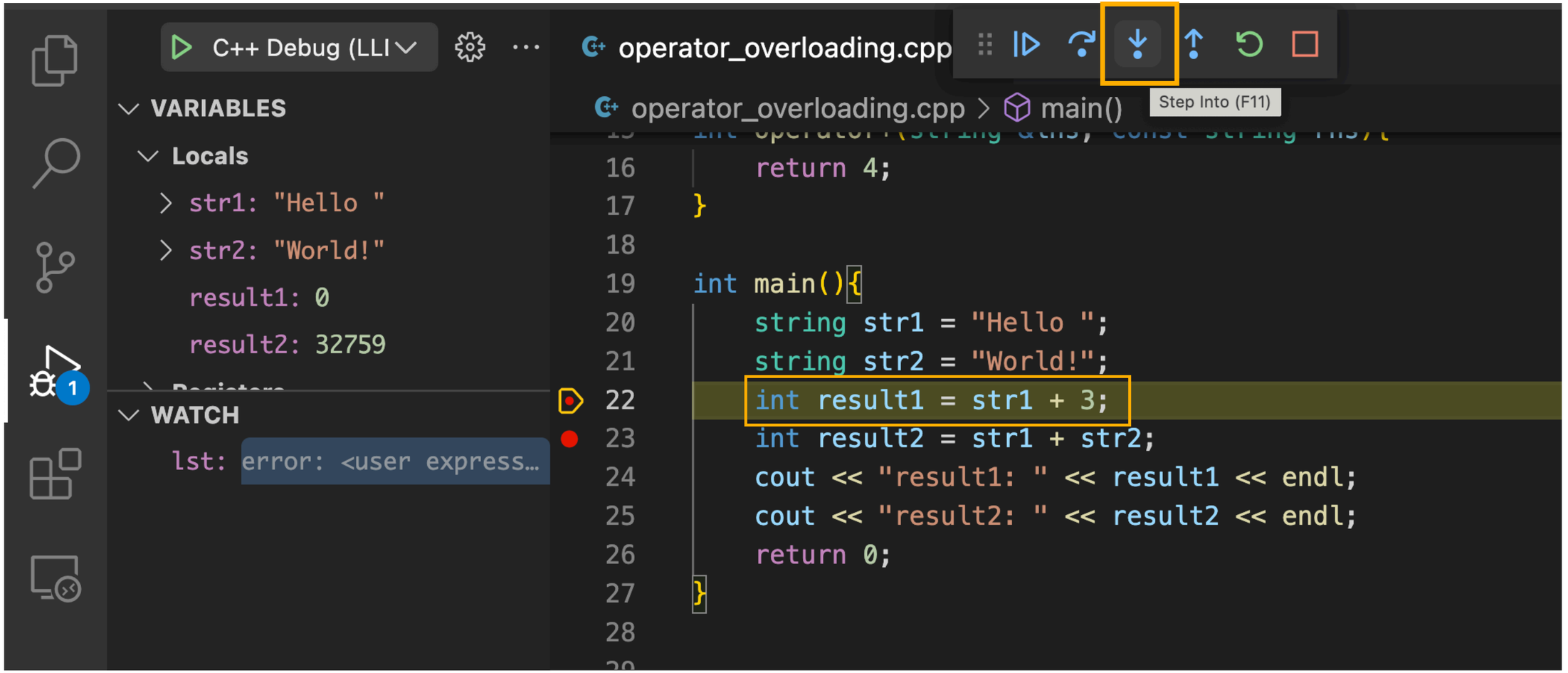
Exercise 1: Using debugger



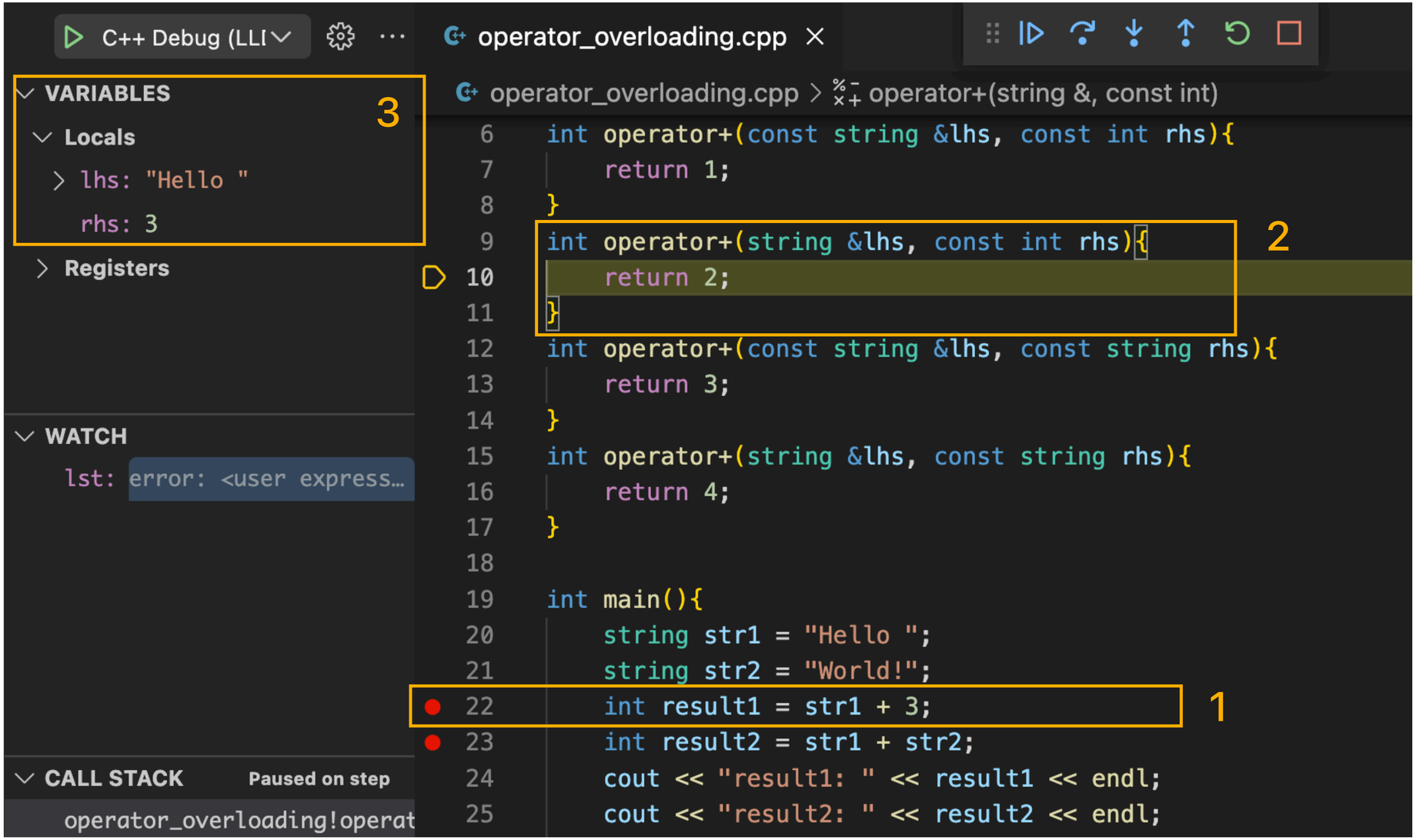
Exercise 2: operator_overloading

```
#include <string>
#include <iostream>

using namespace std;

int operator+(const string &lhs, const int rhs){
   return 1;
}
int operator+(string &lhs, const int rhs){
   return 2;
}
int operator+(const string &lhs, const string rhs){
   return 3;
}
int operator+(string &lhs, const string rhs){
   return 4;
}
int main(){
    string str1 = "Hello ";
    string str2 = "World!";
    int result1 = str1 + 3;
    int result2 = str1 + str2;
    cout << "result1: " << result1 << endl;
    cout << "result2: " << result2 << endl;
    return 0;
}
```

