Overview of C++: Part 5

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

Template Implementation in C++

- A parameterized type Stack class interface using C++

```
template <typename T> class Stack {
public:
  Stack (size_t size);
  Stack (const Stack<T> &rhs);
  void operator= (const Stack<T> &rhs);
  ~Stack (void)
  void push (const T &item);
  void pop (T &item); // Beware this implementation!
  bool is_empty (void) const;
  bool is_full (void) const;

private:
  size_t top_, size_; scoped_array<T> stack_;
  void swap (scoped_array<T> &t);
};
```

Template Implementation in C++

- A parameterized type Stack class implementation using C++

```
template <typename T> inline
Stack<T>::Stack (size_t size)
: top_ (0), size_ (size), stack_ (new T[size]) { }

template <typename T> inline
Stack<T>::~Stack (void) { /* no-op! */ }

template <typename T> inline void
Stack<T>::push (const T &item) {
  stack_[top_++] = item;
}

template <typename T> inline void // Beware this implementation!
Stack<T>::pop (T &item) {
  item = stack_[--top_];
}
```

Template Implementation in C++

- Note minor changes to accommodate parameterized types

```
#include "Stack.h"
void foo (void) {
  Stack<int> s1 (1000);
  Stack<float> s2;
  Stack< Stack <Activation_Record> *> s3;

  s1.push (-291);
  s2.top_ = 3.1416; // Access violation caught!
  s3.push (new Stack<Activation_Record>);
  Stack <Activation_Record> *sar;
  s3.pop (sar); // Beware this form of pop()!!!

  delete sar;
  // Termination of s1, s2, & s3 handled automatically
}
```

Template Implementation in C++

- Another parameterized type Stack class

```
template <typename T, size_t SIZE> class Stack {
public:
  Stack (void);
  ~Stack (void)
  void push (const T &item);
  void pop (T &item);
private:
  size_t top_, size_;
  T stack_[SIZE];
};
```

- No need for dynamic memory, though SIZE must be const, e.g.,

```
Stack<int, 200> s1;
```

Overview of C++: Part 6

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

Object-Oriented Implementation in C++

- Problems with previous examples:
  - Changes to the implementation will require recompilation & relinking of clients
  - Extensions will require access to the source code
- Solutions
  - Combine inheritance with dynamic binding to completely decouple interface from implementation & binding time
  - This requires the use of C++ abstract base classes

Object-Oriented Implementation in C++

- Defining an abstract base class in C++

```
template <typename T>
class Stack {
public:
  virtual ~Stack (void) = 0; // Need implementation!
  virtual void push (const T &item) = 0;
  virtual void pop (T &item) = 0;
  virtual bool is_empty (void) const = 0;
  virtual bool is_full (void) const = 0;
  void top (T &item); // Apply Template Method pattern
};
```

- By using “pure virtual methods,” we can guarantee that the C++ compiler won’t allow instantiation!

Object-Oriented Implementation in C++

- It's entirely possible (& often desirable) to implement a non-virtual method in an abstract base class, e.g.:

```
template <typename T> void Stack::top (T &item) {
  // Apply the Template Method pattern
  pop (item);
  push (item);
}
```

- Subclasses of Stack *must* implement pop() & push()

Object-Oriented Implementation in C++

• Deriving from the Stack abstract base class to define a V_Stack subclass

```
#include "Stack.h"
#include <vector>
template <typename T> class V_Stack : public Stack<T> {
public:
  enum { DEFAULT_SIZE = 100 };
  V_Stack (size_t size = DEFAULT_SIZE);
  V_Stack (const V_Stack &rhs);
  virtual void push (const T &item);
  virtual void pop (T &item);
  virtual bool is_empty (void) const;
  virtual bool is_full (void) const;

private:
  size_t top_;
  std::vector<T> stack_;
};
```

Object-Oriented Implementation in C++

- class V_Stack implementation

```
template <typename T>
V_Stack<T>::V_Stack (size_t size)
  : top_ (0), stack_ (size) {}

template <typename T>
V_Stack<T>::V_Stack (const V_Stack &rhs)
  : top_ (rhs.top_), stack_ (rhs.stack_) {}
```

Object-Oriented Implementation in C++

- class V_Stack implementation

```
template <typename T> void
V_Stack<T>::push(const T &item){ stack_[top_++] = item; }

template <typename T> void
V_Stack<T>::pop (T &item) { item = stack_[--top_]; }

template <typename T> int
V_Stack<T>::is_full (void) const {
  return top_ >= stack_.size ();
}

template <typename T> int
V_Stack<T>::is_empty (void) const {
  return top_ == 0;
}
```

Object-Oriented Implementation in C++

- Inheritance can also create an linked list stack (L_Stack):

```
template <typename T> class Node; // forward declaration.

template <typename T> class L_Stack : public Stack<T> {
public:
  enum { DEFAULT_SIZE = 100 };
  L_Stack (size_t size_hint = DEFAULT_SIZE);
  virtual ~L_Stack (void);
  virtual void push (const T &new_item);
  virtual void pop (T &top_item);
  virtual bool is_empty (void) const;
  virtual bool is_full (void) const;

private:
  Node<T> *head_; // Head of linked list of Node<T>’s
};
```

Object-Oriented Implementation in C++

- class Node implementation

```
template <typename T> class Node
{
   friend template <typename T> class L_Stack;
public:
  Node (T i, Node<T> *n = 0): item_ (i), next_ (n) {}

private:
  T item_;
  Node<T> *next_;
};
```

- Note that the use of the “Cheshire cat” idiom allows the library writer to completely hide the representation of class V Stack...

Object-Oriented Implementation in C++

- class L Stack implementation:

```
template <typename T> L_Stack<T>::L_Stack (size_t)
  : head_ (0) {}

template <typename T> void L_Stack<T>::push (const T &it)
{
  Node<T> *t = new Node<T> (it, head_); head_ = t;
}

template <typename T> void L_Stack<T>::pop (T &top_item){
  top_item = head_->item_;
  Node<T> *t = head_; head_ = head_->next_;
  delete t;
}
```

Object-Oriented Implementation in C++

- class L_Stack implementation:

```
template <typename T> L_Stack<T>::~L_Stack (void) {
  for (T t; head_ != 0; pop (t)) continue;
}

template <typename T> L_Stack<T>::is_empty (void) const {
  return head_ == 0;
}

template <typename T> L_Stack<T>::is_full (void) const {
  ... // You fill in here ... ;-)
}
```

Object-Oriented Implementation in C++

- Using our abstract base class, it is possible to write code that does not depend on the stack implementation, e.g.,

```
Stack<int> *make_stack (bool v_Stack) {
  return v_Stack ? new V_Stack<int> : new L_Stack<int>;
}

void print_top (Stack<int> *stack) {
  std::cout << "top = " << stack->top () << std::endl;
}

int main (int argc, char **) {
  std::auto_ptr <Stack<int>> sp (make_stack (argc > 1));
  sp->push (10); // Equivalent to (*sp->vptr[1])(sp, 10);
  print_top (sp.get ());
}
```