What I learned about language and library design by working on Swift

Dave Abrahams | Principal Scientist | STLab

Artwork by **Elise Swopes** / USA

About the artist

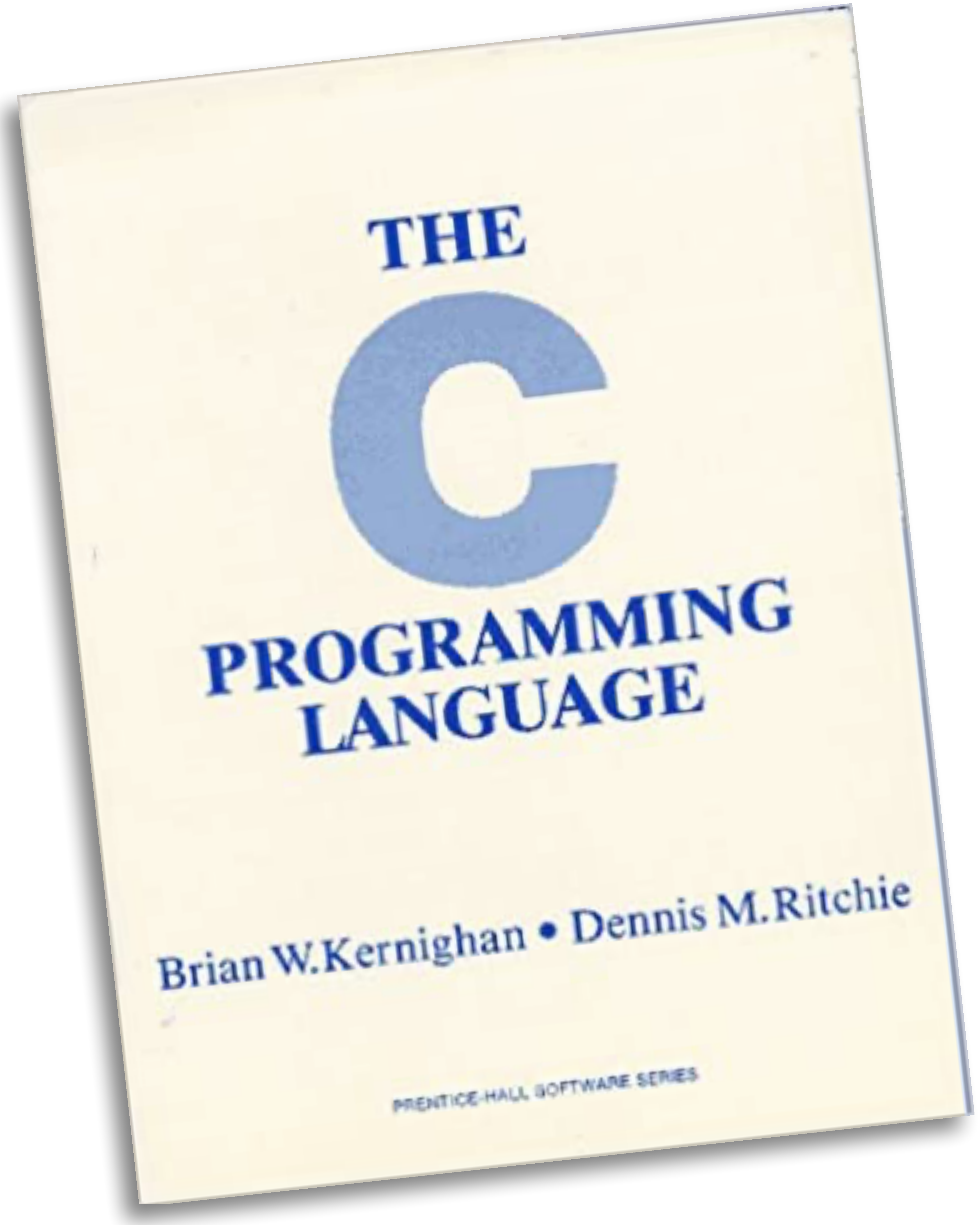
Elise Swopes

New York City-based photographer and graphic designer Elise Swopes manipulates the everyday into unexpected works of art. With her mobile phone, Adobe Photoshop, and Adobe Photoshop Lightroom, she creates whimsical, weird, dreamlike scenes. This piece was inspired from a recent visit to Copenhagen, where the bold colors and shapes of Danish architecture made quite an impression. A brilliant yellow apartment building inspired Swopes to create another entry in her surrealistic series featuring giraffes in unusual settings.