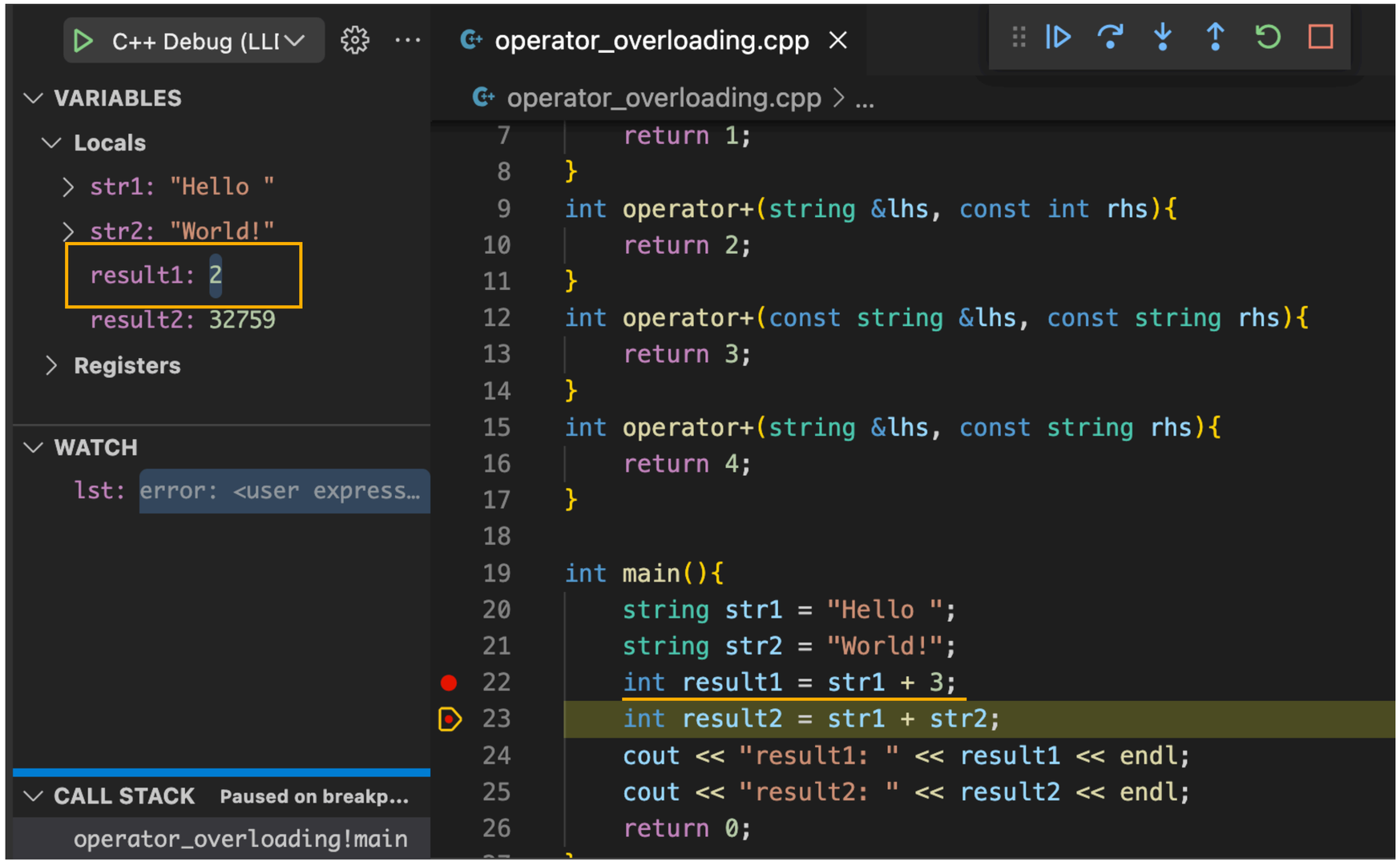
Exercise 2: operator_overloading



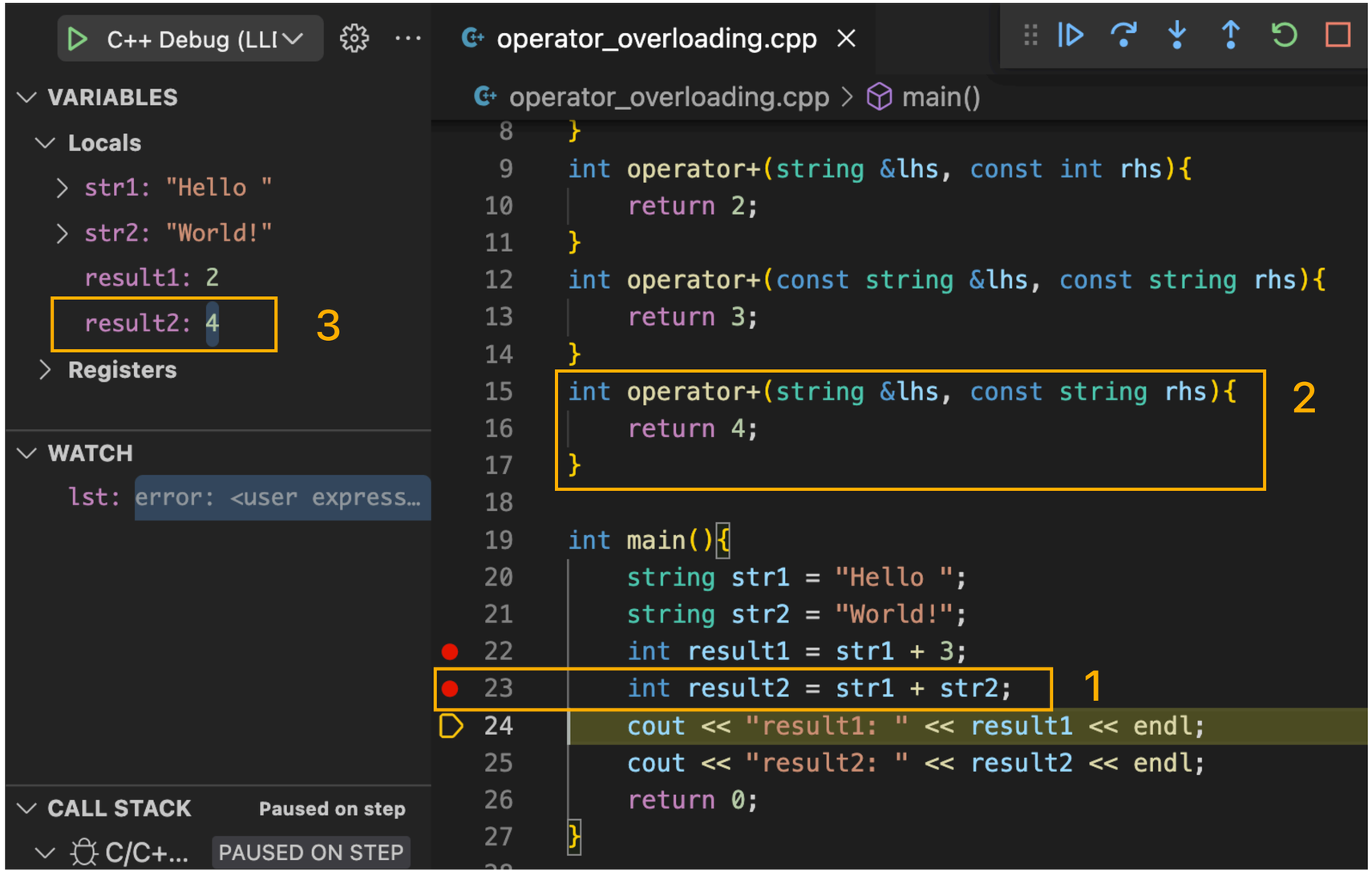
Exercise 2: operator_overloading



Exercise 2: operator_overloading



Exercise 2: operator_overloading



Exercise 3: simple_vector_bug.cpp

simple_vector_bug.cpp

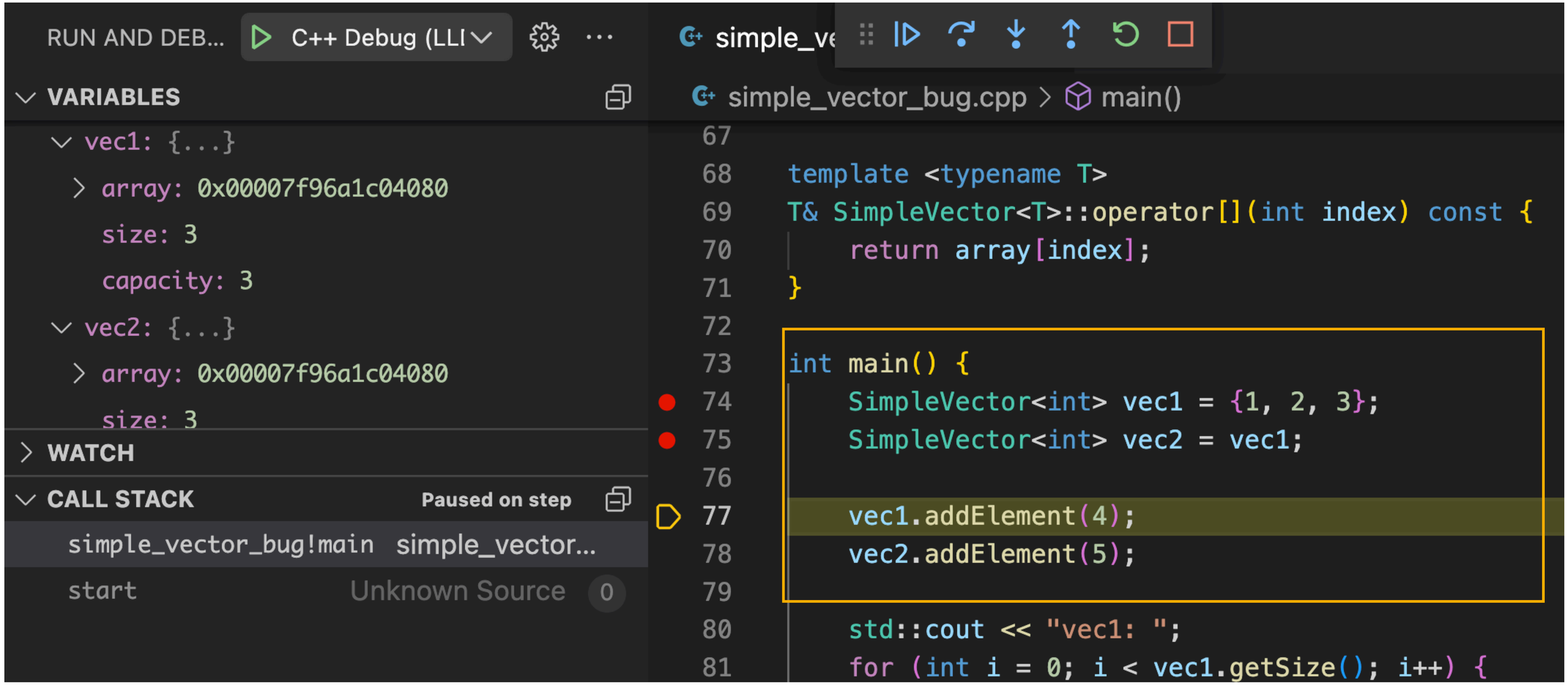
simple_vector_bug.cpp > ...

```
11
12      void resize();
13
14  public:
15      SimpleVector(int initialCapacity);
16      SimpleVector(std::initializer_list<T> elements);
17      ~SimpleVector();
18
19      void addElement(T element);
20      int getSize() const;
21
22      T& operator[](int index) const;
23  };
24
```

