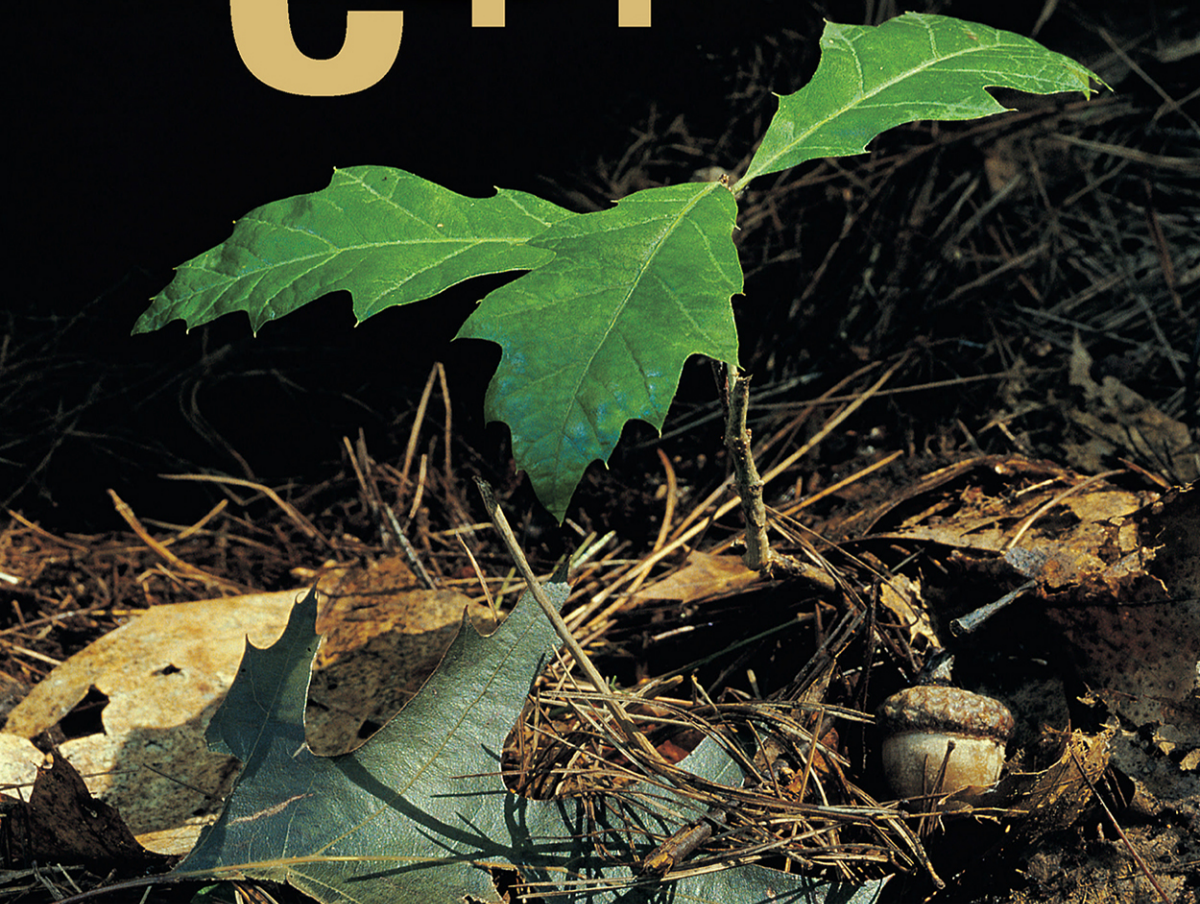




Winner! *Software Development Productivity Award*

BJARNE STROUSTRUP

The Design and Evolution of **C++**



FREE SAMPLE CHAPTER

SHARE WITH OTHERS



The Design and Evolution of C++

Bjarne Stroustrup

AT&T Bell Laboratories
Murray Hill, New Jersey



ADDISON-WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sidney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and we were aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers discounts on this book when ordered in quantity for special sales. For more information, please contact:

Pearson Education Corporate Sales Division
201 W. 103rd Street
Indianapolis, IN 46290
(800) 428-5331
corpsales@pearsoned.com

Visit AW on the Web: www.awl.com/cseng/

Library of Congress Cataloging-in-Publication Data

Stroustrup, Bjarne.

The design and evolution of C++ / Bjarne Stroustrup.

p. cm.

Includes bibliographical references and index.

ISBN 0-201-54330-3

1. C++ (Computer program language) I. Title. II. Title: Design and evolution of C plus plus.

QA76.73.C153S79 1994

005.13'3—dc20

93-50758

CIP



AT&T

Copyright © 1994 by AT&T Bell Labs.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

This book was typeset in Times Roman and Courier by the author.

Text printed on recycled and acid-free paper.

ISBN 0201543303

Text printed in the United States at Offset Paperback Manufacturers in Laflin, Pennsylvania.

Preface

*He who does not plow,
must write.*

– Martin A. Hansen

The ACM HOPL-2 conference on the History of Programming Languages asked me to write a paper on the history of C++. This seemed a reasonable idea and a bit of an honor, so I started writing. To get a more comprehensive and balanced view of C++'s growth, I asked a few friends from the early days of C++ for their recollections. That caused news of this project to travel through the grapevine. There, the story mutated, and one day I received a message from a friend asking where he could buy my new book on the design of C++. That email message is the real origin of this book.

Traditional books about programming and programming languages explain *what* a language is and how to use it. However, many people are also curious about *why* a language is the way it is and how it came to be that way. This book answers these last two questions for C++. It explains how C++ evolved from its first design to the language in use today. It describes the key problems, design aims, language ideas, and constraints that shaped C++, and how they changed over time.

Naturally, C++ and the ideas about design and programming that shaped it didn't just mutate by themselves. What really evolved was the C++ users' understanding of their practical problems and of the tools needed to help solve them. Consequently, this book also traces the key problems tackled using C++ and the views of the people who tackled them in ways that influenced C++.

C++ is still a young language. Some of the issues discussed here are yet unknown to many users. Many implications of decisions described here will not become obvious for years to come. This book presents my view of how C++ came about, what it is, and what it ought to be. I hope this will be of help to people trying to understand how best to use C++ and in the continuing evolution of C++.

The emphasis is on the overall design goals, practical constraints, and people that shaped C++. The key design decisions relating to language features are discussed and put into their historical context. The evolution of C++ is traced from C with Classes through Release 1.0 and 2.0 to the current ANSI/ISO standards work and the explosion of use, interest, commercial activity, compilers, tools, environments, and libraries. C++'s relationship to C and Simula is discussed in detail. C++'s relationship to other languages is discussed briefly. The design of major language facilities such as classes, inheritance, abstract classes, overloading, memory management, templates, exception handling, run-time type information, and namespaces are discussed in some detail.

The primary aim of this book is to give C++ programmers a better idea of the background and fundamental concepts of their language and hopefully to inspire them to experiment with ways of using C++ that are new to them. This book can also be read by experienced programmers and students of programming languages and might help them decide whether using C++ might be worth their while.

Acknowledgments

I am very grateful to Steve Clamage, Tony Hansen, Lorraine Juhl, Peter Juhl, Brian Kernighan, Lee Knight, Doug Lea, Doug McIlroy, Barbara Moo, Jens Palsberg, Steve Rumsby, and Christopher Skelly for reading complete drafts of this book. Their constructive comments caused major changes to the contents and organization of this book. Steve Buroff, Martin Carroll, Sean Corfield, Tom Hagelskjær, Rick Hollinbeck, Dennis Mancl, and Stan Lippman helped by commenting on selected chapters. Also, thanks to Archie Lachner for asking for this book before I had thought of writing it.

Naturally, I owe thanks to the many people who helped make C++. In a sense, this book is a tribute to them and some of their names can be found throughout the chapters and in the index. Should I single out individuals, it must be Brian Kernighan, Andrew Koenig, Doug McIlroy, and Jonathan Shapiro, each of whom has been a steady source of help, encouragement, and ideas for more than a decade. Also, thanks to Kristen Nygaard and Dennis Ritchie as the designers of Simula and C from which the key ingredients of C++ were borrowed. Over the years, I have come to appreciate them not only as brilliant and practical language designers, but also as gentlemen and thoroughly likable individuals.