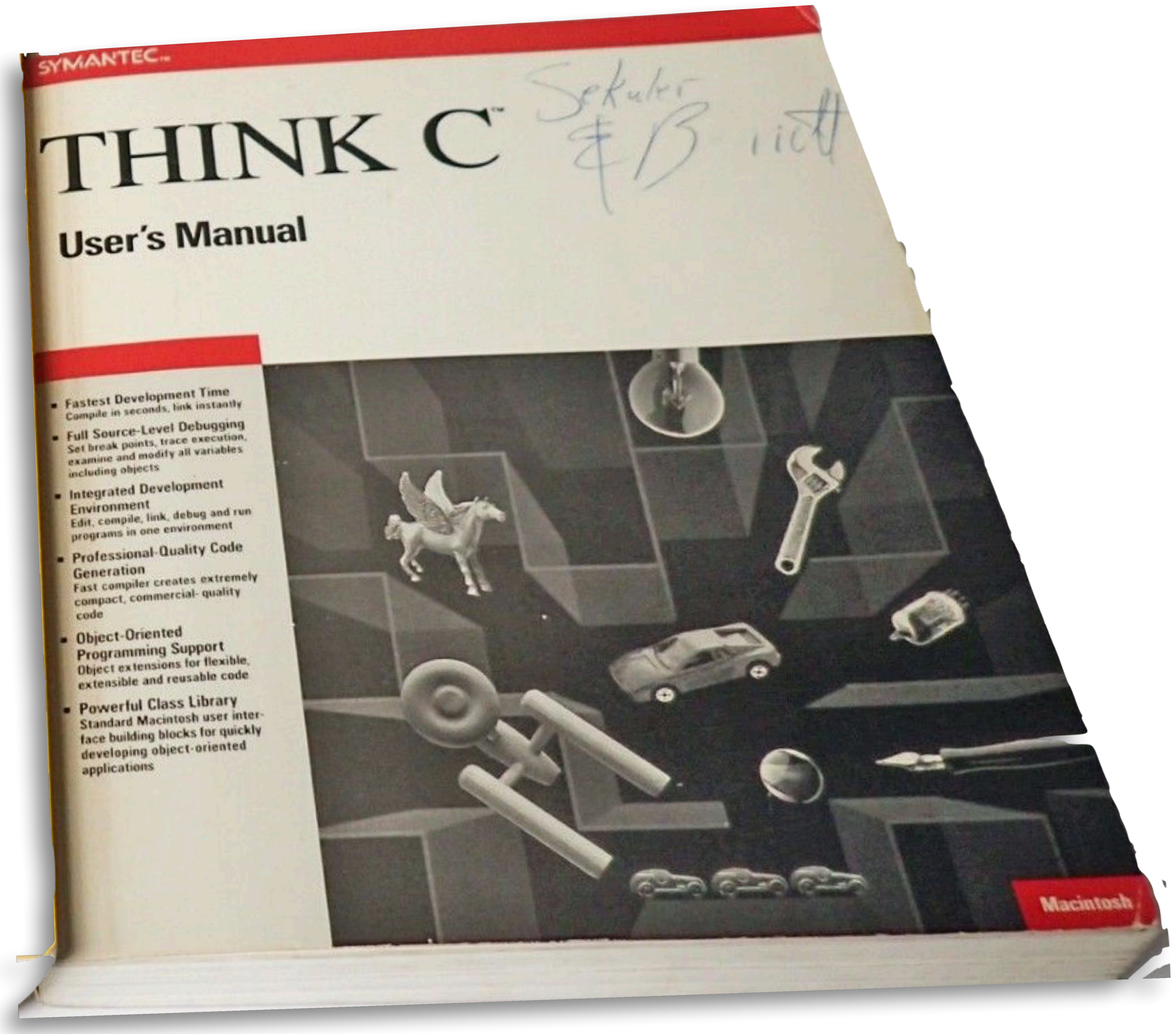
Made with

Ps Adobe Photoshop   Lr Adobe Photoshop Lightroom

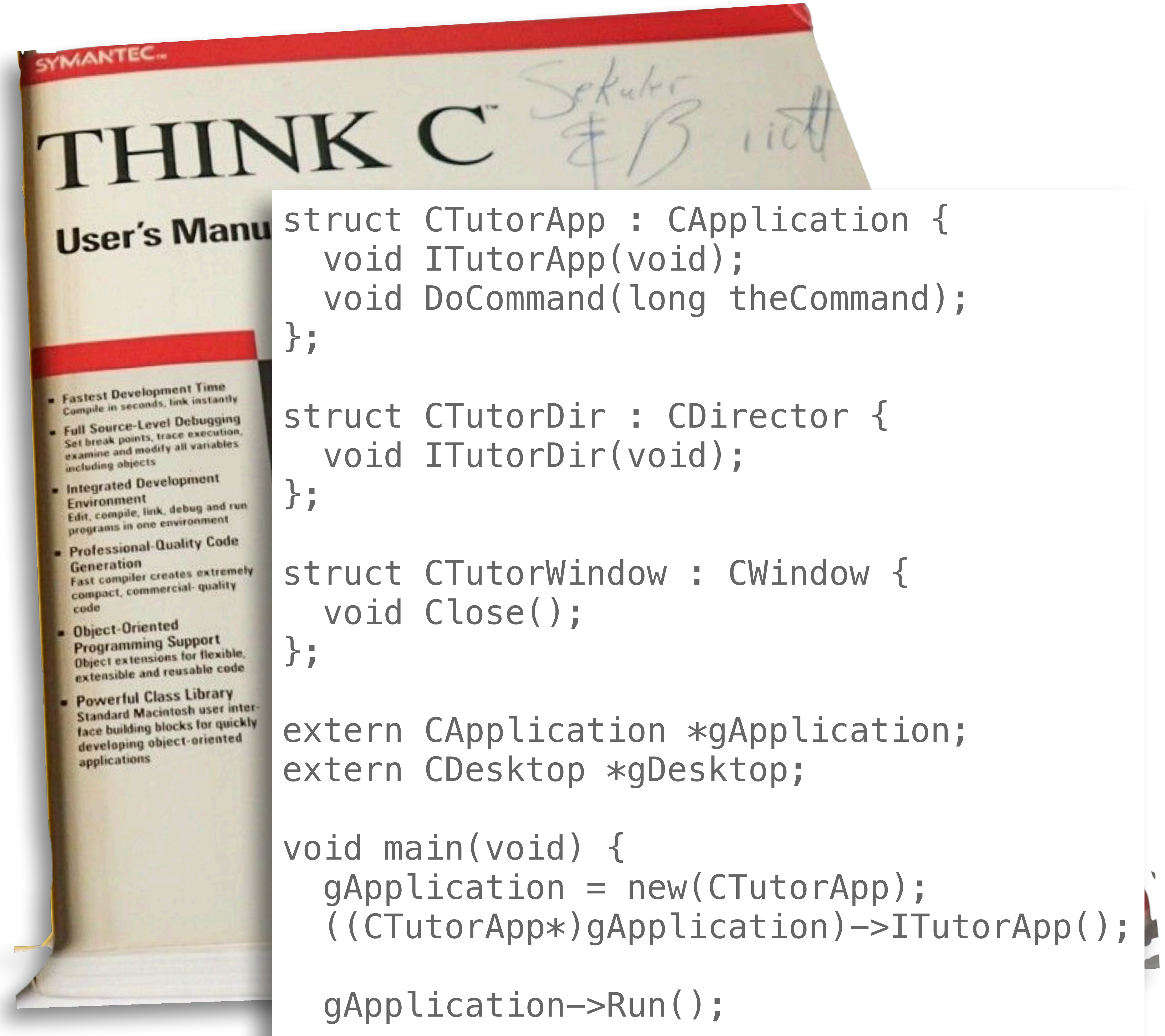
Mark of the Unicorn | 1988



Mark of the Unicorn | 1991

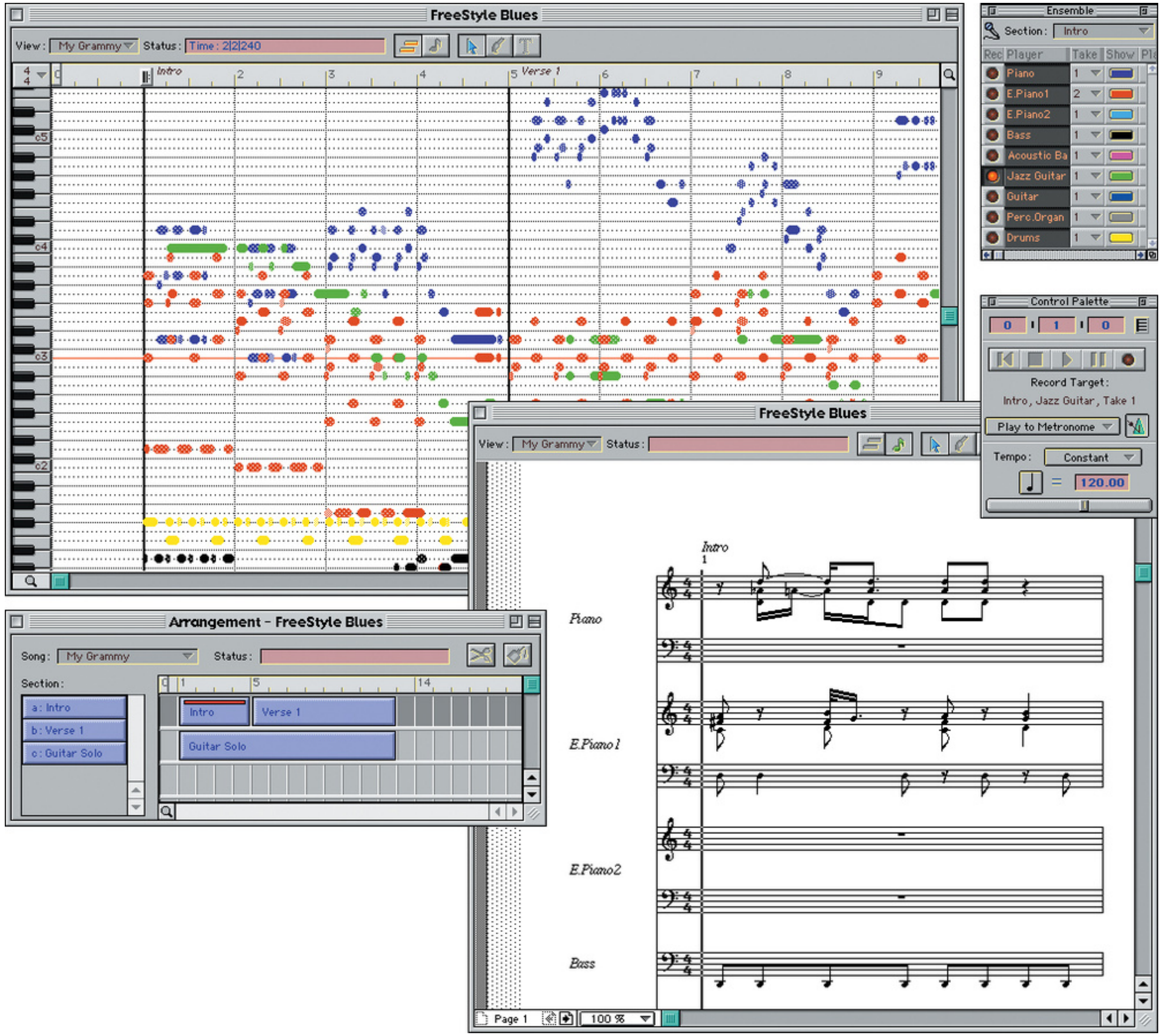


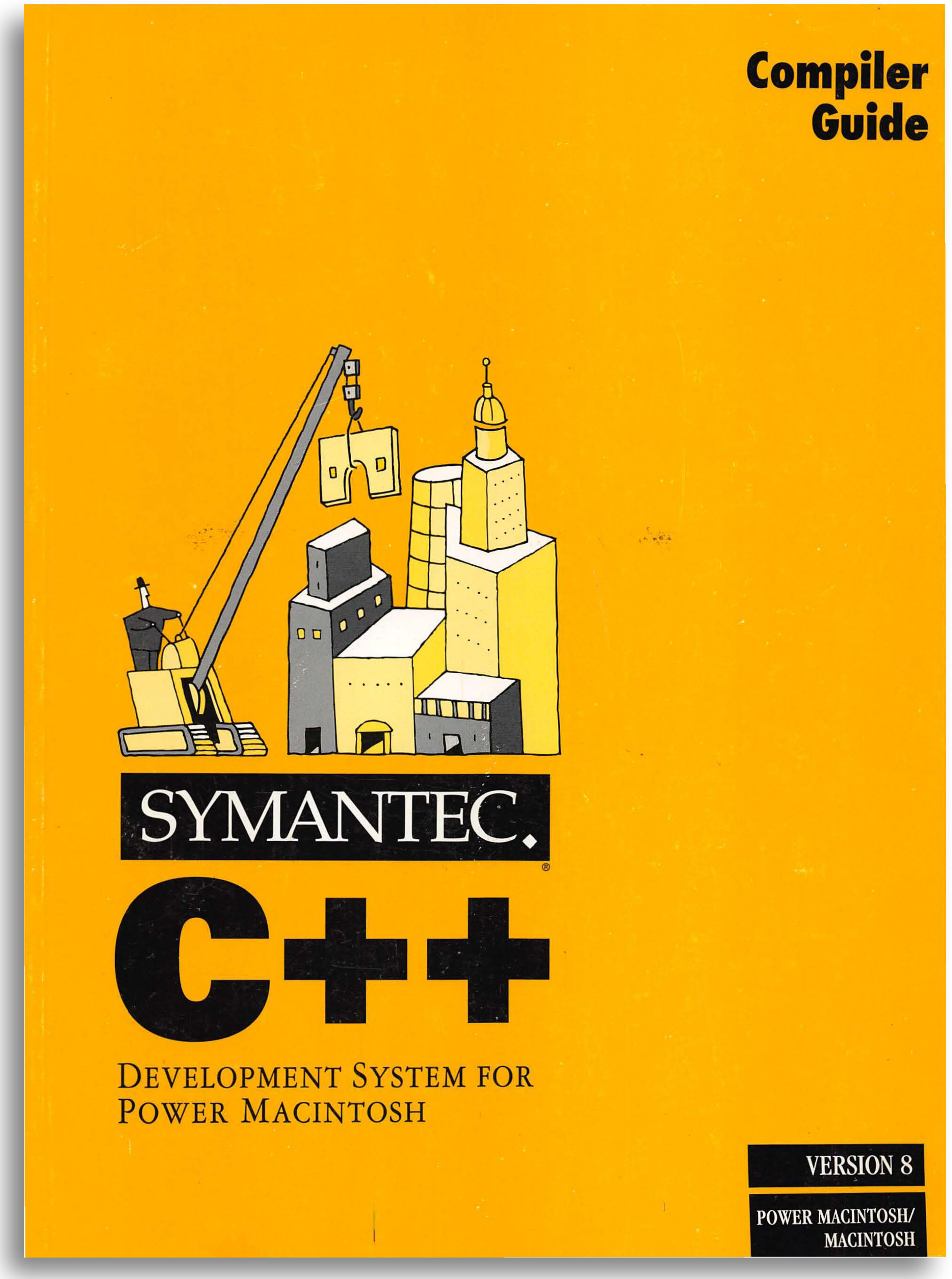
Mark of the Unicorn | 1991




```
struct CTutorApp : CApplication {
  void ITutorApp(void);
  void DoCommand(long theCommand);
};

struct CTutorDir : CDirector {
  void ITutorDir(void);