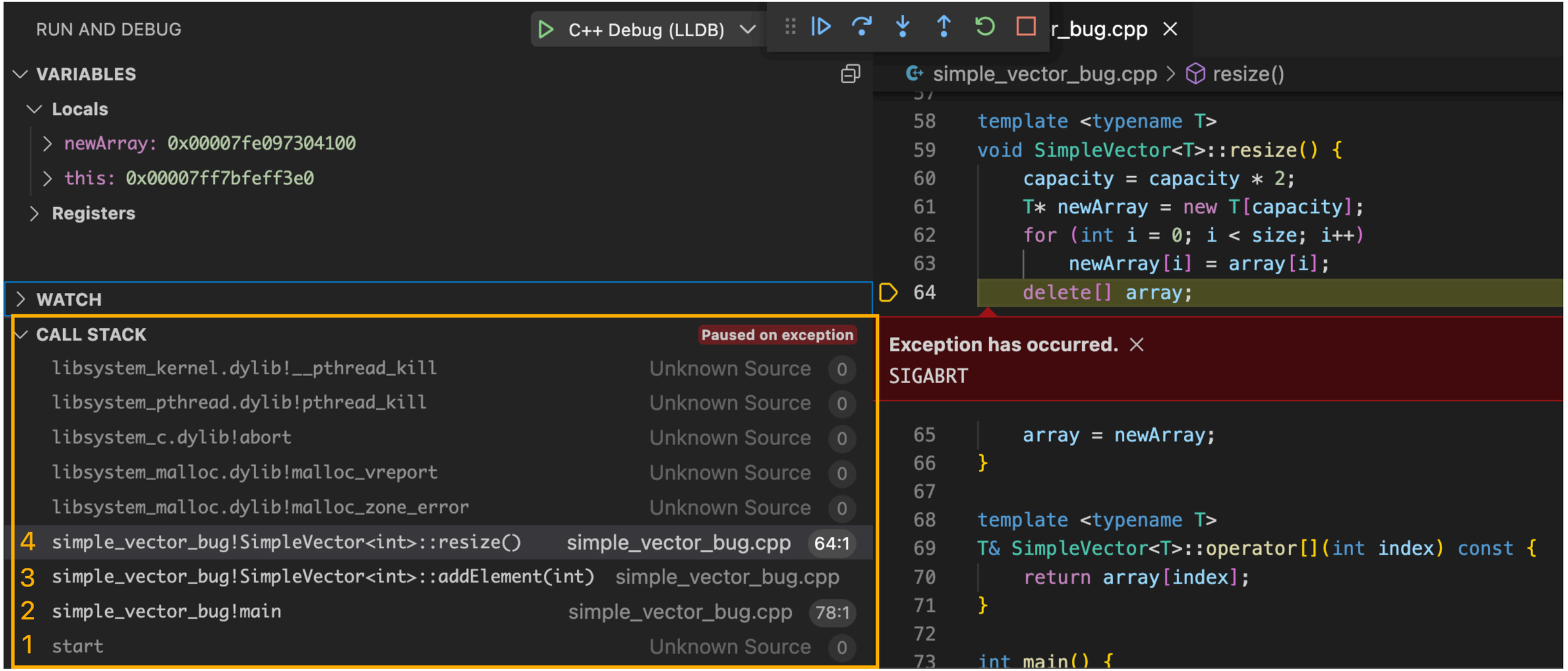
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER zsh

```
ghseo@MacBook-Pro-21 code % ./simple_vector_bug
simple_vector_bug(3942,0x7ff8455737c0) malloc: Double free of object 0x7fb15cf05e80
simple_vector_bug(3942,0x7ff8455737c0) malloc: *** set a breakpoint in malloc_error_break to debug
zsh: abort      ./simple_vector_bug
```