Murray Hill, New Jersey

Bjarne Stroustrup

Contents

Preface	iii
Acknowledgments	iv
Contents	v
Notes to the Reader	1
Introduction	1
How to Read this Book	2
C++ Timeline	4
Focus on Use and Users	4
Programming Languages	5
References	7
Part I:	17
The Prehistory of C++	19
1.1 Simula and Distributed Systems	19
1.2 C and Systems Programming	22
1.3 General Background	23

C with Classes	27
2.1 The Birth of C with Classes	27
2.2 Feature overview	29
2.3 Classes	30
2.4 Run-Time Efficiency	32
2.5 The Linkage Model	34
2.6 Static Type Checking	40
2.7 Why C?	43
2.8 Syntax Problems	45
2.9 Derived Classes	49
2.10 The Protection Model	53
2.11 Run-Time Guarantees	56
2.12 Minor Features	58
2.13 Features Considered, but not Provided	59
2.14 Work Environment	60
 The Birth of C++	 63
3.1 From C with Classes to C++	63
3.2 Aims	65
3.3 Cfront	66
3.4 Language Features	71
3.5 Virtual Functions	72
3.6 Overloading	78
3.7 References	85
3.8 Constants	89
3.9 Memory Management	91
3.10 Type Checking	92
3.11 Minor Features	93
3.12 Relationship to Classic C	100
3.13 Tools for Language Design	103
3.14 The C++ Programming Language (1st edition)	105
3.15 The Whatis? Paper	106
 C++ Language Design Rules	 109
4.1 Rules and Principles	109
4.2 General Rules	110
4.3 Design Support Rules	114
4.4 Language-Technical Rules	117
4.5 Low-Level Programming Support Rules	120
4.6 A Final Word	122

Chronology 1985-1993	123
5.1 Introduction	123
5.2 Release 2.0	124
5.3 The Annotated Reference Manual	126
5.4 ANSI and ISO Standardization	128
Standardization	133
6.1 What is a Standard?	133
6.2 How does the Committee Operate?	136
6.3 Clarifications	138
6.4 Extensions	147
6.5 Examples of Proposed Extensions	153
Interest and Use	163
7.1 The Explosion in Interest and Use	163
7.2 Teaching and Learning C++	168
7.3 Users and Applications	173
7.4 Commercial Competition	175
Libraries	181
8.1 Introduction	181
8.2 C++ Library Design	182
8.3 Early Libraries	184
8.4 Other Libraries	191
8.5 A Standard Library	194
Looking Ahead	195
9.1 Introduction	195
9.2 Retrospective	195
9.3 Only a Bridge?	200
9.4 What Will Make C++ Much More Effective?	205

Part II:	209
Memory Management	211
10.1 Introduction	211
10.2 Separating Allocation and Initialization	212
10.3 Array Allocation	213
10.4 Placement	214
10.5 Deallocation Problems	216
10.6 Memory Exhaustion	218
10.7 Automatic Garbage Collection	219
Overloading	223
11.1 Introduction	223
11.2 Overload Resolution	224
11.3 Type-Safe Linkage	232
11.4 Object Creation and Copying	237
11.5 Notational Convenience	241
11.6 Adding Operators to C++	247
11.7 Enumerations	253
Multiple Inheritance	257
12.1 Introduction	257
12.2 Ordinary Base Classes	258
12.3 Virtual Base Classes	259
12.4 The Object Layout Model	264
12.5 Method Combination	268
12.6 The Multiple Inheritance Controversy	269
12.7 Delegation	272
12.8 Renaming	273
12.9 Base and Member Initializers	275
Class Concept Refinements	277
13.1 Introduction	277
13.2 Abstract Classes	277
13.3 const Member Functions	284
13.4 Static Member Functions	288
13.5 Nested Classes	289
13.6 Inherited::	290

13.7 Relaxation of Overriding Rules	293
13.8 Multi-methods	297
13.9 Protected Members	301
13.10 Improved Code Generation	302
13.11 Pointers to Members	303
Casting	305
14.1 Major Extensions	305
14.2 Run-Time Type Information	306
14.3 A New Cast Notation	327
Templates	337
15.1 Introduction	337
15.2 Templates	338
15.3 Class Templates	341
15.4 Constraints on Template Arguments	343
15.5 Avoiding Code Replication	346
15.6 Function Templates	348
15.7 Syntax	355
15.8 Composition Techniques	356
15.9 Template Class Relationships	360
15.10 Template Instantiation	365
15.11 Implications of Templates	378
Exception Handling	383
16.1 Introduction	383
16.2 Aims and Assumptions	384
16.3 Syntax	385
16.4 Grouping	386
16.5 Resource Management	388
16.6 Resumption vs. Termination	390
16.7 Asynchronous Events	393
16.8 Multi-level Propagation	394
16.9 Static Checking	395
16.10 Invariants	397
Namespaces	399
17.1 Introduction	399

17.2 The Problem	400
17.3 Ideals for a Solution	402
17.4 The Solution: Namespaces	404
17.5 Implications for Classes	417
17.6 C Compatibility	420
 The C Preprocessor	 423
18.1 Cpp	423
 Index	 427

Notes to the Reader

*Writing is the only art
that must be learned by wrote.*
— anon

Main themes of this book — how to read this book — a timeline for C++
— C++ and other programming languages — references.

Introduction

C++ was designed to provide Simula's facilities for program organization together with C's efficiency and flexibility for systems programming. It was intended to deliver that to real projects within half a year of the idea. It succeeded.

At the time, mid-1979, neither the modesty nor the preposterousness of that goal was realized. The goal was modest in that it did not involve innovation, and preposterous in both its time scale and its Draconian demands on efficiency and flexibility. While a modest amount of innovation did emerge over the years, efficiency and flexibility have been maintained without compromise. While the goals for C++ have been refined, elaborated, and made more explicit over the years, C++ as used today directly reflects its original aims.

The purpose of this book is to document those aims, track their evolution, and present C++ as it emerged from the efforts of many people to create a language that served its users according to those aims. In doing so, I try to balance historical facts (such as names, places, and events) against technical issues of language design, implementation, and use. It is not my aim to document every little event, but to focus on the key events, ideas, and trends that actually influenced the definition of C++ or might influence its further evolution and use.

Wherever events are presented, I try to describe them as they happened rather than how I or others might have liked them to have happened. Where reasonable, I use

quotes from papers to illustrate the aims, principles, and features as they appeared at the time. I try not to project hindsight into events; rather, retrospective comments and comments about the implications of a decision are presented separately and are explicitly marked as retrospective. Basically, I abhor revisionist history and try to avoid it. For example, I mention that “I had found Pascal’s type system worse than useless – a straitjacket that caused more problems than it solved by forcing me to warp my designs to suit an implementation-oriented artifact.” That I thought that at the time is a fact, and it is a fact that had important implications for the evolution of C++. Whether that harsh judgement on Pascal was fair and whether I would make the same judgement today (more than a decade later) is irrelevant. I could not delete the fact (say, to spare the feelings of Pascal fans or to spare myself embarrassment or controversy) or modify it (by providing a more complete and balanced view) without warping the history of C++.

I try to mention people who contributed to the design and evolution of C++, and I try to be specific about their contribution and about when it occurred. This is somewhat hazardous. Since I don’t have a perfect memory, I will overlook some contributions. I offer my apologies. I name the people who caused a decision to be made for C++. Inevitably, these will not always be the people who first encountered a particular problem or who first thought of a solution. This can be unfortunate, but to be vague or to refrain from mentioning names would be worse. Feel free to send me information that might help clarify such points.

Where I describe historical events, there is a question of how objective my descriptions are. I have tried to compensate for unavoidable bias by obtaining information about events I wasn’t part of, by talking to other people involved in events, and by having several of the people involved in the evolution of C++ read this book. Their names can be found at the end of the preface. In addition, the History of Programming Languages (HOPL-2) paper [Stroustrup,1993] that contains the central historical facts from this book was extensively reviewed and deemed free of unsuitable bias.

How to Read this Book

Part I goes through the design, evolution, use, and standardization of C++ in roughly chronological order. I chose this organization because during the early years, major design decisions map onto the timeline as a neat, logical sequence. Chapters 1, 2, and 3 describe the origins of C++ and its evolution through C with Classes to Release 1.0. Chapter 4 describes the rules that guided C++’s growth during that period and beyond. Chapter 5 provides a chronology of post-1.0 developments, and Chapter 6 describes the ANSI/ISO C++ standards effort. To provide perspective, Chapters 7 and 8 discuss applications, tools, and libraries. Finally, Chapter 9 presents a retrospective and some thoughts on the future.

Part II presents the post-Release-1.0 development of C++. The language grew within a framework laid down around the time of Release 1.0. This framework included a set of desired features, such as templates and exception handling, and rules guiding their design. After Release 1.0, chronology didn’t matter much to the

development of C++. The current definition of C++ would have been substantially the same had the chronological sequence of post-1.0 extensions been different. The actual sequence in which the problems were solved and features provided is therefore of historical interest only. A strictly chronological presentation would interfere with the logical flow of ideas, so Part II is organized around major language features instead. Part II chapters are independent, so they can be read in any order: Chapter 10, memory management; Chapter 11, overloading; Chapter 12, multiple inheritance; Chapter 13, class concept refinements; Chapter 14, casting; Chapter 15, templates; Chapter 16, exception handling; Chapter 17, namespaces; Chapter 18, the C preprocessor.

Different people expect radically different things from a book on the design and evolution of a programming language. In particular, no two people seem to agree on what level of detail is appropriate for a discussion of this topic. *Every* review I received on the various versions of the HOPL-2 paper (well over a dozen reviews) was of the form, “This paper is too long ... please add information on topics X, Y, and Z.” Worse, about a third of the reviews had comments of the form, “Cut the philosophical/religious nonsense and give us proper technical details instead.” Another third commented, “Spare me the boring details and add information on your design philosophy.”

To wiggle out of this dilemma, I have written a book within a book. If you are not interested in details, then at first skip all subsections (numbered §x.y.z, where *x* is the chapter number and *y* is the section number). Later, read whatever else looks interesting. You can also read this book sequentially starting at page one and carry on until the end. Doing that, you might get bogged down in details. This is not meant to imply that details are unimportant. On the contrary, no programming language can be understood by considering principles and generalizations only; concrete examples are essential. However, looking at the details without an overall picture to fit them into is a way of getting seriously lost.

As an additional help, I have concentrated most of the discussion of new features and features generally considered advanced in Part II. This allows Part I to concentrate on basics. Almost all of the information on nontechnical aspects of C++’s evolution is found in Part I. People with little patience for “philosophy” can break up the discussion in Chapters 4 through 9 by looking ahead to the technical details of language features in Part II.

