CS-034

From C++ to C++

Thomas Doeppner
Pascal Van Hentenryck

Stacks …

```
class Stack {
public:
  Stack();
  ~Stack();
  void push(int);
  int pop();
  bool empty();
private:
  Node *top;
};
```

```
class Node {
public:
  Node(int v, Node *n):
      value(v), next(n){}
  int value;
  Node *next;
};
```

Stacks (continued)

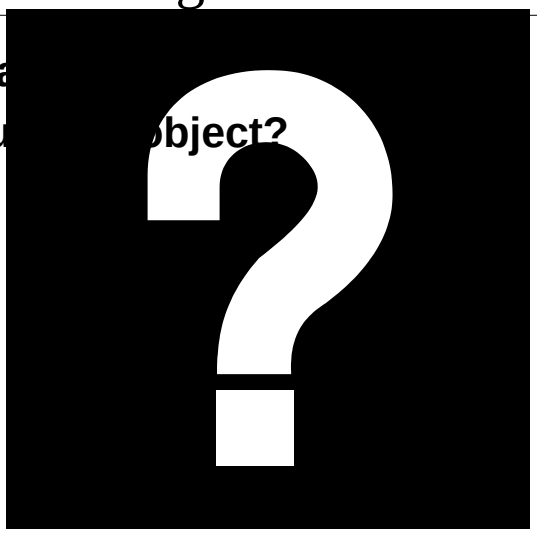
```
Stack::Stack() : top(0) { }

Stack::~Stack() {
  while(!empty())
    pop();
}

bool Stack::empty() {
  return top == 0;
}
```


Stacks (and still continued)

```
void Stack::push(int item) {
  top = new Node(item, top);
}

int Stack::pop() {
  int RetVal = top->value;
  Node *OldTop = top;
  top = top->next;
  delete OldTop;
  return RetVal;
}
```

What's Missing?

- **As a da**
- **As a ru bject?**



A Simple Data Type

```
int a=6;          // local variable
int *ap = new int;
                  // from heap

int b = a;        // copy
*ap = b;          // assignment
a = int(6.3);     // conversion
```

Copy Constructor

```
class Stack {
public:
  …
  Stack(Stack& s) {  // copy constructor
    Node **p = &top;
    for (Node *n = s.top; n; n=n->next) {
      *p = new Node(n->value, 0);
      p = &((*p)->next);
    }
  }
  …
};
```


Assignment

```
class Stack {
public:
  …
  Stack& operator=(Stack& s) {  // assignment
    while (!empty())    // delete old contents
      pop();
    Node **p = &top;
    for (Node *n = s.top; n; n=n->next) {
      *p = new Node(n->value, 0);
      p = &((*p)->next);
    }
  }
  …
};
```

Another Stack …

```
class PascalStack {
  int st[MaxStack];
  int top;
public:
  PascalStack() : top(0) { }
  void push(int v) { st[top++] = v; }
  int pop() { return st[--top]; }
  bool empty() { return top == 0; }
};
```


Conversion

```
class Stack {
public:
  …
  Stack& operator=(PascalStack& s) {  // assignment
    while (!empty())    // delete old contents
      pop();
    Node **p = &top;
    for (int i = s.top; i>0; i--) {
      *p = new Node(s.st[i-1], 0);
      p = &((*p)->next);
    }
  }
  …
};
```

Friendship Required …

```
class PascalStack {
  int st[MaxStack];
  int top;
public:
  PascalStack() : top(0) { }
  void push(int v) { st[top++] = v; }
  int pop() { return st[--top]; }
  bool empty() { return top == 0; }
  friend Stack& Stack::operator=(PascalStack&);
};
```


What We've Accomplished …

```
int main() {
  Stack s1;
  s1.push(1); s1.push(2);
  Stack s2(s1);
  s2.push(9);
  s1 = s2;
  PascalStack ps;
  ps.push(15); ps.push(16); ps.push(17);
  s2 = ps;
  …
}
```

But …

I want a stack of floats!

I want a stack of doubles!

I want a stack of pages!

I want a stack of files!

I want a stack of sockets!

I want a stack of tables!

I want a stack of complex!

I want a stack of photos!

I want a stack of records!

I want a stack of dogs!

I want a stack of cats!

I want a stack of threads!

I want a stack of bitmaps!

I want a stack of CDs!

I want a stack of money!

Templates to the Rescue

```
Stack<float>   s1;
Stack<bitmap>  s2;
Stack<socket>  s3;
Stack<page>    s4;
Stack<money>   s5;
```

Types as Parameters

```
template<class C> class PascalStack {
  C st[MaxStack];
  int top;
public:
  PascalStack() : top(0) { }
  void push(C v) { st[top++] = v; }
  C pop() { return st[--top]; }
  bool empty() { return top == 0; }
};
```


Dealing with Problems

```
int main() {
  Stack S;

  int i = S.pop();
     // how do we check for stack underflow?
  while(1)
     S.push(6);
     // what about stack overflow?
}
```

Possibilities …

- **Return an error indication**

```
int i = S.pop();
if (i == NoGood)
    cerr << "you blew it" << endl;
    // what's "NoGood"?
T x = S.push(6);
    // big change to push
```


Possibilities …

- **Set a value someplace**

```
int i = S.pop();
if (error)
    cerr << "you blew it" << endl;
S.push(6);
if (error)
    cerr << "you blew it" << endl;
// little change to push
// But where's error?
```