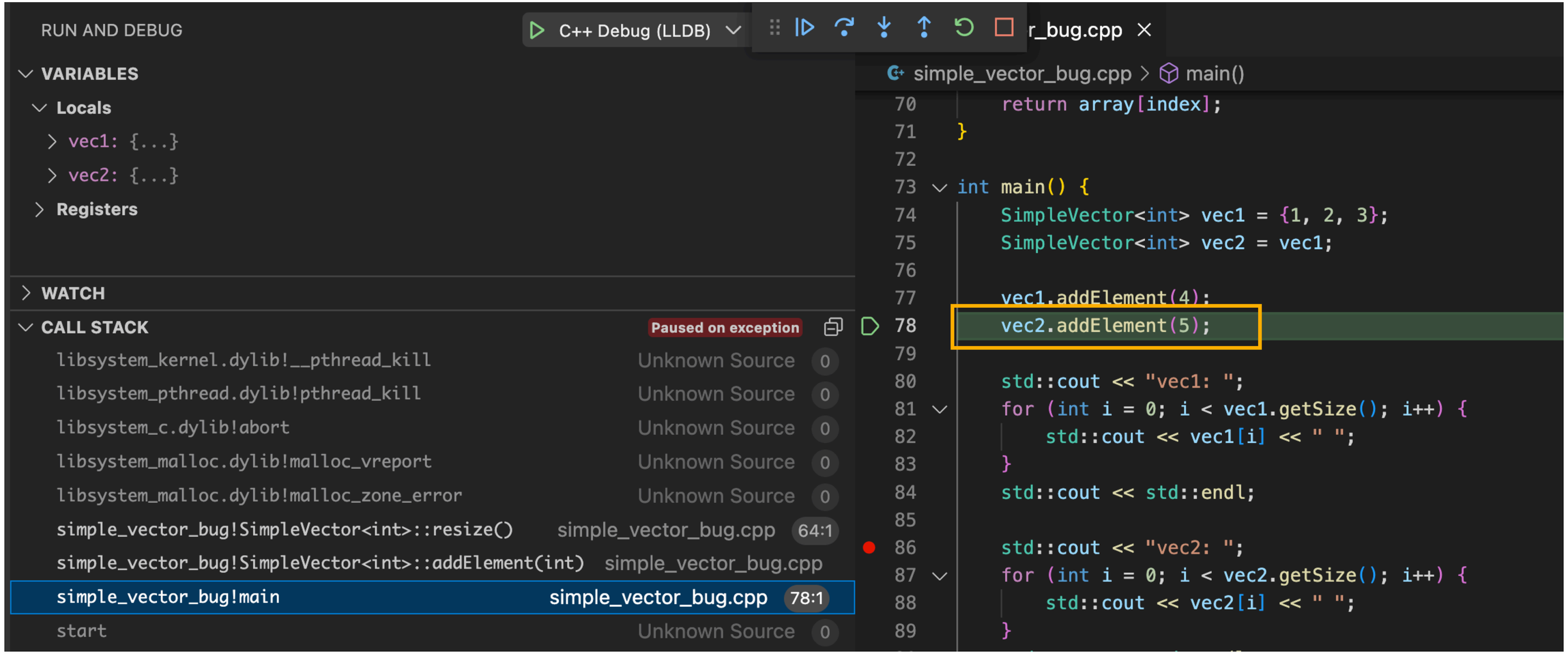
Exercise 3: simple_vector_bug.cpp



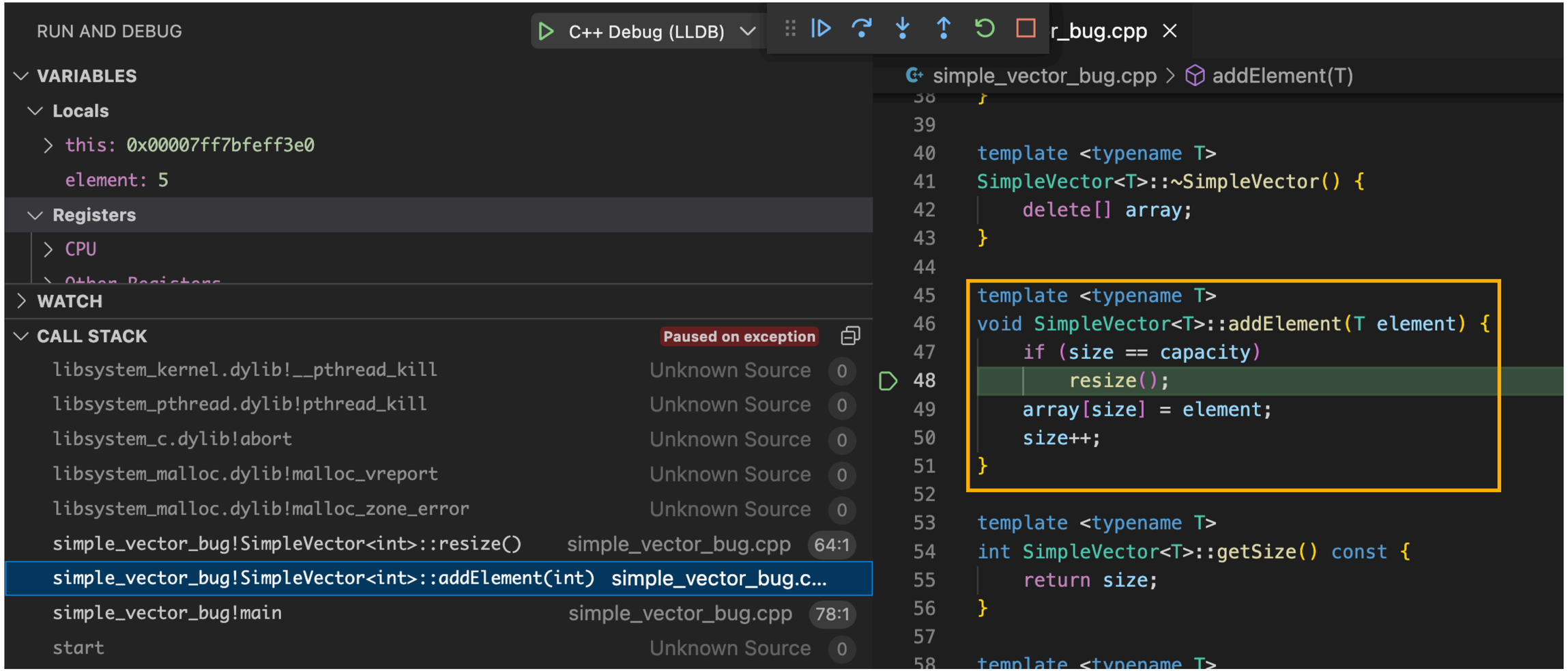
Exercise 3: simple_vector_bug.cpp



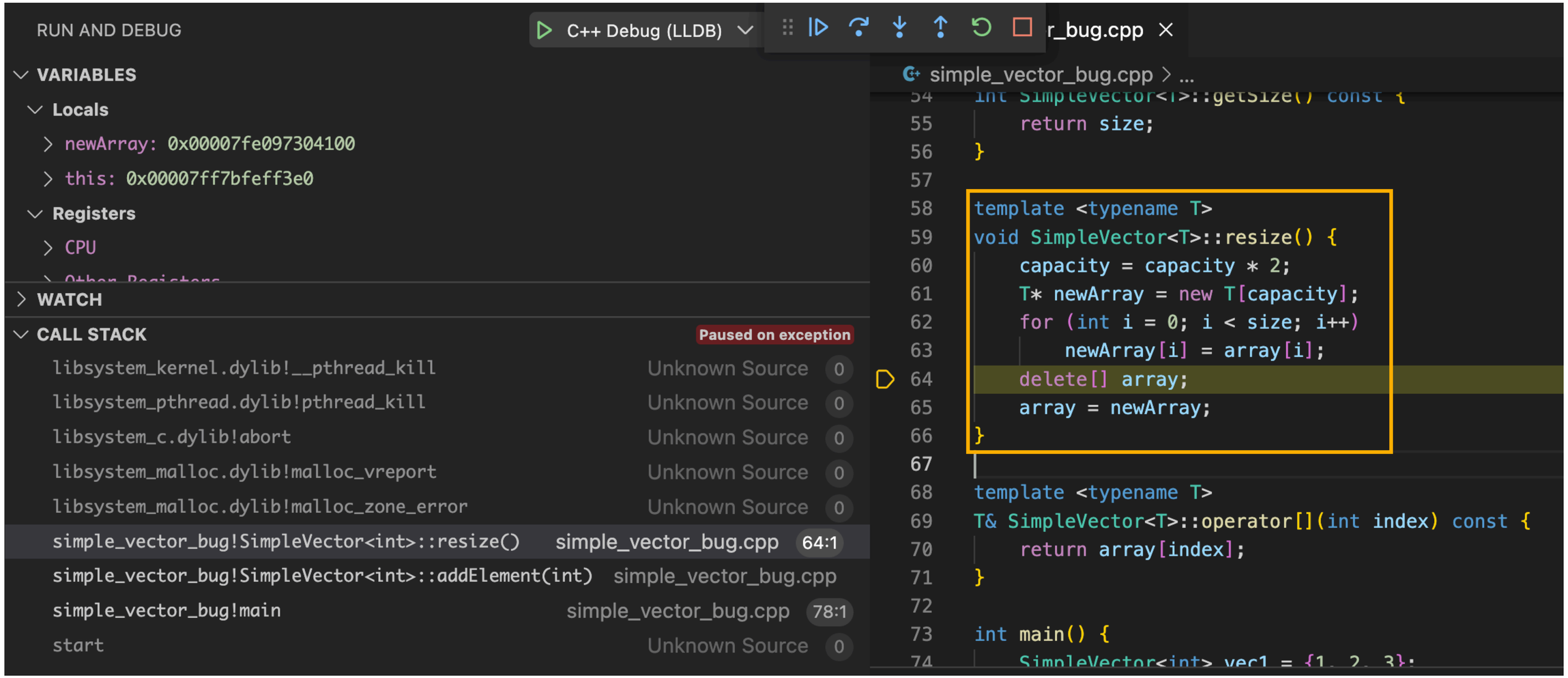
Exercise 3: simple_vector_bug.cpp



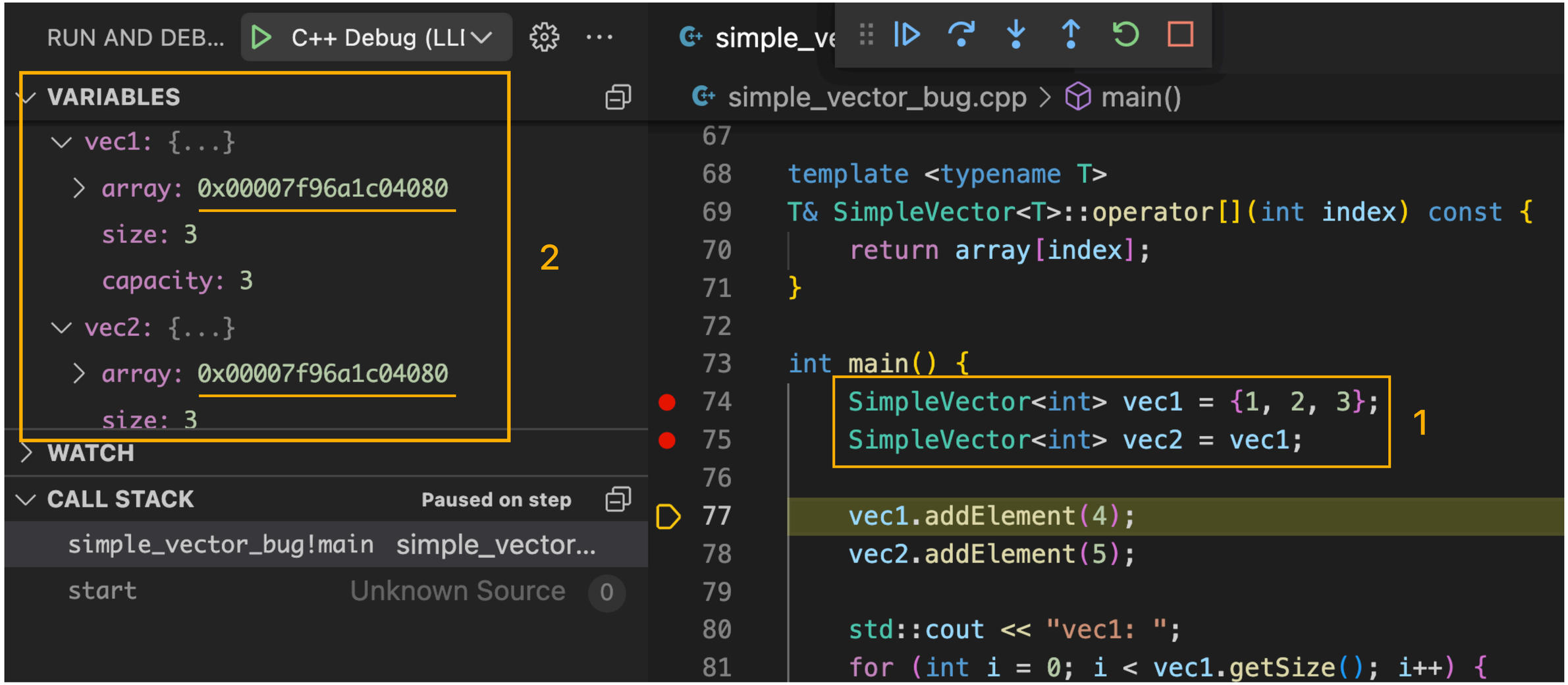
Exercise 3: simple_vector_bug.cpp



Exercise 3: simple_vector_bug.cpp



Exercise 3: simple_vector_bug.cpp



Exercise 3: simple_vector_bug.cpp

```
73  int main() {
74      SimpleVector<int> vec1 = {1, 2, 3};
75      SimpleVector<int> vec2 = vec1;
76
77      vec1.addElement(4);
78      vec2.addElement(5);
```

```
73  int main() {
74      SimpleVector<int> vec1 = {1, 2, 3};
75      SimpleVector<int> vec2 = {1, 2, 3};
76
77      vec1.addElement(4);
78      vec2.addElement(5);
79
```

Thank you

- https://code.visualstudio.com/docs/cpp/cpp-debug

Operator overloading exercise

Doyoun Kim

Overview

- A goal of this exercise is to implement class templates and apply operator overloading.
- In this exercise, you will…
  - Learn how to design a generic class that handles multiple data types.
  - Implement custom operators such as [], !=, *, and ^.

Overview

Members of the Point Class:

- T x, y: Represent the two coordinates of the point.
- Supports multiple data types (int, float, double) using class templates.

Key Methods:

- Constructor: Initializes x and y.
- Copy/Move constructors: Manages copying and moving of objects.
- Operator Overloads:
  - []: Access x and y using indices.
  - !=: Compares two points for inequality.
  - *: Calculates the dot product of two points.
  - ^: Computes the L1 or L2 norm of the point.

TODO 1&2: Implement copy/move constructor

- The copy constructor should initialize the new object's x and y with the corresponding values from another Point object.

- The move constructor should take the values from another Point object and reset the original object's values.

- Add debug output to the copy constructor, move constructor to verify whether the intended methods are being called during the tests.

```
Copy constructor called.
Move constructor called.
```

TODO 3&4: Implement copy/move assignment operator

- Copy assignment operator: The existing object should take on the same values as the other object.
- Move assignment operator: After the assignment, the original object should be reset to its default state.

- Add debug output to the copy and move assignment operator to verify whether the intended methods are being called during the tests.

```
Copy assingment operator called.
Move assingment operator called.
```

TODO 5: Implement the initializer_list assignment operator

- Implement the assignment operator for the Point class using an initializer_list.

```
Point<int> p1;
p1 = {3, 4};
```

p1 should have x = 3 and y = 4.

TODO 6: Implement the [] operator

- This operator should allow access to the x and y coordinates using an index 0 and 1.

  (0 for x, 1 for y).

```
p[0] should return the x coordinate,
p[1] should return the y coordinate.
```

TODO 7: Implement the * operator

- Implement the * operator to calculate the dot product of two Point objects.

```
Point<int> p1(1, 2), p2(3, 4);
int result = p1 * p2;
```

The result should be 11.

TODO 8: Implement the ^ operator

- Implement the ^ operator to calculate the L1 norm and L2 norm of a Point object.

```
Point<int> p1(3, 4);
double result = p1^2;
```

The result should be 5.0.

TODO 9: Implement the != operator

- Implement the != operator to check if two Point objects are not equal.
- The != operator should return true if the x or y coordinates of the two points are different. Otherwise, it should return false.

```
Point<int> p1(3, 4);
Point<int> p2(3, 5);
bool result = (p1 != p2);
```

true

Run test cases

- Find the TODO sections in exercise.h files and implement them correctly based on the instructions.
- There are 9 TODOs.
- $ g++ test_cases.cpp -o run_tests -std=c++17
- $ ./run_tests