I assume that some will use this book as a reference and that many will read individual chapters without bothering with all preceding chapters. To make such use feasible, I have made the individual chapters relatively self-contained for the experienced C++ programmer and been liberal with cross references and index terms.

Please note that I don’t try to define the features of C++ here, I present only as much detail as is necessary to provide a self-contained description of how the features came about. I don’t try to teach C++ programming or design either; for a tutorial, see [2nd].

C++ Timeline

This C++ timeline might help you keep track of where the story is taking you:

1979	May	Work on C with Classes starts
	Oct	1st C with Classes implementation in use
1980	Apr	1st internal Bell Labs paper on C with Classes [Stroustrup,1980]
1982	Jan	1st external paper on C with Classes [Stroustrup,1982]
1983	Aug	1st C++ implementation in use
	Dec	C++ named
1984	Jan	1st C++ manual
1985	Feb	1st external C++ release (Release E)
	Oct	Cfront Release 1.0 (first commercial release)
	Oct	<i>The C++ Programming Language</i> [Stroustrup,1986]
1986	Aug	The “whatis paper” [Stroustrup,1986b]
	Sep	1st OOPSLA conference (start of OO hype centered on Smalltalk)
	Nov	1st commercial Cfront PC port (Cfront 1.1, Glockenspiel)
1987	Feb	Cfront Release 1.2
	Nov	1st USENIX C++ conference (Santa Fe, NM)
	Dec	1st GNU C++ release (1.13)
1988	Jan	1st Oregon Software C++ release
	June	1st Zortech C++ release
	Oct	1st USENIX C++ implementers workshop (Estes Park, CO)
1989	June	Cfront Release 2.0
	Dec	ANSI X3J16 organizational meeting (Washington, DC)
1990	May	1st Borland C++ release
	Mar	1st ANSI X3J16 technical meeting (Somerset, NJ)
	May	<i>The Annotated C++ Reference Manual</i> [ARM]
	July	Templates accepted (Seattle, WA)
	Nov	Exceptions accepted (Palo Alto, CA)
1991	June	<i>The C++ Programming Language (second edition)</i> [2nd]
	June	1st ISO WG21 meeting (Lund, Sweden)
	Oct	Cfront Release 3.0 (including templates)
1992	Feb	1st DEC C++ release (including templates and exceptions)
	Mar	1st Microsoft C++ release
	May	1st IBM C++ release (including templates and exceptions)
1993	Mar	Run-time type identification accepted (Portland, OR)
	July	Namespaces accepted (Munich, Germany)
1994	Aug	ANSI/ISO Committee Draft registered

Focus on Use and Users

This book is written for C++ users, that is, for programmers and designers. I have tried (believe it or not) to avoid truly obscure and esoteric topics to give a user’s view of the C++ language, its facilities, and its evolution. Purely language-technical

discussions are presented only if they shed light on issues that directly impact users. The discussions of name lookup in templates (§15.10) and of lifetime of temporaries (§6.3.2) are examples.

Programming language specialists, language lawyers, and implementers will find many tidbits in this book, but the aim is to present the large picture rather than to be precise and comprehensive about every little detail. If precise language-technical details is what you want the definition of C++ can be found in *The Annotated C++ Reference Manual* (the ARM) [ARM], in *The C++ Programming Language (second edition)* [2nd], and in the ANSI/ISO standards committee's working paper. However, the details of a language definition are incomprehensible without an understanding of the purpose of the language. The language, details and all, exists to help build programs. My intent with this book is to provide insights that can help in this endeavor.

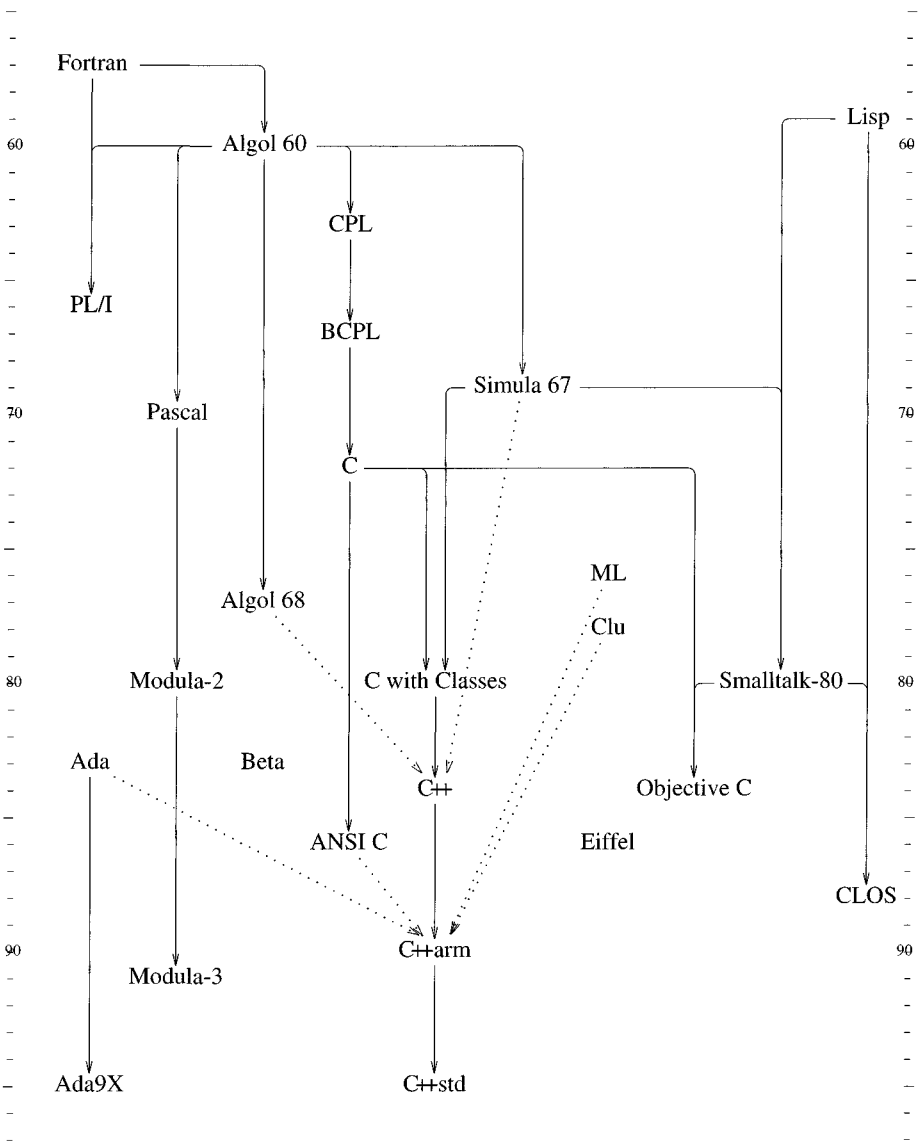
Programming Languages

Several reviewers asked me to compare C++ to other languages. This I have decided against doing. Thereby, I have reaffirmed a long-standing and strongly held view: Language comparisons are rarely meaningful and even less often fair. A good comparison of major programming languages requires more effort than most people are willing to spend, experience in a wide range of application areas, a rigid maintenance of a detached and impartial point of view, and a sense of fairness. I do not have the time, and as the designer of C++, my impartiality would never be fully credible.

I also worry about a phenomenon I have repeatedly observed in honest attempts at language comparisons. The authors try hard to be impartial, but are hopelessly biased by focusing on a single application, a single style of programming, or a single culture among programmers. Worse, when one language is significantly better known than others, a subtle shift in perspective occurs: Flaws in the well-known language are deemed minor and simple workarounds are presented, whereas similar flaws in other languages are deemed fundamental. Often, the workarounds commonly used in the less-well-known languages are simply unknown to the people doing the comparison or deemed unsatisfactory because they would be unworkable in the more familiar language.

Similarly, information about the well-known language tends to be completely up-to-date, whereas for the less-known language, the authors rely on several-year-old information. For languages that are worth comparing, a comparison of language X as defined three years ago vs. language Y as it appears in the latest experimental implementation is neither fair nor informative. Thus, I restrict my comments about languages other than C++ to generalities and to very specific comments. This is a book about C++, its design, and the factors that shaped its evolution. It is not an attempt to contrast C++ language features with those found in other languages.

To fit C++ into a historical context, here is a chart of the first appearances of languages that often crop up in discussions about C++:



The chart is not intended to be anywhere near complete except for significant influences on C++. In particular, the chart understates the influence of the Simula class concept; Ada [Ichbiah,1979] and Clu [Liskov,1979] are weakly influenced by Simula [Birtwistle,1979]; Ada9X [Taft,1992], Beta [Madsen,1993], Eiffel [Meyer,1988], and Modula-3 [Nelson,1991] are strongly influenced. C++'s influence on other languages is left unrepresented. Solid lines indicate an influence on the structure of the

language; dotted lines indicate an influence on specific features. Adding lines to show this for every language would make the diagram too messy to be useful. The dates for the languages are generally those of the first usable implementation. For example, Algol68 [Woodward,1974] can be found by the year 1977 rather than 1968.