};

struct CTutorWindow : CWindow {
  void Close();
};

extern CApplication *gApplication;
extern CDesktop *gDesktop;

void main(void) {
  gApplication = new(CTutorApp);
  ((CTutorApp*)gApplication)->ITutorApp();

  gApplication->Run();

  gApplication->Exit();
}
```

Mark of the Unicorn | 1995





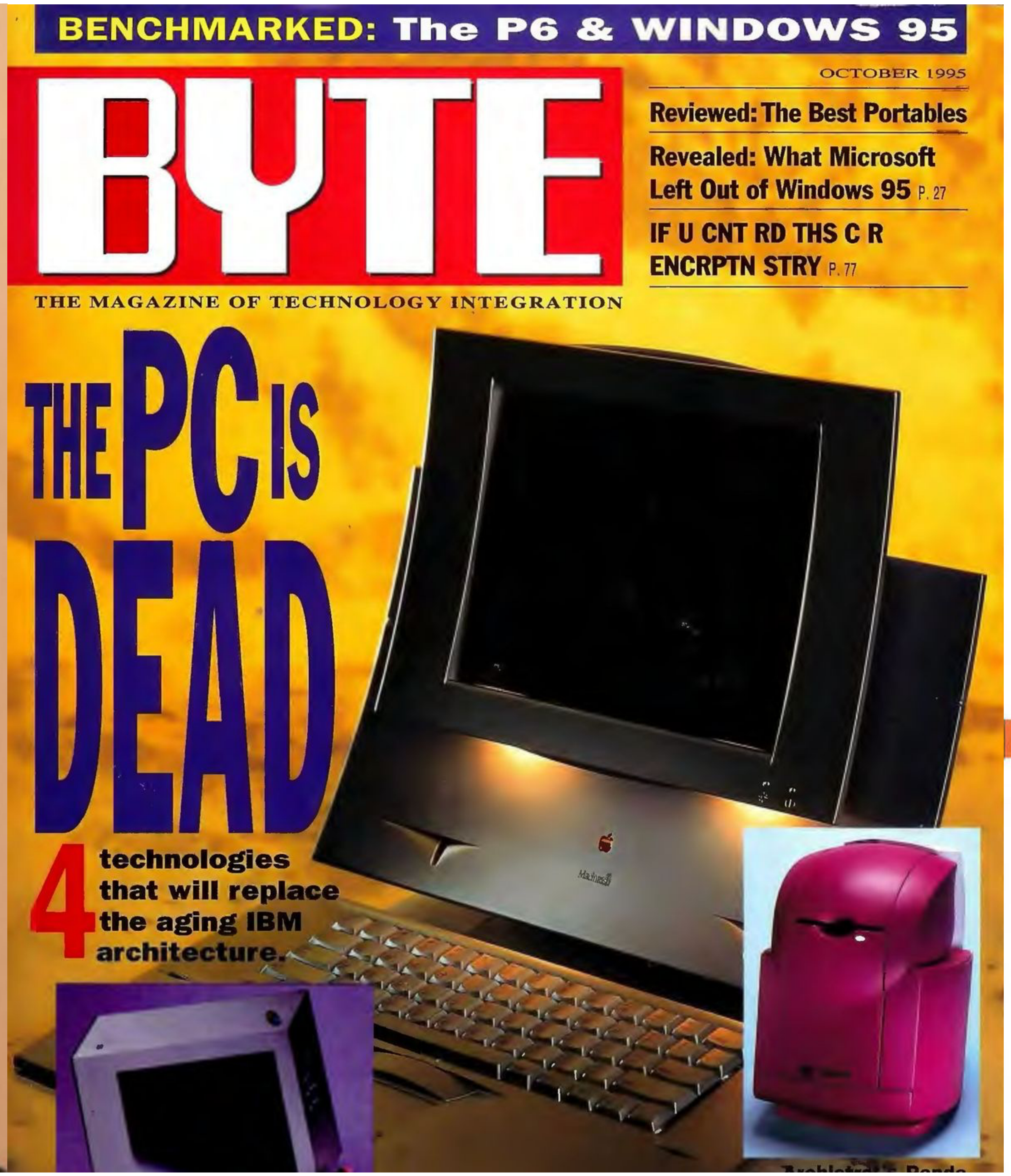
Blame it all on this guy…

Mark Wachsler

Blame it all on this guy...

Mark Wachsler

...and his tendency to learn...

