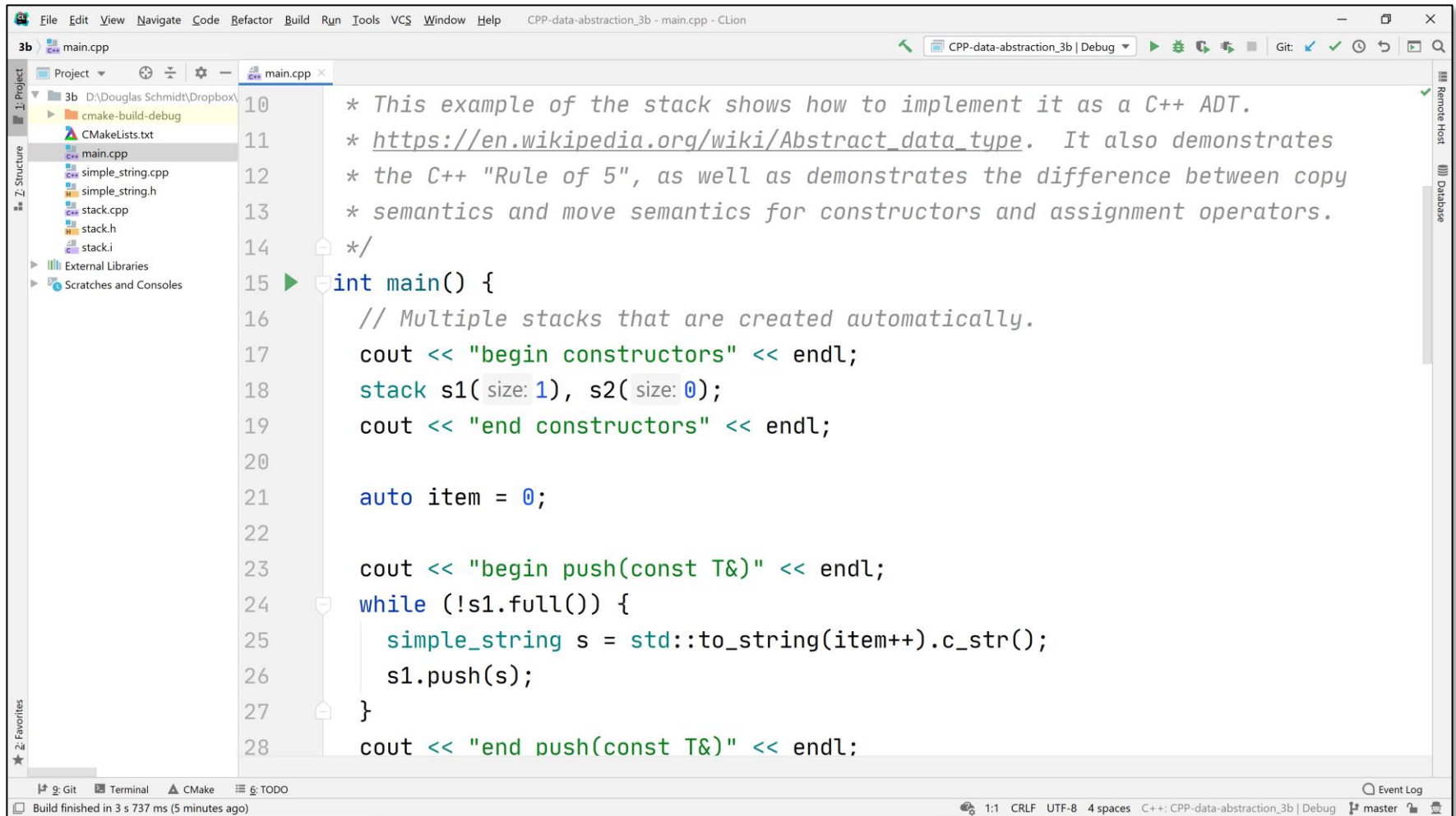


# Data Abstraction Implementation in C++ (P2)

- Use more advanced C++ class features, including `move()` optimizations



```
10  * This example of the stack shows how to implement it as a C++ ADT.
11  * https://en.wikipedia.org/wiki/Abstract\_data\_type. It also demonstrates
12  * the C++ "Rule of 5", as well as demonstrates the difference between copy
13  * semantics and move semantics for constructors and assignment operators.
14  */
15  int main() {
16      // Multiple stacks that are created automatically.
17      cout << "begin constructors" << endl;
18      stack s1( size: 1), s2( size: 0);
19      cout << "end constructors" << endl;
20
21      auto item = 0;
22
23      cout << "begin push(const T&)" << endl;
24      while (!s1.full()) {
25          simple_string s = std::to_string(item++).c_str();
26          s1.push(s);
27      }
28      cout << "end push(const T&)" << endl;
```

# Pros of Data Abstraction in C++ (P2)

- Optimizes pass-by-value semantics via `std::move()` & rvalue references
- Inlines small methods to avoid function call overhead



# Cons of Data Abstraction in C++ (P2)

- Error handling is still obtrusive
  - Use exception handling to solve this (but be careful)!
- The example is limited to a single type of stack element (`simple_string` in this case)
  - We can use C++ “parameterized types” to remove this limitation



---

# End of C++ Data Abstraction Stack Implementations

---