One conclusion I drew from the wildly divergent comments on the HOPL-2 paper – and from many other sources – is that there is no agreement on what a programming language really is and what its main purpose is supposed to be. Is a programming language a tool for instructing machines? A means of communicating between programmers? A vehicle for expressing high-level designs? A notation for algorithms? A way of expressing relationships between concepts? A tool for experimentation? A means of controlling computerized devices? My view is that a general-purpose programming language must be all of those to serve its diverse set of users. The only thing a language cannot be – and survive – is a mere collection of “neat” features.

The difference in opinions reflects differing views of what computer science is and how languages ought to be designed. Ought computer science be a branch of mathematics? Of engineering? Of architecture? Of art? Of biology? Of sociology? Of philosophy? Alternatively, does it borrow techniques and approaches from all of these disciplines? I think so.

This implies that language design parts ways from the “purer” and more abstract disciplines such as mathematics and philosophy. To serve its users, a general-purpose programming language must be eclectic and take many practical and sociological factors into account. In particular, every language is designed to solve a particular set of problems at a particular time according to the understanding of a particular group of people. From this initial design, it grows to meet new demands and reflects new understandings of problems and of tools and techniques for solving them. This view is pragmatic, yet not unprincipled. It is my firm belief that *all* successful languages are grown and not merely designed from first principles. Principles underlie the first design and guide the further evolution of the language. However, even principles evolve.

References

This section contains the references from every chapter of this book.

- [2nd] see [Stroustrup,1991].
- [Agha,1986] Gul Agha: *An Overview of Actor languages*. ACM SIGPLAN Notices. October 1986.
- [Aho,1986] Alfred Aho, Ravi Sethi, and Jeffrey D. Ullman: *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA. 1986. ISBN 0-201-10088-6.
- [ARM] see [Ellis,1990].
- [Babcisky,1984] Karel Babcisky: *Simula Performance Assessment*. Proc. IFIP WG2.4 Conference on System Implementation Languages: Experience and Assessment. Canterbury, Kent, UK. September 1984.
- [Barton,1994] John J. Barton and Lee R. Nackman: *Scientific and*

- Engineering C++: An Introduction with Advanced Techniques and Examples.* Addison-Wesley, Reading, MA. 1994. ISBN 0-201-53393-6.
- [Birtwistle,1979] Graham Birtwistle, Ole-Johan Dahl, Björn Myrhaug, and Kristen Nygaard: *SIMULA BEGIN*. Studentlitteratur, Lund, Sweden. 1979. ISBN 91-44-06212-5.
- [Boehm,1993] Hans-J. Boehm: *Space Efficient Conservative Garbage Collection*. Proc. ACM SIGPLAN '93 Conference on Programming Language Design and Implementation. ACM SIGPLAN Notices. June 1993.
- [Booch,1990] Grady Booch and Michael M. Vilot: *The Design of the C++ Booch Components*. Proc. OOPSLA'90. October 1990.
- [Booch,1991] Grady Booch: *Object-Oriented Design*. Benjamin Cummings, Redwood City, CA. 1991. ISBN 0-8053-0091-0.
- [Booch,1993] Grady Booch: *Object-oriented Analysis and Design with Applications, 2nd edition*. Benjamin Cummings, Redwood City, CA. 1993. ISBN 0-8053-5340-2.
- [Booch,1993b] Grady Booch and Michael M. Vilot: *Simplifying the C++ Booch Components*. The C++ Report. June 1993.
- [Budge,1992] Ken Budge, J.S. Perry, and A.C. Robinson: *High-Performance Scientific Computation using C++*. Proc. USENIX C++ Conference. Portland, OR. August 1992.
- [Buhr,1992] Peter A. Buhr and Glen Ditchfield: *Adding Concurrency to a Programming Language*. Proc. USENIX C++ Conference. Portland, OR. August 1992.
- [Call,1987] Lisa A. Call, et al.: *CLAM – An Open System for Graphical User Interfaces*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.
- [Cameron,1992] Don Cameron, et al.: *A Portable Implementation of C++ Exception Handling*. Proc. USENIX C++ Conference. Portland, OR. August 1992.
- [Campbell,1987] Roy Campbell, et al.: *The Design of a Multiprocessor Operating System*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.
- [Cattell,1991] Rich G.G. Cattell: *Object Data Management: Object-Oriented and Extended Relational Database Systems*. Addison-Wesley, Reading, MA. 1991. ISBN 0-201-53092-9.
- [Cargill,1991] Tom A. Cargill: *The Case Against Multiple Inheritance in C++*. USENIX Computer Systems. Vol 4, no 1, 1991.
- [Carroll,1991] Martin Carroll: *Using Multiple Inheritance to Implement Abstract Data Types*. The C++ Report. April 1991.
- [Carroll,1993] Martin Carroll: *Design of the USL Standard Components*. The C++ Report. June 1993.
- [Chandy,1993] K. Mani Chandy and Carl Kesselman: *Compositional C++*:

- [Cristian,1989] *Compositional Parallel Programming*. Proc. Fourth Workshop on Parallel Computing and Compilers. Springer-Verlag. 1993.
- [Cox,1986] Flaviu Cristian: *Exception Handling*. Dependability of Resilient Computers, T. Andersen, editor. BSP Professional Books, Blackwell Scientific Publications, 1989.
- [Dahl,1988] Brad Cox: *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley, Reading, MA. 1986.
- [Dahl,1988] Ole-Johan Dahl: Personal communication.
- [Dearle,1990] Fergal Dearle: *Designing Portable Applications Frameworks for C++*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.
- [Dorward,1990] Sean M. Dorward, et al.: *Adding New Code to a Running Program*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.
- [Eick,1991] Stephen G. Eick: *SIMLIB - An Object-Oriented C++ Library for Interactive Simulation of Circuit-Switched Networks*. Proc. Simulation Technology Conference. Orlando, FL. October 1991.
- [Ellis,1990] Margaret A. Ellis and Bjarne Stroustrup: *The Annotated C++ Reference Manual*. Addison-Wesley, Reading, MA. 1990. ISBN 0-201-51459-1.
- [Faust,1990] John E. Faust and Henry M. Levy: *The Performance of an Object-Oriented Threads Package*. Proc. ACM joint ECOOP and OOPSLA Conference. Ottawa, Canada. October 1990.
- [Fontana,1991] Mary Fontana and Martin Neath: *Checked Out and Long Overdue: Experiences in the Design of a C++ Class Library*. Proc. USENIX C++ Conference. Washington, DC. April 1991.
- [Forslund,1990] David W. Forslund, et al.: *Experiences in Writing Distributed Particle Simulation Code in C++*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.
- [Gautron,1992] Philippe Gautron: *An Assertion Mechanism based on Exceptions*. Proc. USENIX C++ Conference. Portland, OR. August 1992.
- [Gehani,1988] Narain H. Gehani and William D. Roome: *Concurrent C++: Concurrent Programming With Class(es)*. Software—Practice & Experience. Vol 18, no 12, 1988.
- [Goldberg,1983] Adele Goldberg and David Robson: *Smalltalk-80, The Language and its Implementation*. Addison-Wesley, Reading, MA. 1983. ISBN 0-201-11371-6.
- [Goodenough,1975] John Goodenough: *Exception Handling: Issues and a Proposed Notation*. Communications of the ACM. December 1975.
- [Gorlen,1987] Keith E. Gorlen: *An Object-Oriented Class Library for C++ Programs*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

- [Gorlen,1990] Keith E. Gorlen, Sanford M. Orlow, and Perry S. Plexico: *Data Abstraction and Object-Oriented Programming in C++*. Wiley. West Sussex. England. 1990. ISBN 0-471-92346-X.
- [Hübel,1992] Peter Hübel and J.T. Thorsen: *An Implementation of a Persistent Store for C++*. Computer Science Department. Aarhus University, Denmark. December 1992.
- [Ichbiah,1979] Jean D. Ichbiah, et al.: *Rationale for the Design of the ADA Programming Language*. SIGPLAN Notices Vol 14, no 6, June 1979 Part B.
- [Ingalls,1986] Daniel H.H. Ingalls: *A Simple Technique for Handling Multiple Polymorphism*. Proc. ACM OOPSLA Conference. Portland, OR. November 1986.
- [Interrante,1990] John A. Interrante and Mark A. Linton: *Runtime Access to Type Information*. Proc. USENIX C++ Conference. San Francisco 1990.
- [Johnson,1992] Steve C. Johnson: Personal communication.
- [Johnson,1989] Ralph E. Johnson: *The Importance of Being Abstract*. The C++ Report. March 1989.
- [Keffer,1992] Thomas Keffer: *Why C++ Will Replace Fortran*. C++ Supplement to Dr. Dobbs Journal. December 1992.
- [Keffer,1993] Thomas Keffer: *The Design and Architecture of Tools.h++*. The C++ Report. June 1993.
- [Kernighan,1976] Brian Kernighan and P.J. Plauger: *Software Tools*. Addison-Wesley, Reading, MA. 1976. ISBN 0-201-03669.
- [Kernighan,1978] Brian Kernighan and Dennis Ritchie: *The C Programming Language*. Prentice-Hall, Englewood Cliffs, NJ. 1978. ISBN 0-13-110163-3.
- [Kernighan,1981] Brian Kernighan: *Why Pascal is not my Favorite Programming Language*. AT&T Bell Labs Computer Science Technical Report No 100. July 1981.
- [Kernighan,1984] Brian Kernighan and Rob Pike: *The UNIX Programming Environment*. Prentice-Hall, Englewood Cliffs, NJ. 1984. ISBN 0-13-937699-2.
- [Kernighan,1988] Brian Kernighan and Dennis Ritchie: *The C Programming Language (second edition)*. Prentice-Hall, Englewood Cliffs, NJ. 1988. ISBN 0-13-110362-8.
- [Kiczales,1992] Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow: *The Art of the Metaobject Protocol*. The MIT Press. Cambridge, Massachusetts. 1991. ISBN 0-262-11158-6.
- [Koenig,1988] Andrew Koenig: *Associative arrays in C++*. Proc. USENIX Conference. San Francisco, CA. June 1988.
- [Koenig,1989] Andrew Koenig and Bjarne Stroustrup: *C++: As close to C as possible – but no closer*. The C++ Report. July 1989.
- [Koenig,1989b] Andrew Koenig and Bjarne Stroustrup: *Exception Handling for*