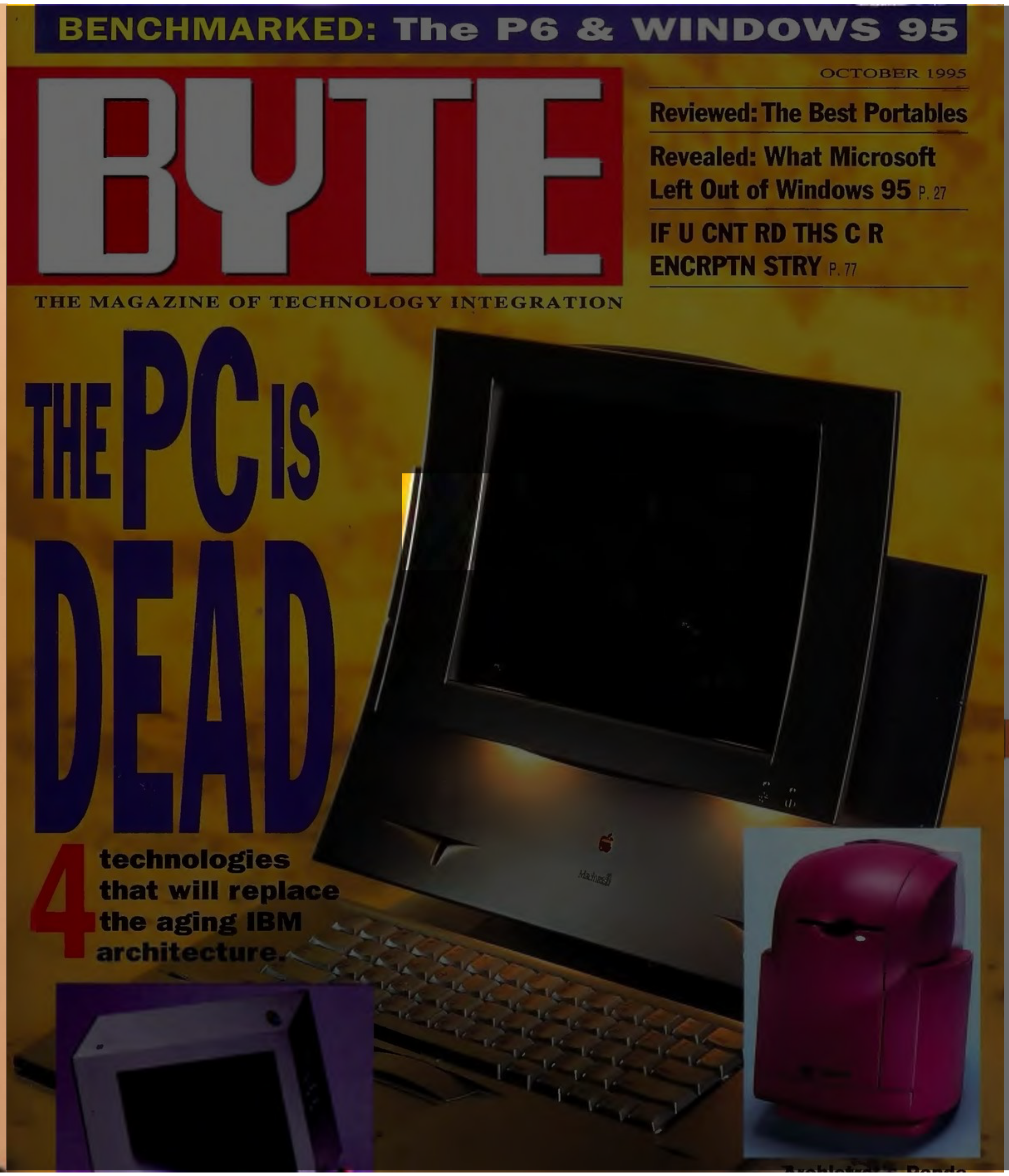


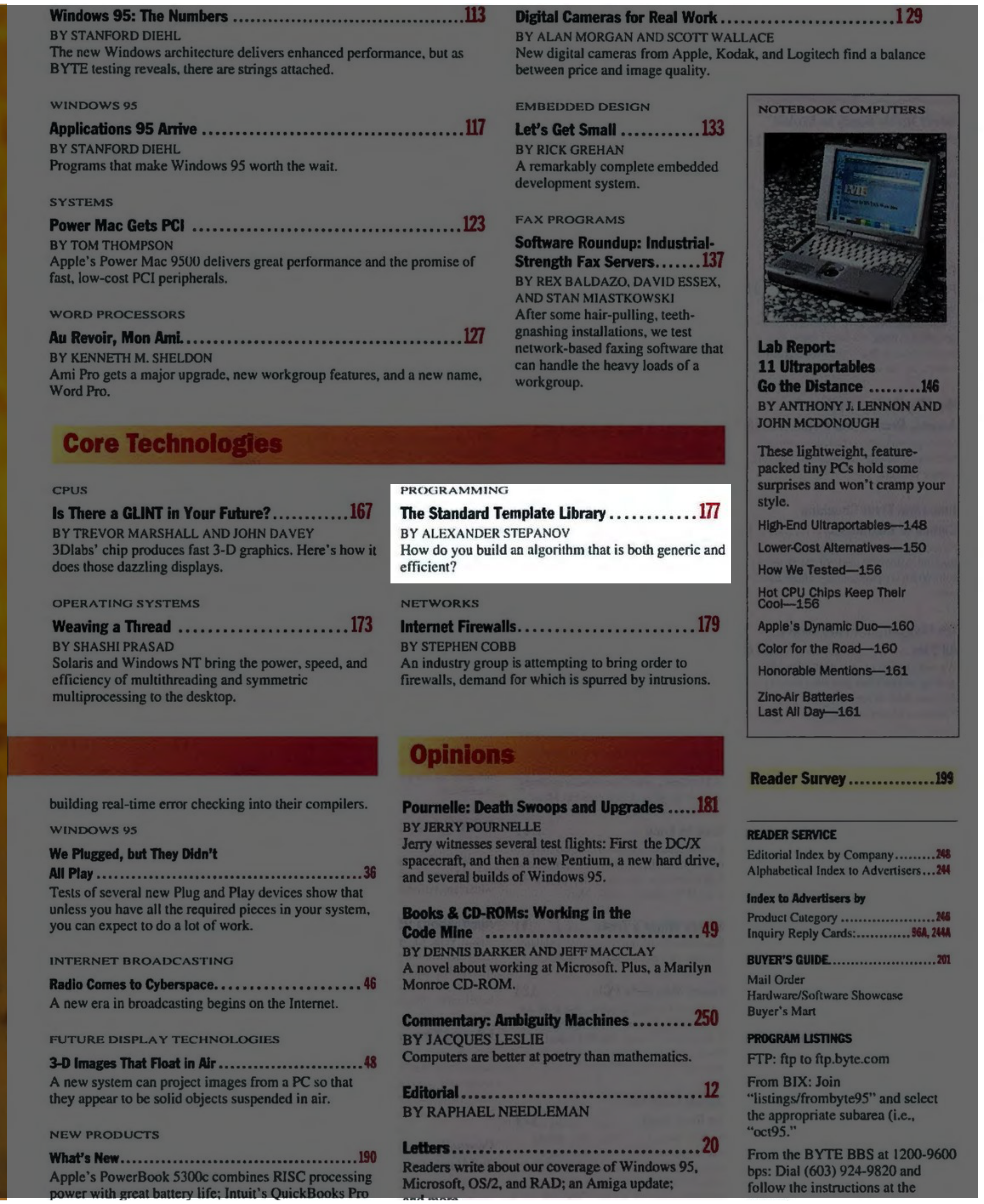
...and to share

PROGRAM LISTINGS
FTP: ftp to ftp.byte.com
From BIX: Join "listings/frombyte95" and select the appropriate subarea (i.e., "oct95."
From the BYTE BBS at 1200-9600 bps: Dial (603) 924-9820 and follow the instructions at the

Blame it all on this guy…

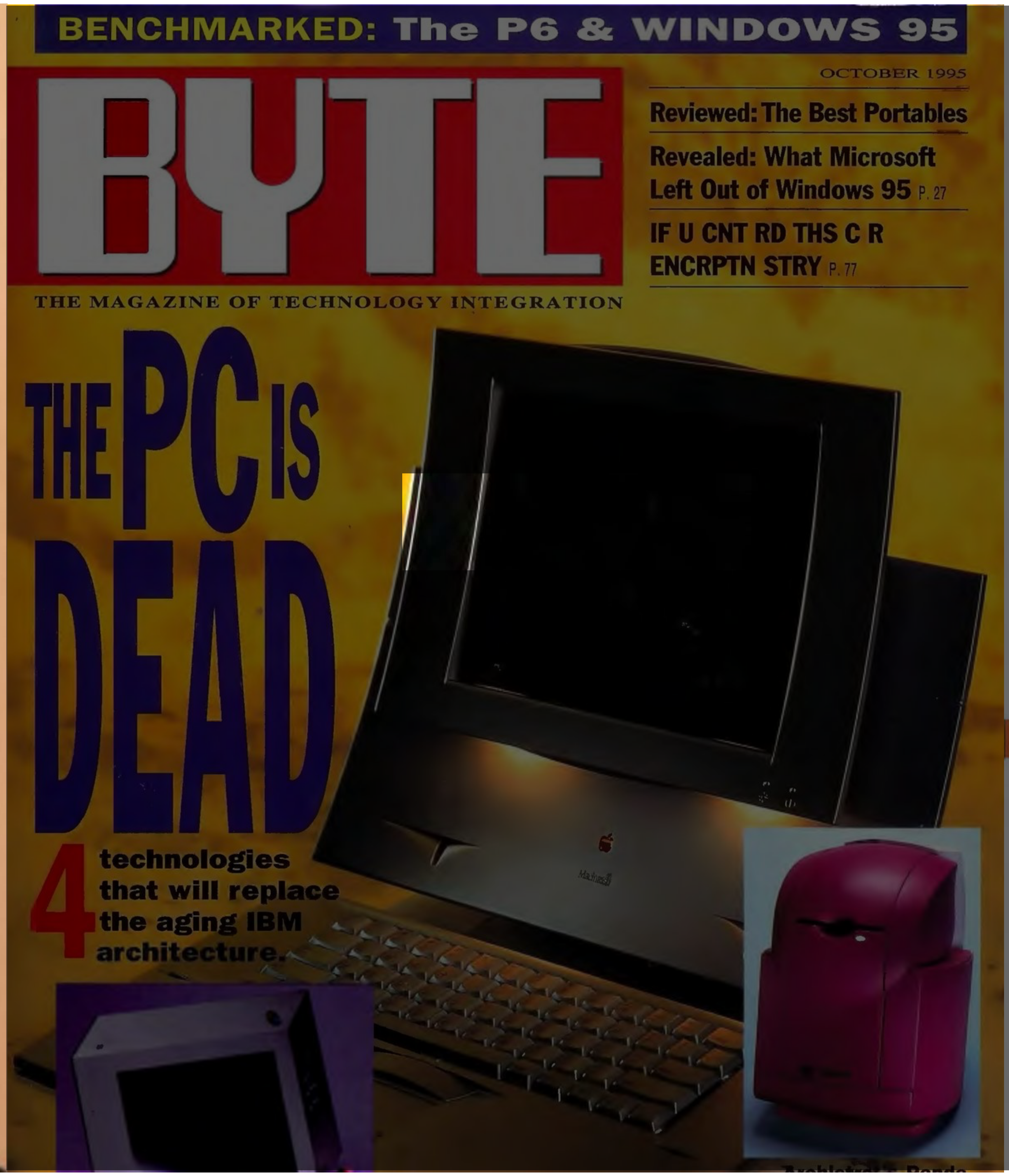
Mark Wachsler | **…and his tendency to learn…** | **…and to share**

