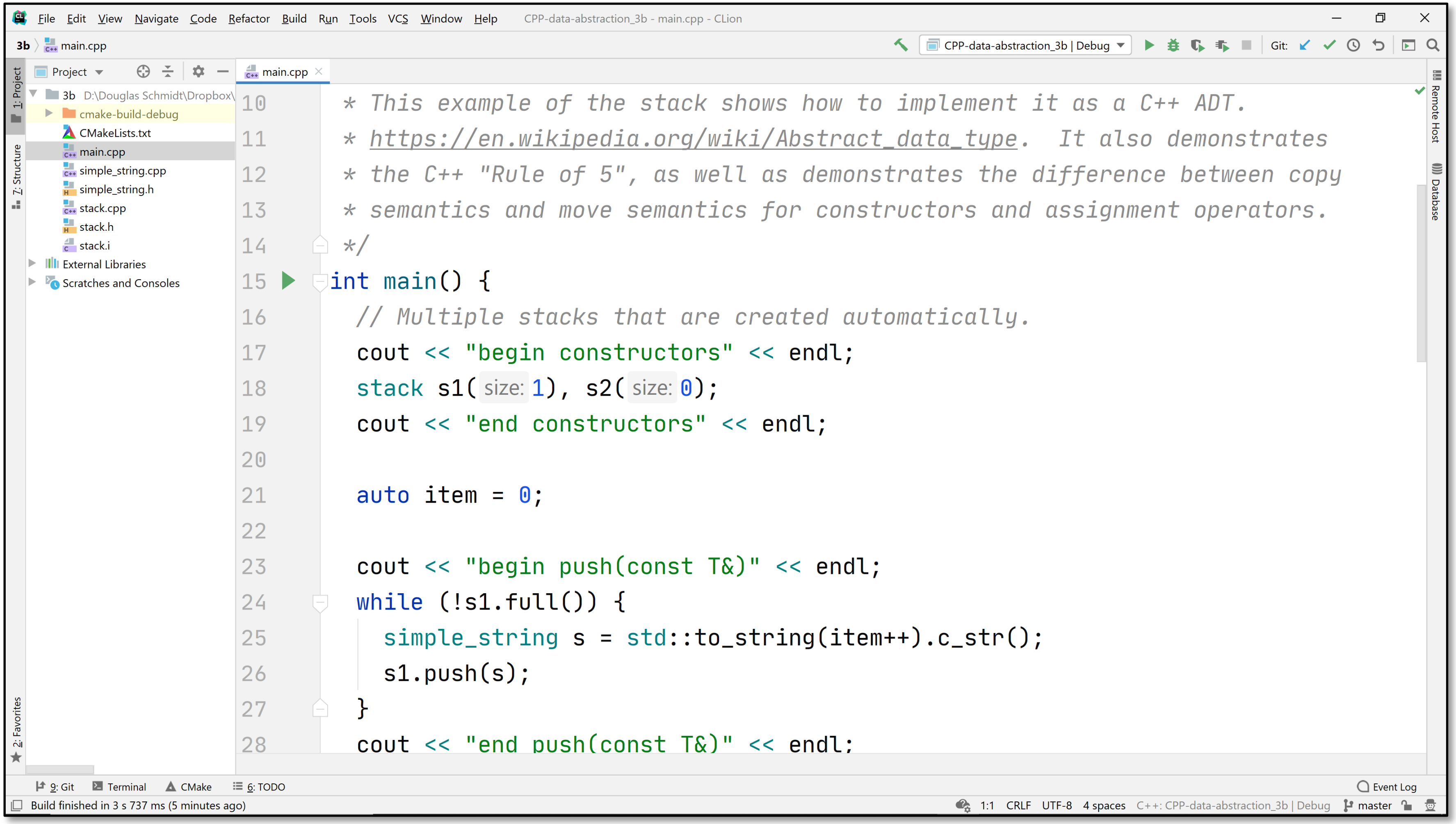
Data Abstraction Implementation in C++ (P2)

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


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
CPP-data-abstraction_3b - main.cpp - CLion
3b main.cpp
CPP-data-abstraction_3b | Debug
Git:
Project
3b D:\Douglas Schmidt\Dropbox\
cmake-build-debug
CMakeLists.txt
main.cpp
simple_string.cpp
simple_string.h
stack.cpp
stack.h
stack.i
External Libraries
Scratches and Consoles
main.cpp
1: Project
7: Structure
2: Favorites
Remote Host
Database
9: Git
Terminal
CMake
6: TODO
Event Log
Build finished in 3 s 737 ms (5 minutes ago)
1:1 CRLF UTF-8 4 spaces C++: CPP-data-abstraction_3b | Debug master

```cpp
 * This example of the stack shows how to implement it as a C++ ADT.
 * https://en.wikipedia.org/wiki/Abstract_data_type.  It also demonstrates
 * the C++ "Rule of 5", as well as demonstrates the difference between copy
 * semantics and move semantics for constructors and assignment operators.
 */
int main() {
  // Multiple stacks that are created automatically.
  cout << "begin constructors" << endl;
  stack s1(size: 1), s2(size: 0);
  cout << "end constructors" << endl;

  auto item = 0;

  cout << "begin push(const T&)" << endl;
  while (!s1.full()) {
    simple_string s = std::to_string(item++).c_str();
    s1.push(s);
  }
  cout << "end push(const T&)" << endl;
```

See CPlusPlus/tree/master/overview/capabilities/3-C++-data-abstraction/3b

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