Data Abstraction Implementation in C++ (P2)

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

```cpp
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](https://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