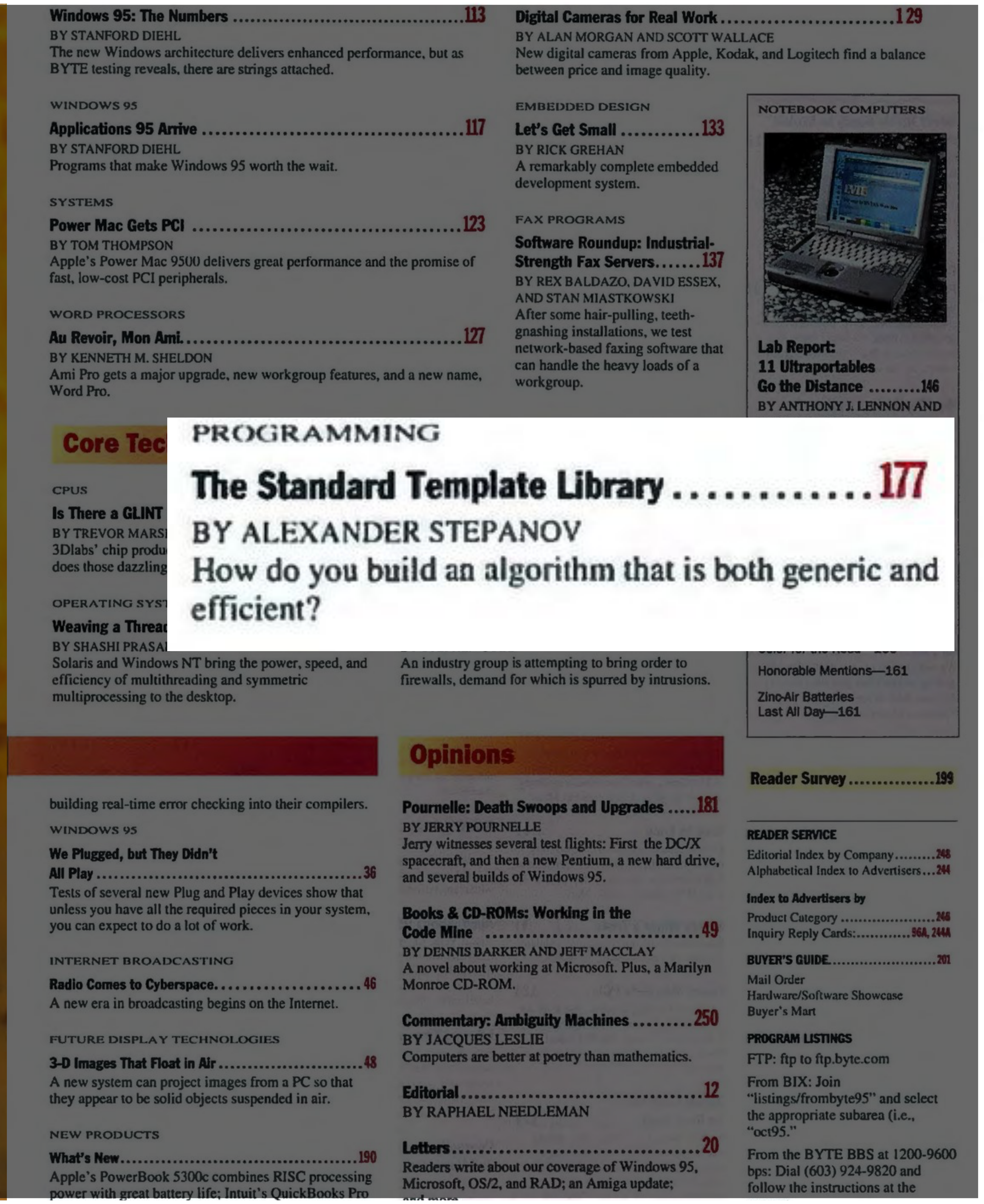




Blame it all on this guy…

Mark Wachsler | **…and his tendency to learn…** | **…and to share**




Windows 95: The Numbers 113
BY STANFORD DIEHL
The new Windows architecture delivers enhanced performance, but as BYTE testing reveals, there are strings attached.

WINDOWS 95
Applications 95 Arrive 117
BY STANFORD DIEHL
Programs that make Windows 95 worth the wait.

SYSTEMS
Power Mac Gets PCI 123
BY TOM THOMPSON
Apple's Power Mac 9500 delivers great performance and the promise of fast, low-cost PCI peripherals.

WORD PROCESSORS
Au Revoir, Mon Ami 127
BY KENNETH M. SHELDON
Ami Pro gets a major upgrade, new workgroup features, and a new name, Word Pro.

Digital Cameras for Real Work 129
BY ALAN MORGAN AND SCOTT WALLACE
New digital cameras from Apple, Kodak, and Logitech find a balance between price and image quality.

EMBEDDED DESIGN
Let's Get Small 133
BY RICK GREHAN
A remarkably complete embedded development system.

FAX PROGRAMS
Software Roundup: Industrial-Strength Fax Servers....137
BY REX BALDAZO, DAVID ESSEX, AND STAN MIASTKOWSKI
After some hair-pulling, teeth-gnashing installations, we test network-based faxing software that can handle the heavy loads of a workgroup.

NOTEBOOK COMPUTERS
Lab Report: 11 Ultraportables Go the Distance 146
BY ANTHONY J. LENNON AND

PROGRAMMING
The Standard Template Library 177
BY ALEXANDER STEPANOV
How do you build an algorithm that is both generic and efficient?

Honorable Mentions—161
Zinc-Air Batteries Last All Day—161

Solaris and Windows NT bring the power, speed, and efficiency of multithreading and symmetric multiprocessing to the desktop.

An industry group is attempting to bring order to firewalls, demand for which is spurred by intrusions.

building real-time error checking into their compilers.

WINDOWS 95
We Plugged, but They Didn't All Play 36
Tests of several new Plug and Play devices show that unless you have all the required pieces in your system, you can expect to do a lot of work.

INTERNET BROADCASTING
Radio Comes to Cyberspace....46
A new era in broadcasting begins on the Internet.

FUTURE DISPLAY TECHNOLOGIES
3-D Images That Float in Air 48
A new system can project images from a PC so that they appear to be solid objects suspended in air.

NEW PRODUCTS
What's New....190
Apple's PowerBook 5300c combines RISC processing power with great battery life; Intuit's QuickBooks Pro

Opinions
Pournelle: Death Swoops and Upgrades 181
BY JERRY POURNELLE
Jerry witnesses several test flights: First the DC/X spacecraft, and then a new Pentium, a new hard drive, and several builds of Windows 95.

Books & CD-ROMs: Working in the Code Mine 49
BY DENNIS BARKER AND JEFF MACCLAY
A novel about working at Microsoft. Plus, a Marilyn Monroe CD-ROM.

Commentary: Ambiguity Machines 250
BY JACQUES LESLIE
Computers are better at poetry than mathematics.

