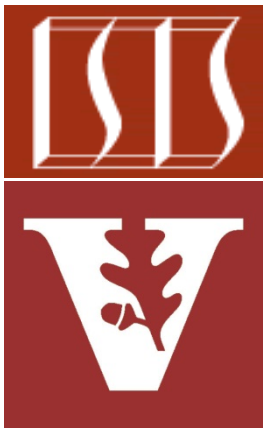
Overview of C++: Part 4

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

Exception Handling Implementation in C++

- C++ exceptions separate error handling from normal processing

```
typedef .... T; // Where "...." is a placeholder
class Stack {
public:
  class Underflow { /* ... */ };
  class Overflow { /* ... */ };
  Stack (size_t size);
  Stack (const Stack &rhs);
  void operator= (const Stack &rhs);
  ~Stack (void);
  void push (const T &item) throw (Overflow);
  void pop (T &item) throw (Underflow);
  // ...
private:
  size_t top_, size_; T *stack_;
};
```

Exception Handling Implementation in C++

- Stack.cpp

```
Stack::Stack (size_t s)
  : top_ (s), size_ (s), stack_ (new T[s]) {}

Stack::~Stack () { delete [] stack_; }

void Stack::push (const T &item) throw (Stack::Overflow) {
  if (is_full ()) throw Stack::Overflow ();
  stack_[top_++] = item;
}

void Stack::pop (T &item) throw (Stack::Underflow) {
  if (is_empty ()) throw Stack::Underflow ();
  item = stack_[--top_];
}
```

Exception Handling Implementation in C++

- Stack.cpp

```
Stack::Stack (const Stack &s):
  : top_ (s.top_), size_ (s.size_), stack_ (0)
{
  scoped_array<T> temp_stack (new T[s.size_]);

  for (size_t i = 0; i < s.size_; ++i)
    temp_stack[i] = s.stack_[i];

  temp_stack.swap (stack_);
}
```

Exception Handling Implementation in C++

- Stack.cpp

```
void Stack::operator= (const Stack &s)
{
  if (this == &s) return; // Check for self-assignment
  scoped_array<T> temp_stack (new T[s.size_]);

  for (size_t i = 0; i < s.size_; ++i)
    temp_stack[i] = s.stack_[i];

  top_ = s.top_; size_ = s.size_;
  temp_stack.swap (stack_);
}
```

Exception Handling Implementation in C++

- scoped_array extends auto_ptr to destroy built-in arrays

```
template <typename T> class scoped_array {
public:
  explicit scoped_array (T *p = 0) : ptr_ (p) {}
  ~scoped_array () { delete [] ptr_; }
  T &operator[](std::ptrdiff_t i) const {return ptr_[i];}
  T *get() const { return ptr_; }
  void swap (T *&b) { T *tmp = b; b = ptr_; ptr_ = tmp; }
  void swap (scoped_array<T> &b)
  { T *tmp=b.ptr_; b.ptr_=this->ptr_; this->ptr_=tmp; }

private:
  T *ptr_;
  scoped_array (const scoped_array<T> &);
  scoped_array &operator=(const scoped_array<T> &);
};
```

Exception Handling Implementation in C++

- There's a better way to implement operator=():

```
void Stack::operator= (const Stack &s) {
  if (this == &s) return; // Check for self-assignment
  Stack t (s);
  std::swap (t.top_, top_); std::swap (t.size_, size_);
  std::swap (t.stack_, stack_);
}
```

- The old way:

```
void Stack::operator= (const Stack &s) {
  if (this == &s) return; // Check for self-assignment
  scoped_array<T> temp_stack (new T[s.size_]);
  for (size_t i = 0; i < s.size_; ++i)
    temp_stack[i] = s.stack_[i];
  top_ = s.top_; size_ = s.size_;
  temp_stack.swap (stack_);
}
```

Exception Handling Implementation in C++

- & yet an even better way to implement an exception-safe Stack:

```
class Stack { // ...
private: // ...
  scoped_array<T> stack_;
  void swap (Stack &);
};

// ...
Stack::Stack (const Stack &s)
: top_(s.top_), size_(s.size_), stack_(new T[s.size_]) {
   for (size_t i = 0; i < s.size_; ++i)
     stack_[i] = s.stack_[i];
}

Stack::~Stack () { /* no-op! */ }
```

Exception Handling Implementation in C++

- & yet an even better way to implement operator=():

```
void Stack::operator= (const Stack &s) {
  if (this == &s) return; // Check for self-assignment
  Stack temp_stack (s);
  swap (temp_stack);
}

void Stack::swap (Stack &t) {
  std::swap (t.top_, top_);
  std::swap (t.size_, size_);
  t.stack_.swap (stack_);
}
```

- This solution is easy to generalize!

Exception Handling Implementation in C++

- Use case

```
#include "Stack.h"
void foo (void) {
  Stack s1 (1), s2 (100);
  try {
    T item;
    s1.push (473);
    s1.push (42); // Exception, push'd full stack!
    s2.pop (item); // Exception, pop'd empty stack!
    s2.top_ = 10; // Access violation caught!
  } catch (Stack::Underflow)
  { /* Handle underflow... */ }
  catch (Stack::Overflow) { /* Handle overflow... */ }
  catch (...) { /* Catch anything else... */ throw; }
  } // Termination is handled automatically.
}
```