More Concerns

- **Errors might be ignored**
- **It might not be clear what to do about them**

Exceptions: Catching

```
int main() {
  PascalStack s;
  …
  try {
    s.pop();
  } catch (StackUnderflow) {
    cerr << "Don't do that!" << endl;
  }
  …
}
```

Exceptions: Throwing

```
class StackUnderflow {};
class PascalStack {
…
  int pop() {
    if (top == 0) throw StackUnderflow();
    return st[--top];
  }
…
};
```


Exceptions

- **Cannot be ignored**
- **Propagated from module to module**

Propagating Exceptions

```
int main() {
  …
  try {
    Stack S1, S2;
    use(S1);
    …
  } catch
     (StackUnderflow) {
    cleanup();
  }
  …
}
```

```
void use(Stack& s) {
  Stack S3;
  S3.pop();
  try {
    Stack S4;
    s.pop();
    …
  } catch
     (StackUnderflow) {
    cout << "!";
    throw;
  }
}
```


Exception Objects

```
class BadChar {
public:
  BadChar(char x): c(x){}
private:
  char c;
};
int convert(char a) {
  if (a > 'f')
    throw BadChar(a);
  …
}
```

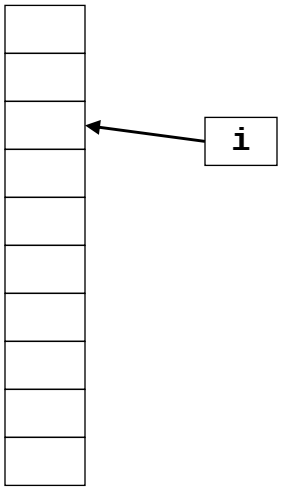
```
…
try {
  convert(z);
  …
} catch (BadChar x) {
  cout << x << "is bad";
}
…
```

And More Exceptions

```
try {
  something_interesting();
} catch (StackUnderflow) {
  …
} catch (StackOverflow) {
  …
} catch (BadChar a) {
  …
} catch (…) {
  cout<<"Unexpected exception" << endl;
}
```


Iterators

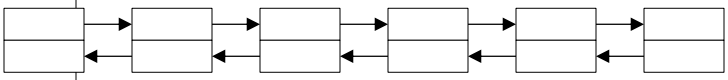
```
float A[10];

for (int i=0; i<10; ++i)
  cout << A[i];

// i is an iterator
```



A List

```
class List {
public:
  List();
  ~List();
  void insertFront(int);
  Node *front;
  Node *rear;
};
```

```
class Node {
public:
  Node(int, Node*, Node*);
  int value;
  Node *prev;
  Node *next;
};
```



Iterating: Approach 1

```
List L;
L.insertFront(1); L.insertFront(2);
L.insertFront(3); L.insertFront(4);

for (Node *i = L.front; i!= 0; i = i->next)
  cout << i->value;
```

Iterating: Approach 2

```
List L;
L.insertFront(1); L.insertFront(2);
L.insertFront(3); L.insertFront(4);

for (iterator i = L.begin(); i!= L.end(); i++)
  cout << i->value;
```


Iterator Declaration

```
class iterator {
public:
  Node* operator->();
  Node operator*();
  iterator operator==(iterator);
  iterator operator!=(iterator);
  iterator operator++();
  iterator operator++(int);
  iterator operator--();
private:
  iterator(Node*);
  Node *pointer;
};
```

Begin and End

```
class List {
public:
  …
  iterator begin() {
    return iterator(front);
  }
  iterator end() {
    return iterator(0);
  }
}
```


Easy Operators

```
iterator::operator->() {
  return pointer;
}
iterator::operator*() {
  return *pointer;
}
iterator::operator==(iterator i) {
  return this->pointer == i.pointer;
}
```

Tougher Operators

```
iterator::operator++() {
  pointer = pointer->next;
  return *this;
}
iterator::operator--() {
  pointer = pointer->prev;
  return *this;
}
```


Toughest Operators

```
iterator::operator++(int) {
  iterator ret(pointer);
  pointer = pointer->next;
  return ret;
}
iterator::operator--(int) {
  iterator ret(pointer);
  pointer = pointer->prev;
  return ret;
}
```

Namespaces: Less than Classes

- **Classes define types**
  - **class complex**
  - **class stack**
  - **class list**
- **Namespaces define modules**
  - **std**

Using Namespaces

```
#include <iostream>   // std I/O
#include <string>     // std strings

  …
  std::cout << "Hello world!" << endl;

  using namespace std;
  cout << "Hello world!" << endl;
```

Defining Namespaces

```
namespace MathLib {
  float sin(float);
  float cos(float);
  float tan(float);
  …
};
```

```
  …
  a = MathLib::sin(b);

using namespace
    Mathlib;
  a = sin(b);
```


Constants

```
#define PI = 3.1415926536
const float pi = 3.1415926536;
const int A[] = {1,2,3};
const char c;
const float *p1 = &pi;   // ptr to constant
float *p2 = &pi;         // illegal
int func(const int* p) {
  *p = 6; // illegal
  …
}
```

And More …

```
const char *pc = string;
     // pointer to constant
pc[1] = 's';   // illegal
pc = string2;  // legal
char *const cp = string;
     // constant pointer
cp[0] = 't';   // legal
cp = string2;  // illegal
const char *const cpc = string;
     // constant pointer to constant
```


And Still More …

```
class xyz {
public:
  xyz(int v) : val(v){};
  void update(int){val += 7;}
  int GetValue() const {return val;}
private:
  int val;
}
```