Editorial 12
BY RAPHAEL NEEDLEMAN

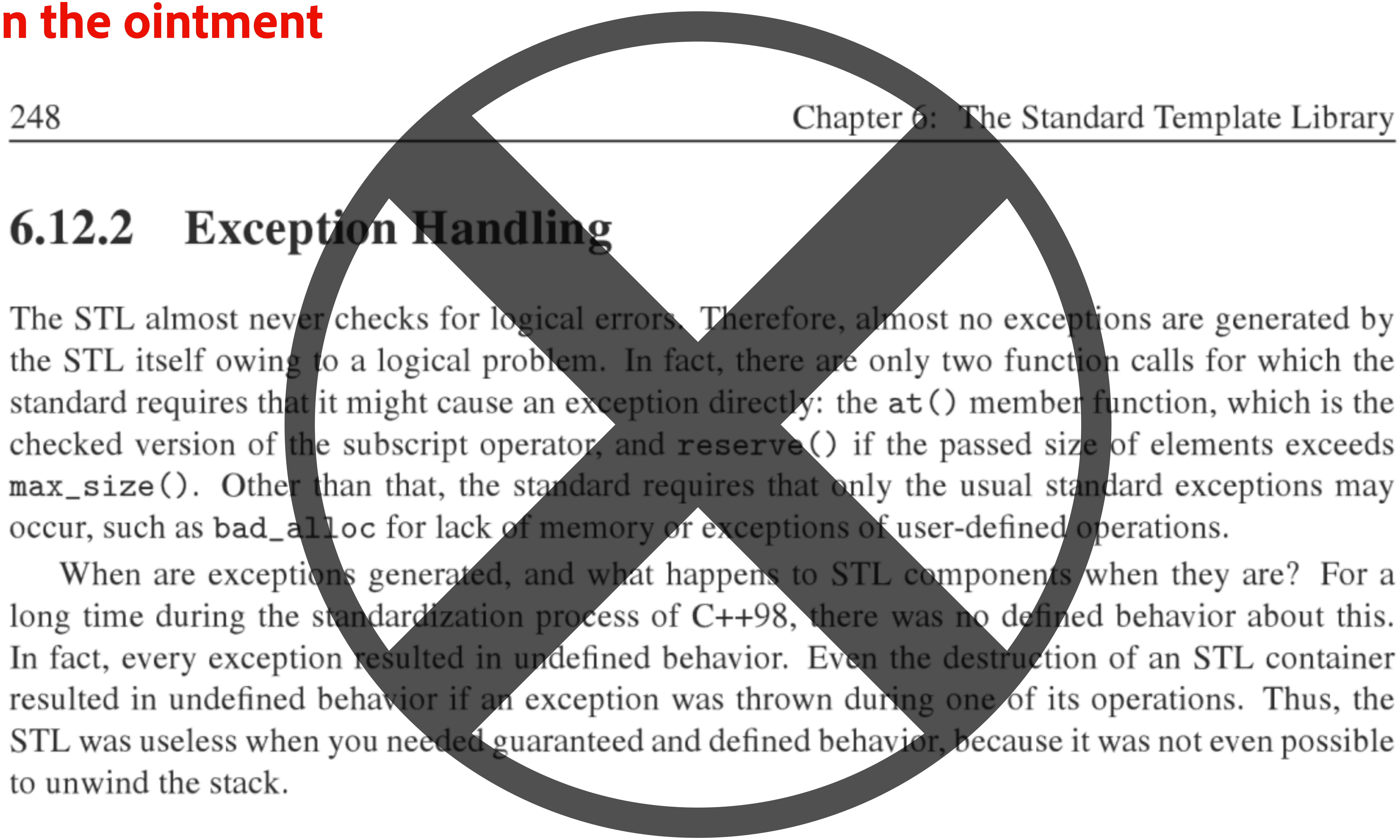
Letters....20
Readers write about our coverage of Windows 95, Microsoft, OS/2, and RAD; an Amiga update;

Reader Survey 199

READER SERVICE
Editorial Index by Company....
Alphabetical Index to Advertisers....
Index to Advertisers by Product Category ....
Inquiry Reply Cards:....
BUYER'S GUIDE....
Mail Order
Hardware/Software Showcase
Buyer's Mart

PROGRAM LISTINGS
FTP: ftp to ftp.byte.com
From BIX: Join "listings/frombyte95" and select the appropriate subarea (i.e., "oct95."
From the BYTE BBS at 1200-9600 bps: Dial (603) 924-9820 and follow the instructions at the

A fly in the ointment


248 Chapter 6: The Standard Template Library

6.12.2 Exception Handling

The STL almost never checks for logical errors. Therefore, almost no exceptions are generated by the STL itself owing to a logical problem. In fact, there are only two function calls for which the standard requires that it might cause an exception directly: the `at()` member function, which is the checked version of the subscript operator, and `reserve()` if the passed size of elements exceeds `max_size()`. Other than that, the standard requires that only the usual standard exceptions may occur, such as `bad_alloc` for lack of memory or exceptions of user-defined operations.

When are exceptions generated, and what happens to STL components when they are? For a long time during the standardization process of C++98, there was no defined behavior about this. In fact, every exception resulted in undefined behavior. Even the destruction of an STL container resulted in undefined behavior if an exception was thrown during one of its operations. Thus, the STL was useless when you needed guaranteed and defined behavior, because it was not even possible to unwind the stack.

A fly in the ointment

6.12.2 Exception Handling

The STL almost never checks for logical errors. Therefore, almost no exceptions are generated by the STL itself owing to a logical problem. In fact, there are only two function calls for which the standard requires that it might cause an exception directly: the `at()` member function, which is the checked version of the subscript operator, and `reserve()` if the passed size of elements exceeds `max_size()`. Other than that, the standard requires that only the usual standard exceptions may occur, such as `bad_alloc` for lack of memory or exceptions of user-defined operations.

When are exceptions generated, and what happens to STL components when they are? For a long time during the standardization process of C++98, there was no defined behavior about this. In fact, every exception resulted in undefined behavior. Even the destruction of an STL container resulted in undefined behavior if an exception was thrown during one of its operations. Thus, the STL was useless when you needed guaranteed and defined behavior, because it was not even possible to unwind the stack.

Dave's chain of custody

Alexander Stepanov

Greg Colvin

Andrew Koenig

C++ Committee Papers | 1997-2013

- N1075 | STL Exception Handling Contract | Dave Abrahams | 1997
- N1086 | Making the C++ Standard Library Exception-Safe | Dave Abrahams and Greg Colvin | 1997
- N1086 | Making the C++ Standard Library Exception Safe | Dave Abrahams and Greg Colvin | 1997
- N1114 | Making the C++ Standard Library More Exception Safe | Dave Abrahams and Greg Colvin | 1997
- N1313 | Binary Search with Heterogeneous Comparison | David Abrahams | 2001
- N1356 | Predictable data layout for certain non-POD types | R.W. Grosse-Kunstleve & D. Abrahams | 2002
- N1377 | A Proposal to Add Move Semantics Support to the C++ Language | H. Hinnant, P. Dimov, D. Abrahams 2002
- N1408 | Qualified Namespaces | David Abrahams | 2002
- N1476 | Iterator Facade and Adaptor | D. Abrahams, J. Siek, T. Witt | 2003
- N1477 | New Iterator Concepts | D. Abrahams, J. Siek, T. Witt | 2003
- N1530 | Iterator Facade and Adaptor | D. Abrahams, J. Siek, T. Witt | 2003
- N1531 | New Iterator Concepts | D. Abrahams, J. Siek, T. Witt | 2003
- N1550 | New Iterator Concepts | D. Abrahams, J. Siek, T. Witt | 2003
- N1610 | Clarification of Initialization of Class Objects by rvalues | D. Abrahams, G. Powell | 2004-02-14
- N1631 | Electronic review process | D. Abrahams, B. Dawes, J. Siek | 2004-04-11
- N1640 | New Iterator Concepts | D. Abrahams, J. Siek, T. Witt | 2004-04-10
- N1641 | Iterator Facade and Adaptor | D. Abrahams, J. Siek, T. Witt | 2004-04-10
- N1690 | A Proposal to Add an Rvalue Reference to the C++ Language | H. Hinnant,D. Abrahams,P. Dimov | 2004-09-07
- N1691 | Explicit Namespaces | David Abrahams | 2004-09-07
- N1770 | A Proposal to Add an Rvalue Reference to the C++ Language: Proposed Wording | H. Hinnant, D. Abrahams, J. Adamczyk, P. Dimov, A. Hommel | 2005-03-05
- N1771 | Impact of the rvalue reference on the Standard Library
- H. Hinnant, D. Abrahams, P. Dimov, D. Gregor, A. Hommel, A. Meredith | 2005-03-03
- N1773 | Proposal to add Contract Programming to C++ (revision 2) | D. Abrahams, L. Crowl, T. Ottosen, J. Widman | 2005-03-04
- N1855 | A Proposal to Add an Rvalue Reference to the C++ Language: Proposed Wording | D. Abrahams, P. Dimov, H. Hinnant, A. Hommel | 2005-08-25
- N1873 | The Cursor/Property Map Abstraction | D. Kühl, D. Abrahams | 2005-08-26
- N2786 | Simplifying unique copy (Revision 1) | D. Gregor, D. Abrahams | 2008-09-19
- N2812 | A Safety Problem with RValue References (and what to do about it) | D. Abrahams, D. Gregor | 2008-12-05
- N2831 | Fixing a Safety Problem with Rvalue References: Proposed Wording | D. Gregor, D. Abrahams | 2009-02-07
- N2844 | Fixing a Safety Problem with Rvalue References: Proposed Wording (Revision 1) | D. Gregor, D. Abrahams | 2009-03-05
- N2845 | Remove std::reference_closure | L. Crowl, D. Gregor, D. Abrahams | 2009-03-05
- N2855 | Rvalue References and Exception Safety | D. Gregor, D. Abrahams | 2009-03-23
- N2916 | Intentional Concept Mapping | D. Abrahams, B. Dawes | 2009-06-22
- N2918 | Exported Concept Maps | D. Abrahams, D. Gregor | 2009-06-22
- N2983 | Allowing Move Constructors to Throw | D. Abrahams, R. Sharoni, D. Gregor | 2009-11-09
- N3050 | Allowing Move Constructors to Throw (Rev. 1) | D. Abrahams, R. Sharoni, D. Gregor | 2010-03-12
- N3153 | Implicit Move Must Go | Dave Abrahams | 2010-10-17
- N3418 | Proposal for Generic (Polymorphic) Lambda Expressions | F. Vali, H. Sutter, D. Abrahams | 2012-09-21
- N3490 | ADL Control for C++ | Dave Abrahams | 2012-10-31
- N3559 | Proposal for Generic (Polymorphic) Lambda Expressions
- F. Vali, H. Sutter, D. Abrahams | 2013-03-17
- N3560 | Proposal for Assorted Extensions to Lambda Expressions | F. Vali, H. Sutter, D. Abrahams | 2013-03-17
- N3649 Generic (Polymorphic) Lambda Expressions (Revision 3) | F. Vali, H. Sutter, D. Abrahams | 2013-04-19

boost C++ LIBRARIES | 1998-?