- C++. Proc. "C++ at Work" Conference. November 1989.
- [Koenig,1990] Andrew Koenig and Bjarne Stroustrup: *Exception Handling for C++ (revised)*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990. Also, Journal of Object-Oriented Programming. July 1990.
- [Koenig,1991] Andrew Koenig: *Applicators, Manipulators, and Function Objects*. C++ Journal, vol. 1, #1. Summer 1990.
- [Koenig,1992] Andrew Koenig: *Space Efficient Trees in C++*. Proc. USENIX C++ Conference. Portland, OR. August 1992.
- [Krogdahl,1984] Stein Krogdahl: *An Efficient Implementation of Simula Classes with Multiple Prefixing*. Research Report No 83. June 1984. University of Oslo, Institute of Informatics.
- [Lea,1990] Doug Lea and Marshall P. Cline: *The Behavior of C++ Classes*. Proc. ACM SOOPPA Conference. September 1990.
- [Lea,1991] Doug Lea: Personal Communication.
- [Lea,1993] Doug Lea: *The GNU C++ Library*. The C++ Report. June 1993.
- [Lenkov,1989] Dmitry Lenkov: *C++ Standardization Proposal*. #X3J11/89-016.
- [Lenkov,1991] Dmitry Lenkov, Michay Mehta, and Shankar Unni: *Type Identification in C++*. Proc. USENIX C++ Conference. Washington, DC. April 1991.
- [Linton,1987] Mark A. Linton and Paul R. Calder: *The Design and Implementation of InterViews*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.
- [Lippman,1988] Stan Lippman and Bjarne Stroustrup: *Pointers to Class Members in C++*. Proc. USENIX C++ Conference. Denver, CO. October 1988.
- [Liskov,1979] Barbara Liskov, et al.: *CLU Reference manual*. MIT/LCS/TR-225. October 1979.
- [Liskov,1987] Barbara Liskov: *Data Abstraction and Hierarchy*. Addendum to Proceedings of OOPSLA'87. October 1987.
- [Madsen,1993] Ole Lehrmann Madsen, et al.: *Object-Oriented Programming in the Beta Programming Language*. Addison-Wesley, Reading, MA. 1993. ISBN 0-201-62430.
- [McCluskey,1992] Glen McCluskey: *An Environment for Template Instantiation*. The C++ Report. February 1992.
- [Meyer,1988] Bertrand Meyer: *Object-Oriented Software Construction*. Prentice-Hall, Englewood Cliffs, NJ. 1988. ISBN 0-13-629049.
- [Miller,1988] William M. Miller: *Exception Handling without Language Extensions*. Proc. USENIX C++ Conference. Denver CO. October 1988.
- [Mitchell,1979] James G. Mitchell, et.al.: *Mesa Language Manual*. XEROX

- PARC, Palo Alto, CA. CSL-79-3. April 1979.
- [Murray,1992] Rob Murray: *A Statically Typed Abstract Representation for C++ Programs*. Proc. USENIX C++ Conference. Portland, OR. August 1992.
- [Nelson,1991] Nelson, G. (editor): *Systems Programming with Modula-3*. Prentice-Hall, Englewood Cliffs, NJ. 1991. ISBN 0-13-590464-1.
- [Rose,1984] Leonie V. Rose and Bjarne Stroustrup: *Complex Arithmetic in C++*. Internal AT&T Bell Labs Technical Memorandum. January 1984. Reprinted in AT&T C++ Translator Release Notes. November 1985.
- [Parrington,1990] Graham D. Parrington: *Reliable Distributed Programming in C++*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.
- [Reiser,1992] John F. Reiser: *Static Initializers: Reducing the Value-Added Tax on Programs*. Proc. USENIX C++ Conference. Portland, OR. August 1992.
- [Richards,1980] Martin Richards and Colin Whitby-Strevens: *BCPL – the language and its compiler*. Cambridge University Press, Cambridge, England. 1980. ISBN 0-521-21965-5.
- [Rovner,1986] Paul Rovner: *Extending Modula-2 to Build Large, Integrated Systems*. IEEE Software Vol 3, No 6, November 1986.
- [Russo,1988] Vincent F. Russo and Simon M. Kaplan: *A C++ Interpreter for Scheme*. Proc. USENIX C++ Conference. Denver, CO. October 1988.
- [Russo,1990] Vincent F. Russo, Peter W. Madany, and Roy H. Campbell: *C++ and Operating Systems Performance: A Case Study*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.
- [Sakkinen,1992] Markku Sakkinen: *A Critique of the Inheritance Principles of C++*. USENIX Computer Systems, vol 5, no 1, Winter 1992.
- [Sethi,1980] Ravi Sethi: *A case study in specifying the semantics of a programming language*. Seventh Annual ACM Symposium on Principles of Programming Languages. January 1980.
- [Sethi,1981] Ravi Sethi: *Uniform Syntax for Type Expressions and Declarators*. Software – Practice and Experience, Vol 11. 1981.
- [Sethi,1989] Ravi Sethi: *Programming Languages – Concepts and Constructs*. Addison-Wesley, Reading, MA. 1989. ISBN 0-201-10365-6.
- [Shopiro,1985] Jonathan E. Shopiro: *Strings and Lists for C++*. AT&T Bell Labs Internal Technical Memorandum. July 1985.
- [Shopiro,1987] Jonathan E. Shopiro: *Extending the C++ Task System for Real-Time Control*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

- [Shopiro,1989] Jonathan E. Shopiro: *An Example of Multiple Inheritance in C++: A Model of the Iostream Library*. ACM SIGPLAN Notices. December 1989.
- [Schwarz,1989] Jerry Schwarz: *Iostreams Examples*. AT&T C++ Translator Release Notes. June 1989.
- [Snyder,1986] Alan Snyder: *Encapsulation and Inheritance in Object-Oriented Programming Languages*. Proc. OOPSLA'86. September 1986.
- [Stal,1993] Michael Stal and Uwe Steinmüller: *Generic Dynamic Arrays*. The C++ Report. October 1993.
- [Stepanov,1993] Alexander Stepanov and David R. Musser: *Algorithm-Oriented Generic Software Library Development*. HP Laboratories Technical Report HPL-92-65. November 1993.
- [Stroustrup,1978] Bjarne Stroustrup: *On Unifying Module Interfaces*. ACM Operating Systems Review Vol 12 No 1. January 1978.
- [Stroustrup,1979] Bjarne Stroustrup: *Communication and Control in Distributed Computer Systems*. Ph.D. thesis, Cambridge University, 1979.
- [Stroustrup,1979b] Bjarne Stroustrup: *An Inter-Module Communication System for a Distributed Computer System*. Proc. 1st International Conf. on Distributed Computing Systems. October 1979.
- [Stroustrup,1980] Bjarne Stroustrup: *Classes: An Abstract Data Type Facility for the C Language*. Bell Laboratories Computer Science Technical Report CSTR-84. April 1980. Revised, August 1981. Revised yet again and published as [Stroustrup,1982].
- [Stroustrup,1980b] Bjarne Stroustrup: *A Set of C Classes for Co-routine Style Programming*. Bell Laboratories Computer Science Technical Report CSTR-90. November 1980.
- [Stroustrup,1981] Bjarne Stroustrup: *Long Return: A Technique for Improving The Efficiency of Inter-Module Communication*. Software Practice and Experience. January 1981.
- [Stroustrup,1981b] Bjarne Stroustrup: *Extensions of the C Language Type Concept*. Bell Labs Internal Memorandum. January 1981.
- [Stroustrup,1982] Bjarne Stroustrup: *Classes: An Abstract Data Type Facility for the C Language*. ACM SIGPLAN Notices. January 1982. Revised version of [Stroustrup,1980].
- [Stroustrup,1982b] Bjarne Stroustrup: *Adding Classes to C: An Exercise in Language Evolution*. Bell Laboratories Computer Science internal document. April 1982. Software: Practice & Experience, Vol 13. 1983.
- [Stroustrup,1984] Bjarne Stroustrup: *The C++ Reference Manual*. AT&T Bell Labs Computer Science Technical Report No 108. January 1984. Revised, November 1984.
- [Stroustrup,1984b] Bjarne Stroustrup: *Operator Overloading in C++*. Proc. IFIP WG2.4 Conference on System Implementation Languages:

- Experience & Assessment. September 1984.
- [Stroustrup,1984c] Bjarne Stroustrup: *Data Abstraction in C*. Bell Labs Technical Journal. Vol 63, No 8. October 1984.
- [Stroustrup,1985] Bjarne Stroustrup: *An Extensible I/O Facility for C++*. Proc. Summer 1985 USENIX Conference. June 1985.
- [Stroustrup,1986] Bjarne Stroustrup: *The C++ Programming Language*. Addison-Wesley, Reading, MA. 1986. ISBN 0-201-12078-X.
- [Stroustrup,1986b] Bjarne Stroustrup: *What is Object-Oriented Programming?* Proc. 14th ASU Conference. August 1986. Revised version in Proc. ECOOP'87, May 1987, Springer Verlag Lecture Notes in Computer Science Vol 276. Revised version in *IEEE Software Magazine*. May 1988.
- [Stroustrup,1986c] Bjarne Stroustrup: *An Overview of C++*. ACM SIGPLAN Notices. October 1986.
- [Stroustrup,1987] Bjarne Stroustrup: *Multiple Inheritance for C++*. Proc. EUUG Spring Conference, May 1987. Also, USENIX Computer Systems, Vol 2 No 4. Fall 1989.
- [Stroustrup,1987b] Bjarne Stroustrup and Jonathan Shopiro: *A Set of C classes for Co-Routine Style Programming*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.
- [Stroustrup,1987c] Bjarne Stroustrup: *The Evolution of C++: 1985-1987*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.
- [Stroustrup,1987d] Bjarne Stroustrup: *Possible Directions for C++*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.
- [Stroustrup,1988] Bjarne Stroustrup: *Type-safe Linkage for C++*. USENIX Computer Systems, Vol 1 No 4. Fall 1988.
- [Stroustrup,1988b] Bjarne Stroustrup: *Parameterized Types for C++*. Proc. USENIX C++ Conference, Denver, CO. October 1988. Also, USENIX Computer Systems, Vol 2 No 1. Winter 1989.
- [Stroustrup,1989] Bjarne Stroustrup: *Standardizing C++*. The C++ Report. Vol 1 No 1. January 1989.
- [Stroustrup,1989b] Bjarne Stroustrup: *The Evolution of C++: 1985-1989*. USENIX Computer Systems, Vol 2 No 3. Summer 1989. Revised version of [Stroustrup,1987c].
- [Stroustrup,1990] Bjarne Stroustrup: *On Language Wars*. Hotline on Object-Oriented Technology. Vol 1, No 3. January 1990.
- [Stroustrup,1990b] Bjarne Stroustrup: *Sixteen Ways to Stack a Cat*. The C++ Report. October 1990.
- [Stroustrup,1991] Bjarne Stroustrup: *The C++ Programming Language (2nd edition)*. Addison-Wesley, Reading, MA. 1991. ISBN 0-201-53992-6.
- [Stroustrup,1992] Bjarne Stroustrup and Dmitri Lenkov: *Run-Time Type Identification for C++*. The C++ Report. March 1992. Revised version: Proc. USENIX C++ Conference. Portland, OR. August

- 1992.
- [Stroustrup,1992b] Bjarne Stroustrup: *How to Write a C++ Language Extension Proposal*. The C++ Report. May 1992.
- [Stroustrup,1993] Bjarne Stroustrup: *The History of C++: 1979-1991*. Proc. ACM History of Programming Languages Conference (HOPL-2). April 1993. ACM SIGPLAN Notices. March 1993.
- [Taft,1992] S. Tucker Taft: *Ada 9X: A Technical Summary*. CACM. November 1992.
- [Tiemann,1987] Michael Tiemann: "Wrappers:" *Solving the RPC problem in GNU C++*. Proc. USENIX C++ Conference. Denver, CO. October 1988.
- [Tiemann,1990] Michael Tiemann: *An Exception Handling Implementation for C++*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.
- [Weinand,1988] Andre Weinand, et al.: *ET++ – An Object-Oriented Application Framework in C++*. Proc. OOPSLA'88. September 1988.
- [Wikström,1987] Åke Wikström: *Functional Programming in Standard ML*. Prentice-Hall, Englewood Cliffs, NJ. 1987. ISBN 0-13-331968-7.
- [Waldo,1991] Jim Waldo: *Controversy: The Case for Multiple Inheritance in C++*. USENIX Computer Systems, vol 4, no 2, Spring 1991.
- [Waldo,1993] Jim Waldo (editor): *The Evolution of C++*. A USENIX Association book. The MIT Press, Cambridge, MA. 1993. ISBN 0-262-73107-X.
- [Wilkes,1979] M.V. Wilkes and R.M. Needham: *The Cambridge CAP Computer and its Operating System*. North-Holland, New York. 1979. ISBN 0-444-00357-6.
- [Woodward,1974] P.M. Woodward and S.G. Bond: *Algol 68-R Users Guide*. Her Majesty's Stationery Office, London. 1974. ISBN 0-11-771600-6.

This page intentionally left blank

Part I

Part I describes the origins of C++ and its evolution through C with Classes to Release 1.0. It also describes the rules that guided C++'s growth during that period and beyond. A chronology of post-1.0 developments is provided, and the C++ standardization effort is described. To provide perspective, the use of C++ is discussed. Finally, a retrospective and some thoughts on the future is presented.

Chapters

- 1 The Prehistory of C++
- 2 C with Classes
- 3 The Birth of C++
- 4 C++ Language Design Rules
- 5 Chronology 1985-1993
- 6 Standardization
- 7 Interest and Use
- 8 Libraries
- 9 Looking Ahead

This page intentionally left blank

The Prehistory of C++

*In olden days,
when EVIL ruled!
– Kristen Nygaard*

Simula and distributed systems — C and systems programming — the influence of mathematics, history, philosophy, and literature.

1.1 Simula and Distributed Systems

The prehistory of C++ – the couple of years before the idea of adding Simula-like features to C occurred to me – is important because during this time, the criteria and ideals that later shaped C++ emerged. I was working on my Ph.D. Thesis in the Computing Laboratory of Cambridge University in England. My aim was to study alternatives for the organization of system software for distributed systems. The conceptual framework was provided by the capability-based Cambridge CAP computer and its experimental and continuously evolving operating system [Wilkes,1979]. The details of this work and its outcome [Stroustrup,1979] are of little relevance to C++. What is relevant, though, was the focus on composing software out of well-delimited modules and that the main experimental tool was a relatively large and detailed simulator I wrote for simulating software running on a distributed system.

The initial version of this simulator was written in Simula [Birtwistle,1979] and ran on the Cambridge University computer center's IBM 360/165 mainframe. It was a pleasure to write that simulator. The features of Simula were almost ideal for the purpose, and I was particularly impressed by the way the concepts of the language helped me think about the problems in my application. The class concept allowed me to map my application concepts into the language constructs in a direct way that made my code more readable than I had seen in any other language. The way Simula

classes can act as co-routines made the inherent concurrency of my application easy to express. For example, an object of class `computer` could trivially be made to work in pseudo-parallel with other objects of class `computer`. Class hierarchies were used to express variants of application-level concepts. For example, different types of computers could be expressed as classes derived from class `computer` and different types of inter-module communication mechanisms could be expressed as classes derived from class `IPC`. The use of class hierarchies was not heavy, though; the use of classes to express concurrency was much more important in the organization of my simulator.

During writing and initial debugging, I acquired a great respect for the expressiveness of Simula's type system and its compiler's ability to catch type errors. I observed that type errors almost invariably reflected either a silly programming error or a conceptual flaw in the design. The latter was by far the most significant and a help that I had not experienced in the use of more primitive "strong" type systems. In contrast, I had found Pascal's type system to be worse than useless – a straitjacket that caused more problems than it solved by forcing me to warp my designs to suit an implementation-oriented artifact. The contrast I perceived between the rigidity of Pascal and the flexibility of Simula was essential for the development of C++. Simula's class concept was seen as the key difference, and ever since I have seen classes as the proper primary focus of program design.

I had used Simula before (during my studies at the University of Aarhus, Denmark), but was very pleasantly surprised by the way the mechanisms of the Simula language became increasingly helpful as the size of the program increased. The class and co-routine mechanisms and the comprehensive type checking ensured that problems and errors did not (as I – and I guess most people – would have expected) grow more than linearly with the size of the program. Instead, the total program acted more like a collection of very small programs than a single large program and was therefore easier to write, comprehend, and debug.

The implementation of Simula, however, did not scale in the same way. As a result, the whole project came close to disaster. My conclusion at the time was that the Simula implementation (as opposed to the Simula language) was geared to small programs and was inherently unsuitable for larger programs [Stroustrup,1979]. Link times for separately compiled classes were abysmal: It took longer to compile 1/30th of the program and link it to a precompiled version of the rest than it took to compile and link the program as a monolith. This I believe, was more a problem with the mainframe linker than with Simula, but it was still a burden. On top of that, the run-time performance was such that there was no hope of obtaining useful data from the simulator. The poor run-time characteristics were a function of the language and its implementation rather than a function of the application. The overhead problems were fundamental to Simula and could not be remedied. The cost arose from several language features and their interactions: run-time type checking, guaranteed initialization of variables, concurrency support, and garbage collection of both user-allocated objects and procedure activation records. For example, measurements showed that more than 80% of the time was spent in the garbage collector despite the

fact that resource management was part of the simulated system so that no garbage was ever produced. Simula implementations are better these days (15 years later), but the order-of-magnitude improvement in run-time performance still has not (to the best of my knowledge) materialized.

To avoid terminating the project – and thus having to leave Cambridge without a Ph.D. – I rewrote the simulator in BCPL and ran it on the experimental CAP computer. The experience of coding and debugging the simulator in BCPL [Richards,1980] was horrible. BCPL makes C look like a very high-level language and provides absolutely no type checking or run-time support. The resulting simulator did, however, run suitably fast and gave a whole range of useful results that clarified many issues for me and provided the basis for several papers on operating system issues [Stroustrup,1978,1979b,1981].

