

Evolution of Programming Abstraction

Mechanisms: C++ Data Abstraction

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

**Institute for Software
Integrated Systems**

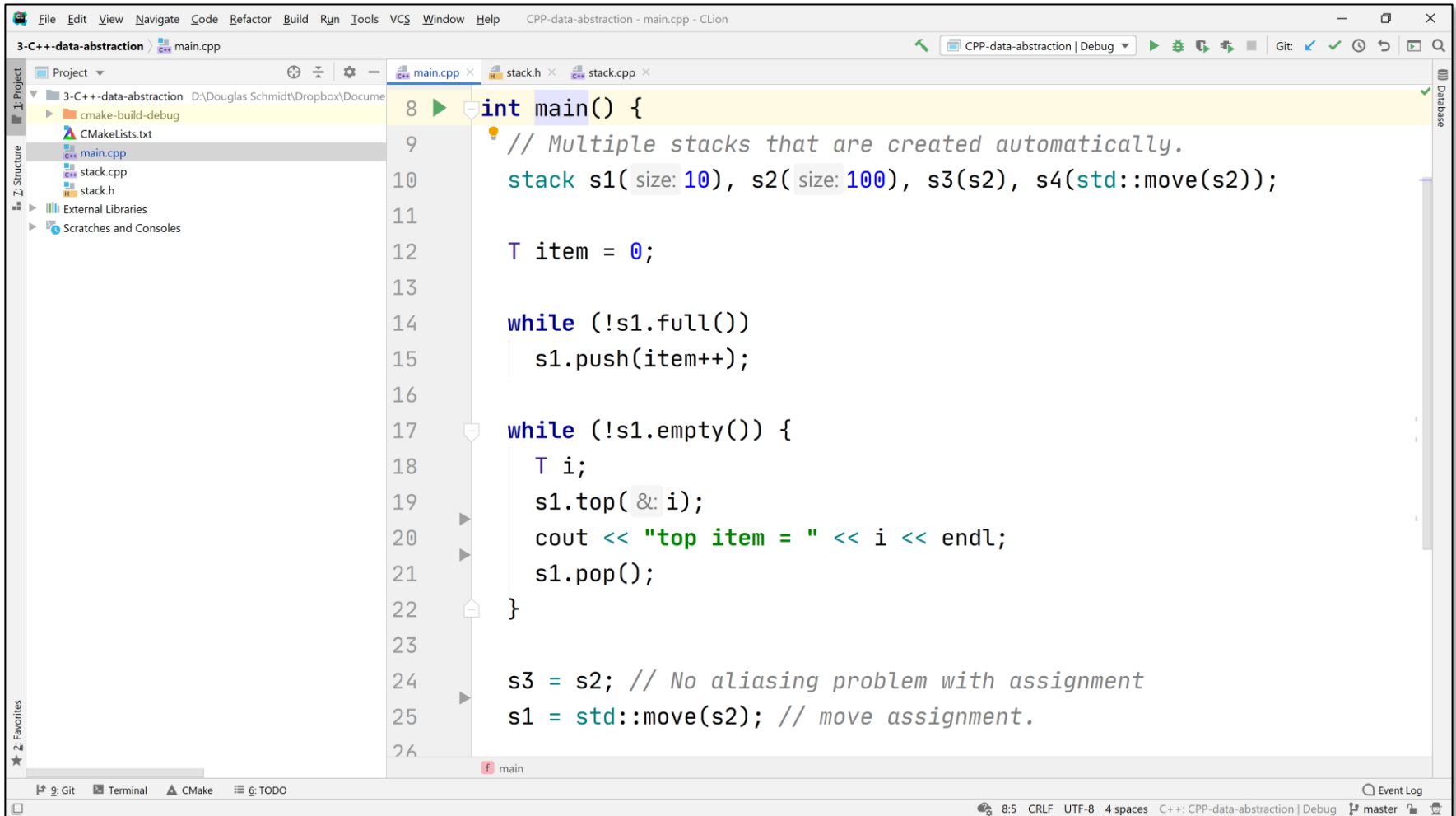
**Vanderbilt University
Nashville, Tennessee, USA**



C++ Data Abstraction Stack Implementations

Data Abstraction Implementation in C++

- Use a C++ class to achieve encapsulation & create more than one stack



```
8 int main() {
9     // Multiple stacks that are created automatically.
10    stack s1(size: 10), s2(size: 100), s3(s2), s4(std::move(s2));
11
12    T item = 0;
13
14    while (!s1.full())
15        s1.push(item++);
16
17    while (!s1.empty()) {
18        T i;
19        s1.top(&i);
20        cout << "top item = " << i << endl;
21        s1.pop();
22    }
23
24    s3 = s2; // No aliasing problem with assignment
25    s1 = std::move(s2); // move assignment.
26}
```

See [CPlusPlus/tree/master/overview/capabilities/3-C++-data-abstraction](https://github.com/DouglasC-Schmidt/CPlusPlus/tree/master/overview/capabilities/3-C++-data-abstraction)

Pros of Data Abstraction in C++

- Information Hiding & data abstraction, e.g.,

```
stack s1 (200);  
s1.top_ = 10 // Error flagged by compiler!
```

- The ability to declare multiple stack objects

```
stack s1 (10), s2 (20), s3 (30);
```

- Automatic initialization & termination

```
{  
    stack s1 (1000); // constructor called automatically.  
    // ...  
    // Destructor called automatically  
}
```



Cons of Data Abstraction in C++

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



End of C++ Data Abstraction Stack Implementations