Beman Dawes, R.I.P.

Brainchild of Beman Dawes

I was a co-founder

Spun off BoostPro Computing

Started BoostCon/C++Now conference

I was a total C++-head…

…and I was starting over

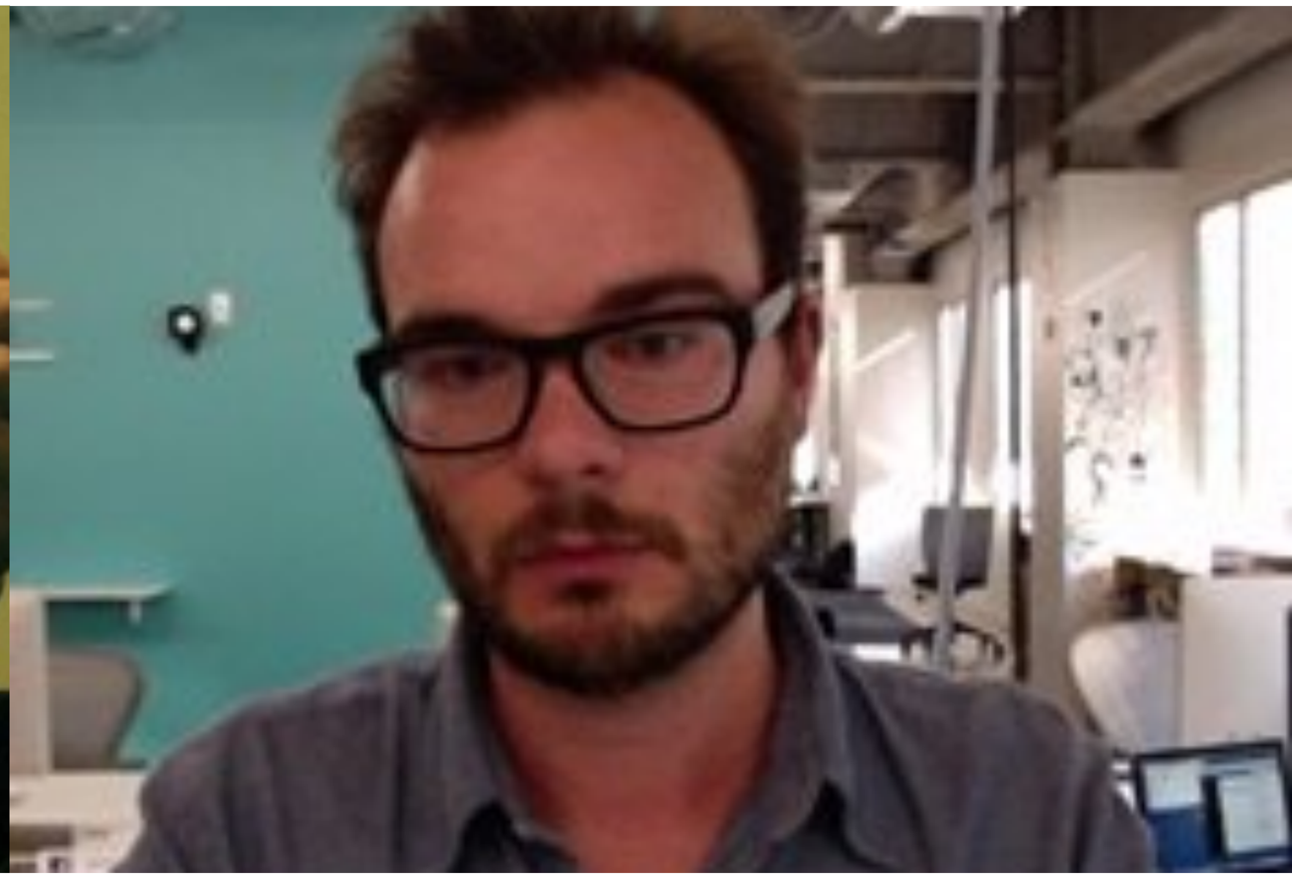
The (rest of the) design team

Dave Zarzycki

Doug Gregor

John McCall

Joe Pamer



Jordan Rose

Chris Lattner

Ted Kremenek

Joe Groff

Value Semantics

Spooky action at a distance

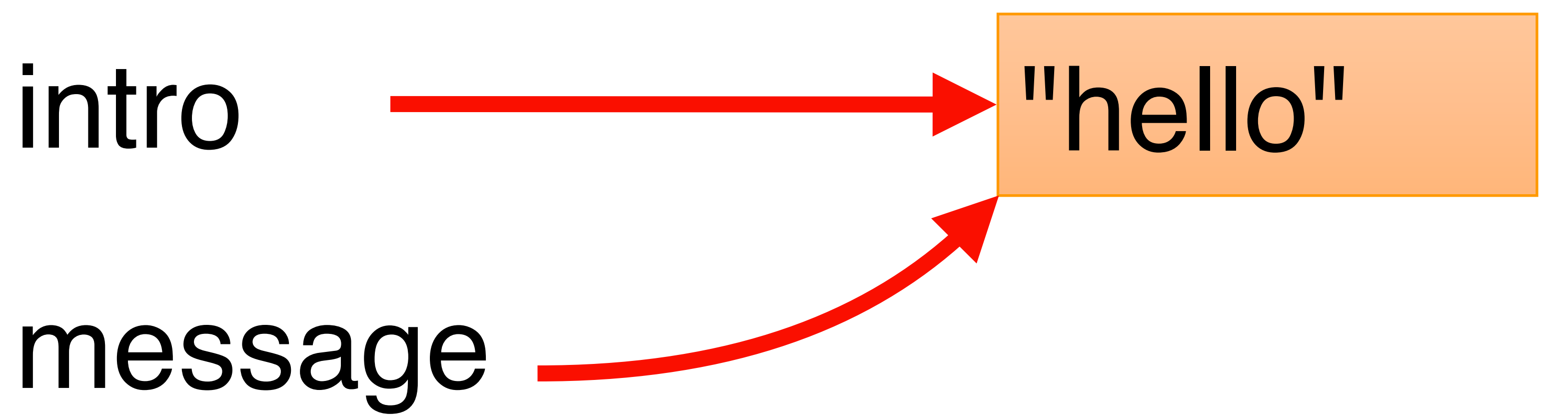
```
intro = "hello"
message = intro
message.append(", world")
```

Spooky action at a distance

```
intro = "hello"
message = intro
message.append(", world")
```

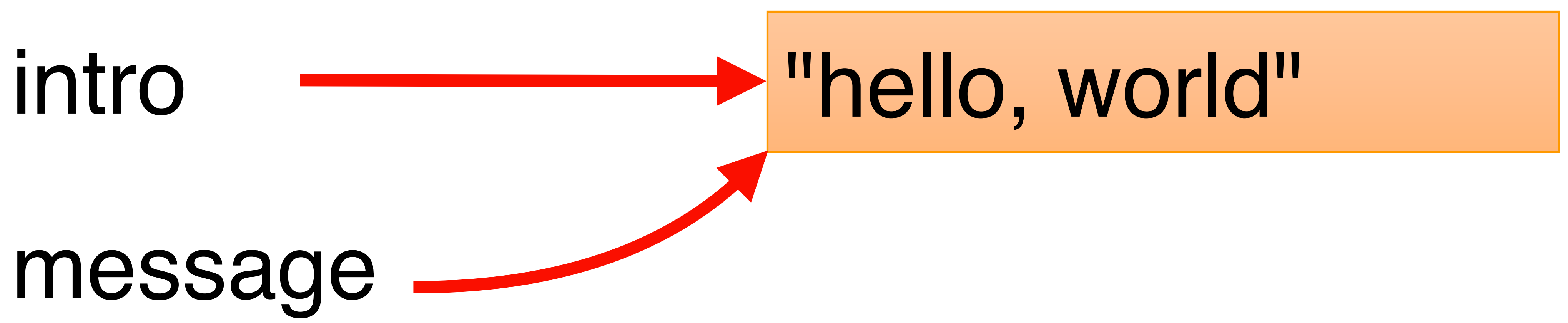
Spooky action at a distance

```
intro = "hello"
message = intro
message.append(", world")
```



Spooky action at a distance

```
intro = "hello"
message = intro
message.append(", world")
```