Upon leaving Cambridge, I swore never again to attack a problem with tools as unsuitable as those I had suffered while designing and implementing the simulator. The significance of this to C++ was the notion I had evolved of what constituted a “suitable tool” for projects such as writing a significant simulator, an operating system, and similar systems programming tasks:

- [1] A good tool would have Simula’s support for program organization – that is, classes, some form of class hierarchies, some form of support for concurrency, and strong (that is, static) checking of a type system based on classes. This I saw (as I still see it today) as support for the process of inventing programs, as support for design rather than just for implementation.
- [2] A good tool would produce programs that run as fast as BCPL programs and share BCPL’s ability to easily combine separately compiled units into a program. A simple linkage convention is essential for combining units written in languages such as C, Algol68, Fortran, BCPL, assembler, etc., into a single program so that programmers can avoid getting caught by inherent limitations in a single language.
- [3] A good tool should also allow for highly portable implementations. My experience was that the “good” implementation I needed would typically not be available until “next year” and then only on a machine I couldn’t afford. This implied that a tool must have multiple sources of implementations (no monopoly would be sufficiently responsive to users of “unusual” machines or to poor graduate students), that there should be no complicated run-time support system to port, and that there should be only very limited integration between the tool and its host operating system.

These criteria were not fully formed when I left Cambridge. Some matured only on further reflection on my experience with the simulator, on programs written over the next couple of years, and on the experiences of others that I learned of through discussions and reading of code. C++ as defined at the time of Release 2.0 strictly fulfills these criteria; the fundamental tensions in the effort to design templates and exception handling mechanisms for C++ arise from the need to depart from some aspects of these criteria. I think the most important aspect of these criteria is that they are only loosely connected with specific programming language features. Instead,

they specify constraints on a solution.

At the time I was there, the Cambridge Computing Laboratory was headed by Maurice Wilkes. I received my main technical guidance from my supervisor, David Wheeler, and from Roger Needham. My background in operating systems and my interest in modularization and communication had permanent effects on C++. The C++ model of protection, for example, is based on the notion of granting and transferring access rights; the distinction between initialization and assignment has its root in thoughts about transferring capabilities; C++'s notion of `const` is derived from hardware read/write access protection mechanisms; and the design of C++'s exception handling mechanism was influenced by work on fault-tolerant systems done by Brian Randell's group in Newcastle during the seventies.

1.2 C and Systems Programming

I had briefly encountered C in London in 1975 and acquired some respect for it compared to other languages of the kind referred to as systems programming languages, machine-oriented languages, or low-level languages. Of those, I knew PL360, Coral, Mary, and others, but my main experience with languages of this class was BCPL. In addition to being a BCPL user, I had once implemented BCPL by microcoding its intermediate form, O-code, so I had a detailed understanding of the low-level efficiency implications of this class of languages.

After finishing my Ph.D. Thesis in Cambridge and getting a job at Bell Labs, I (re)learned C from [Kernighan,1978]. Thus, at the time, I was not a C expert and saw C primarily as the most modern and prominent example of the systems programming languages. Only later did I achieve a fuller understanding of C based on personal experience and discussion with people such as Stu Feldman, Steve Johnson, Brian Kernighan, and Dennis Ritchie. The general idea of a systems programming language thus determined the growth of C++ to at least the same extent as did the specific language-technical details of C.

I knew Algol68 [Woodward,1974] pretty well from using it for minor projects in Cambridge. I appreciated the relationship between its constructs and those of C, and sometimes find it useful to consider C constructs as specialized versions of Algol68's more general concepts. Curiously enough, I did not see Algol68 as a systems programming language (despite having used an operating system written in Algol68). I suspect the reason was the emphasis I placed on portability, ease of linkage to code written in other languages, and run-time efficiency. I have on occasion described my dream language as Algol68 with Simula-like classes. However, for building a practical tool, C seemed a much better choice than Algol68.

1.3 General Background

It is often claimed that the structure of a system reflects the structure of the organization that created it. Within reason, I subscribe to that idea. It follows that when a system is primarily the work of an individual, the system reflects the fundamental outlook of that individual. In retrospect, I think the overall structure of C++ was shaped as much by my general “world view” as it was shaped by the detailed computer science concepts used to form its individual parts.

I studied pure and applied mathematics so that my Danish “masters degree” (a Cand.Scient. degree) is in mathematics and computer science. This left me with an appreciation of the beauty of mathematics, but also with a bias towards mathematics as a practical tool for problem solving as opposed to an apparently purposeless monument to abstract truth and beauty. I have a lot of sympathy for the student Euclid reputedly had evicted for asking, “But what is mathematics for?” Similarly, my interest in computers and programming languages is fundamentally pragmatic. Computers and programming languages can be appreciated as works of art, but aesthetic factors should complement and enhance utility, not substitute for or compromise utility.

My long-term (continuous for at least 25 years) hobby is history, and I spent significant time in university and later studying philosophy. This has given me a rather conscious view of where my intellectual sympathies lie and why. Among the long-standing schools of thought, I feel most at home with the empiricists rather than with the idealists – the mysticists I just can’t appreciate. That is, I tend to prefer Aristotle to Plato, Hume to Descartes, and shake my head sadly over Pascal. I find comprehensive “systems” like those of Plato and Kant fascinating, yet fundamentally unsatisfying in that they appear to me dangerously remote from everyday experiences and the essential peculiarities of individuals.

I find Kierkegaard’s almost fanatical concern for the individual and keen psychological insights much more appealing than the grandiose schemes and concern for humanity in the abstract of Hegel or Marx. Respect for groups that doesn’t include respect for individuals of those groups isn’t respect at all. Many C++ design decisions have their roots in my dislike for forcing people to do things in some particular way. In history, some of the worst disasters have been caused by idealists trying to force people into “doing what is good for them.” Such idealism not only leads to suffering among its innocent victims, but also to delusion and corruption of the idealists applying the force. I also find idealists prone to ignore experience and experiment that inconveniently clashes with dogma or theory. Where ideals clash and sometimes even when pundits seem to agree, I prefer to provide support that gives the programmer a choice.

My preferences in literature have reinforced this unwillingness to make a decision based on theory and logic alone. In this sense, C++ owes as much to novelists and essayists such as Martin A. Hansen, Albert Camus, and George Orwell, who never saw a computer, as it does to computer scientists such as David Gries, Don Knuth, and Roger Needham. Often, when I was tempted to outlaw a feature I personally

disliked, I refrained from doing so because I did not think I had the right to force my views on others. I know that much can be achieved in a relatively short time by the energetic pursuit of logic and by ruthless condemnation of “bad, outdated, and confused habits of thought.” However, the human cost is often high. A high degree of tolerance and acceptance that different people do think in different ways and strongly prefer to do things differently is to me far preferable.

My preference is to slowly – often painfully slowly – persuade people to try new techniques and adapt the ones that suit their needs and tastes. There are effective techniques for achieving “religious conversions” and “revolutions,” but I have fundamental qualms about those techniques and grave doubts about their effects in the long term and on a large scale. Often, if someone can be easily converted to “religion” X, a further conversion to “religion” Y is likely, and the gain ephemeral. I prefer skeptics to “true believers.” I value a small piece of solid evidence over most theories, and a solid experimental result over most logical arguments.

These views could easily lead to fatalistic acceptance of status quo. After all, one cannot make an omelet without cracking a few eggs and most people do not actually want to change – at least “not just now” or in ways that will disrupt their everyday lives. This is where respect for facts comes in – and a modicum of idealism. Things in programming and in the world in general really aren’t in a very good state, and much can be done to improve them. I designed C++ to solve a problem, not to prove a point, and it grew to serve its users. The underlying view is that it is possible to achieve improvements through gradual change. The ideal situation is to maintain the greatest rate of change that improves the welfare of the individuals involved. The main difficulties are to determine what constitutes progress, to develop techniques to smooth transitions, and to avoid excesses caused by over-enthusiasm.

I’m willing to work hard for the adoption of ideas that I have become convinced will be of help to people. In fact, I consider it the obligation of scientists and intellectuals to ensure that their ideas are made accessible and thus useful to society instead of being mere playthings for specialists. However, I’m not willing to sacrifice people to ideas. In particular, I do not try to enforce a single style of design through a narrowly defined programming language. People’s ways of thinking and working are so diverse that an attempt to force a single style would do more harm than good. Thus, C++ is deliberately designed to support a variety of styles rather than a would-be “one true way.”

Chapter 4 presents the more detailed and practical rules that guided the design of C++. In those rules, you can find the echoes of the general ideas and ideals mentioned here.

A programming language can be the most important factor in a programmer’s day. However, a programming language is really a very tiny part of the world, and as such, it ought not be taken too seriously. Keep a sense of proportion and – most importantly – keep a sense of humor. Among major programming languages, C++ is the richest source of puns and jokes. That is no accident.

Philosophy, like discussion of language features, does tend to get overly serious and preachy. For this, I apologize, but I felt like acknowledging my intellectual roots

and I guess this is harmless – well, mostly harmless. And no, my preferences in literature are not limited to writers emphasizing philosophical and political themes; those are just the ones who left the most obvious traces in the fabric of C++.