Immutable string is less spooky

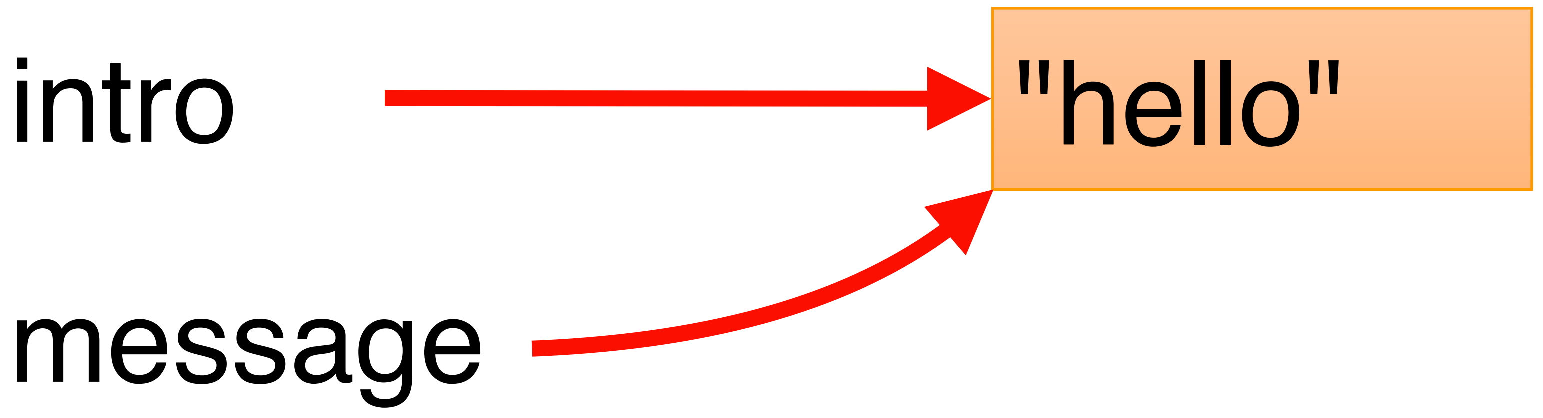
```
intro = "hello"
message = intro
message = message.concat(", world")
```

Immutable string is less spooky

```
intro = "hello"
message = intro
message = message.concat(", world")
```

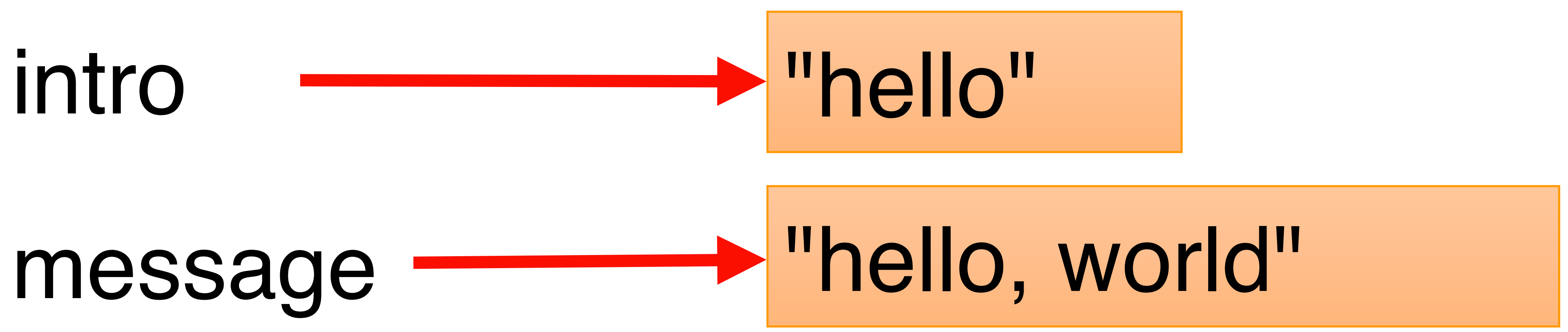
Immutable string is less spooky

```
intro = "hello"
message = intro
message = message.concat(", world")
```



Immutable string is less spooky

```
intro = "hello"
message = intro
message = message.concat(", world")
```



Philosophy of value semantics

Maybe the problem isn't mutation by itself, but mutation of shared state?

A deeper idea than I thought.

Philosophy of value semantics

Maybe the problem isn't mutation by itself, but mutation of shared state?

A deeper idea than I thought.

Problems with the immutable string scheme:

- Creating a new string buffer for every mutation step is expensive — $O(N^2)$.
- Add StringBuilder to manage mutating string buffers in place.
- StringBuilder ends up duplicating string's non-mutating API.
- One more type for the user to learn.

Dave's declaration

"Over my dead body will Swift have a StringBuilder!"

No rule of five

Array

Was a hack for testing the compiler, but not a design.

Had reference semantics!

Copy/assignment operators not implemented… yet?

Array

Was a hack for testing the compiler, but not a design.

Had reference semantics!

Copy/assignment operators not implemented… yet?

Time to bother Doug

Doug Gregor's declaration

"No rule-of-five programming for you!"

Doug Gregor's declaration

"No rule-of-five programming for you!"

🤷🏻‍♂️

Doug Gregor's declaration

“No rule-of-five programming for you!”

🤷‍♂️

“Use copy-on-write.”

Doug Gregor's declaration

"No rule-of-five programming for you!"

🤷🏻‍♂️

"Use copy-on-write."

"Go away and don't bother me; I need to code."

https://gist.github.com/alf-p-steinbach/c53794c3711eb74e7558bb514204e755

alf-p-steinbach / **strings 1 - Why COW is ungood for std string\strings 1 - Why COW is ungood for std string .md**

Created 3 years ago • Report abuse

Unsubscribe Star 7

Code Revisions 1 Stars 7 Forks 1 Embed <script src="https://

Why COW was deemed ungood for std::string

strings 1 – Why COW is ungood for std string\strings 1 – Why COW is ungood for std string .md Raw

Why COW was deemed ungood for std::string.

COW, short for *copy on write*, is a way to implement mutable strings so that creating strings and logically copying strings, is reduced to almost nothing; conceptually they become free operations like no-ops.

Basic idea: to share a data buffer among string instances, and only make a copy for a specific instance (the *copy on write*) when that instance's data is modified. The general cost of this is only an extra indirection for accessing the value of a string, so a COW

No rule of five | Consequences

All variable-sized value types use CoW

Copy and assignment never have to allocate memory

Copy and assignment is always O(1)

Copy and assignment can never fail

Optimizer was taught to remove redundant reference counting.

Optimizer was taught to hoist uniqueness checks

We fearlessly pass arrays, strings, and dictionaries by value

Chris Lattner's Observation

"C++ has value semantics, but nobody uses it."

Parameter passing

Say what you mean? | Sincere parameter passing

```
// Returns the sum of elements in `x` and calls `dump` on each one.
auto sumAndDump(std::vector<int> const x, void(*dump)(int)) -> int {
    ranges::for_each(x, dump);
    return std::accumulate(x.begin(), x.end(), 0);
}
```

Say what you mean? | Sincere parameter passing

```
// Returns the sum of elements in `x` and calls `dump` on each one.
auto sumAndDump(std::vector<int> const x, void(*dump)(int)) -> int {
    ranges::for_each(x, dump);
    return std::accumulate(x.begin(), x.end(), 0);
}

std::vector x = {0, 1, 2, 3};
void d(int) { x[e] += 1; }
int main() {
    std::cout << f(x, d); // Prints "6"
}
```

Say what you mean? | Facetious parameter passing

```
// Returns the sum of elements in `x` and calls `dump` on each one.
auto sumAndDump(std::vector<int> const& x, void(*dump)(int)) -> int {
    ranges::for_each(x, dump);
    return std::accumulate(x.begin(), x.end(), 0);
}

std::vector x = {0, 1, 2, 3};
void d(int) { x[e] += 1; }
int main() {
    std::cout << f(x, d); // Prints "10"
}
```

Say what you mean? | Facetious parameter passing

```
// Returns the sum of elements in `x` and calls `dump` on each one.
auto sumAndDump(std::vector<int> const& x, void(*dump)(int)) -> int {
    ranges::for_each(x, dump);
    return std::accumulate(x.begin(), x.end(), 0);
}

std::vector x = {0, 1, 2, 3};
void d(int) { x[e] += 1; }
int main() {
    std::cout << f(x, d); // Prints "10", not "6"
}
```

Mutation

Facetious | Pass by reference

```
X y;

auto frob(X& y) -> void {
  foo();
  bar(y);

}

frob(y);
```

Sincere | Functional update

```
X y;

auto frob(X y) -> X {
  foo();
  bar(y);
  return y;
}

y = frob(y);
```

Mutation

Facetious | Pass by reference

```
X y;

auto frob(X& y) -> void {
  foo();
  bar(y);
}


frob(y);
```

Sincere | Functional update

```
X y;

auto frob(X y) -> X {
  foo();
  bar(y);
  return y;
}

y = frob(y);
```

Mutation in Swift

```
var y: X;

func frob(_ y: inout X) -> Void {
  foo();
  bar(&y);
}


frob(y);
```

Dave's last theorem

Swift's model of parameter passing can be extended to allow noncopyable types to be efficently passed "by value" or by move with minimal complexity, and the model could be applied to a future version of C++.


Adobe
Artwork by Elise Swopes / USA
Ps
Lr