This page intentionally left blank

I

Index

=0 279
 ?= 326
 100% C compatibility 101
 2.0, Release 187
 7-bit character 158
 8-bit character set 161
 :, comp.lang.c++ 168
 ,, operator 247
 %: digraph 160
 %: %: digraph 160
 %> digraph 160
 *, ** and 248
 **
 and * 248
 exponentiation operator 247
 *^, exponentiation operator 249
 ++
 increment operator 245
 prefix and postfix 245
 --
 decrement operator 245
 prefix and postfix 245
 ->
 delegation and 242
 operator 241
 ->*, operator 247
 .
 :: vs. 95
 operator 242
 .*, operator 247
 . . . , ellipsis 333
 //
 BCPL comments 44, 93
 and C compatibility 93
 ::
 explicit qualification 412
 vs. . 95
 :> digraph 160

<% digraph 160
 < . . . > syntax 341, 355
 <: digraph 160
 <<
 operator 186
 output operator 186
 =
 Release 2.0 82
 operator 239
 >> problem with template syntax 356
 0
 NULL 230
 false 254
 null pointer 229
 =0 syntax 281
 1, true 254

A

abolish Cpp 426
 abomination *f (void)* 41
 absence of template 378
 abstract
 class 35, 261, 277
 class and library 183
 class, template and 379
 type 279
 abstraction, C++ and data 197
 academic pursuit of learning 70
 acceptance
 criteria, feature 61
 process, feature 61
 accepted
 extensions 151
 features 151
 proposals 151
 accepting a proposal 147
 access

- control 31, 301
- control and `static` member 288
- control for constructor 94
- control of names 54
- granting 53–54
- to base class 301
- using-declaration* adjusting 419
- vs. visibility 54
- access-declaration* 54
- and *using-declaration* 419
- deprecated 419
- accessible, C++ 197
- accident, protection against 116
- Ada 6, 44, 67, 105, 107–108, 175, 339
- C++ and 192
- I/O 185
- complexity 199
- extensions 147
- new 365
- Ada9X 6, 177, 202
- Adcock, Jim 243
- address, special 214
- adjusting access, *using-declaration* 419
- advantages and disadvantages, C 44
- advertising, AT&T C++ 177
- affordability of C with Classes 33
- affordable features 115
- :*after*, *return()* and 57, 268
- Agrawal, Sudhir 61
- Aho, Al 68, 105
- aims
 - AT&T 178
 - C with Classes 28
 - for exception handling 384
 - for standardization 130
 - of C++ 1, 65, 109, 196
 - of early libraries 184
 - of namespace design 402
 - of standardization 135
- Algol60 265
- Algol68 6, 22, 40, 84–85, 100, 105, 108
- a rose 37
- declaration 99
- operators 250
- references 86
- references overloading 44
- algorithms, standard 194
- alias 403
- namespace 404, 408
- problem with 275
- allocation 357
- and constructor, new 56
- and initialization 212–213
- and real-time, dynamic 211
- arena 214
- array 213
- new and 212
- prohibit `auto` 237
- prohibit free store 238
- prohibit global 237
- prohibit `static` 237
- allocator, per-class 91, 211
- alternative
 - RTTI design 323
 - points of destruction of temporary 145
 - template syntax 355
 - to `#ifdef`, `if` 425
 - to `#include`, `include` 425
 - to Cpp 425
 - to cast 309
- ambiguity 227
- control 226
- in template 371
- multiple inheritance and 258
- American national standard 136
- analysis, one-pass 140
- `and` keyword 160
- `and_eq` keyword 160
- Annemarie 404
- ANSI
 - C C85 64
 - C grammar 69
 - C local static arrays 128
 - C volatile 128
 - C++ standard 128
 - X3J16 136
 - and ISO 136
 - standard library 194
- ANSI3.4-1968, ISO-646-1983 ASCII 158
- ANSI/ISO name lookup 141
- Apple
 - Mac MPW 167
 - Macintosh 165
- application
 - environment, standard 135
 - programming, general 204
- archetypical problems 62
- architecture-specific extension 158
- arena
 - allocation 214
 - memory 215
- argument
 - check, run-time 296
 - constraint on template 343
 - deducing template 349
 - default 59
 - depend on template 370
 - explicit template 350
 - function template 350
 - keyword 153
 - matching rules 228
 - named 153

- names 155
 - namespace as `template` 343
 - non-type `template` 342
 - over-constrained 344
 - rules, relaxation of 295
 - `template` as `template` 343
 - workaround, keyword 156
 - arguments
 - average number of 156
 - for resumption 391
 - for termination 391
 - value of 301
 - arithmetic, mixed-mode 80, 223
 - ARM 124, 172
 - C++ in 127
 - features 128
 - learning C++ from 127
 - name lookup 139
 - purpose of 127
 - reference manual 126
 - array
 - allocation 213
 - associative 322
 - deallocating 217
 - standard associative 194
 - `template` 342
 - arrays, ANSI C local static 128
 - ASCII ANSI3.4-1968, ISO-646-1983 158
 - assembler
 - C++ and 120, 201
 - learning 169
 - `Assert()` 397
 - `assert()` 397
 - assignment 239
 - initialization and 22
 - overloading of 58
 - to `this` 91, 212
 - `void*` 230
 - associative
 - array 322
 - array, standard 194
 - assumptions for exception handling 384
 - asynchronous events 393
 - AT&T 100, 128–129
 - C++ advertising 177
 - C++ marketing 177
 - C++ use within 65
 - aims 178
 - compiler 166
 - library 184
 - Austern, Matt 248
 - auto allocation, prohibit 237
 - automatic
 - garbage collection 219
 - storage 32
 - `template` instantiation 365
 - autoprototyping 40
 - availability, commercial 70
 - available, C is 43
 - average number of arguments 156
- ## B
- Babcisky, Karel 32, 106
 - `bad_cast` 194
 - Ball, Mike 166
 - base
 - and derived 74
 - and derived, overload 417
 - and derived overloading 77
 - cast from `virtual` 267, 312
 - cast to `private` 328
 - `class` 49
 - `class`, access to 301
 - `class`, ordinary 258
 - `class`, `virtual` 260
 - derived hiding 417
 - first construction 282
 - initializer 275
 - layout, `virtual` 266
 - member, hiding of 77
 - basic language support 194
 - `basic_string` 359
 - Bauche, Volker 400
 - BCPL 21–22, 64
 - comments `//` 44, 93
 - speed 21
 - beauty and efficiency, C++ 379
 - `:before, call()` and 57, 268
 - `before()` 317
 - behavior, implementation-dependent 133–134
 - Bell
 - Labs 27, 174, 177
 - Labs CSRC 60
 - Labs C++ developers 101
 - Labs library 184
 - Bennett, Tanj 409
 - Beta 6, 326
 - better
 - C, C++ not just a 204
 - C subset of C++ 171
 - match 228
 - bigotry, language wars and 299
 - binary literal 152
 - binding
 - of reference 86
 - rule, name 368
 - `bitand` keyword 160
 - `bitor` keyword 160
 - `bits<N>` 194
 - `bitstring` 194

- bitwise copy 239
- BIX c.plus.plus 166
- blackboards, design on 61
- BLAS 158
- Booch, Grady 192, 271
- Booch
 - components 358, 379
 - components library 192
- book for users 4
- books, journals and 165
- bool, Boolean type 254
- Boolean type bool 254
- bored programmer 388
- Borland 164
 - compiler 166
- bridge 200
- Bright, Walter 40, 84, 166
- Brown, Phil 125
- Bruns, John 294, 400
- Brück, Dag 145, 248, 254–255, 287, 290, 327, 391–392, 400
- Budge, Ken 192, 194
- Buffer 342, 349
- bugs, multiple inheritance 270
- built-in
 - type, constructor for 380
 - type, derivation from 380
 - type, destructor for 380
 - types, definition of 380
 - types, template and 380
 - types, user-defined and 32, 117

C

- ++C 64
- C 22, 64
 - C++ and 100, 176, 201
 - C++ not just a better 204
 - C++ separate from 65
 - I/O 96
 - I/O printf 185
 - a weed 37
 - advantages and disadvantages 44
 - and C++ compatibility 120, 129
 - and C++ const 90
 - and C++, learning 169
 - and C++ linkage 233
 - and C++ reference manual 170
 - and C++ textbooks 170
 - and Simula 1
 - code generated 39
 - compatibility 131, 225, 227, 229, 240
 - compatibility, // and 93
 - compatibility, 100% 101
 - compatibility, C with Classes 38
 - compatibility, conversion and 227
 - compatibility, namespaces and 420
 - compiler, PCC 69
 - declaration syntax 45
 - dialects 101
 - enum 253
 - error handling 338
 - exceptions and 385
 - extensions 147
 - features, dangerous 29
 - features, ugly 29
 - fixing 45
 - generated 67
 - in C++, writing 170
 - is available 43
 - is efficient 43
 - is flexible 43
 - is portable 43
 - learning 169
 - linkage 232
 - numerical 157
 - preprocessor 68
 - preprocessor Cpp 119, 423
 - problem with 374
 - semantics 103
 - techniques, undesirable 378
 - variables 198
 - why 43
 - with Classes 27
 - with Classes C compatibility 38
 - with Classes a weed 37
 - with Classes, affordability of 33
 - with Classes aims 28
 - with Classes dilemma 63
 - with Classes documentation 29
 - with Classes efficiency 28
 - with Classes features 29
 - with Classes implementation 29
 - with Classes layout compatibility 28
 - with Classes low-level features 29
 - with Classes object layout 38
 - with Classes preprocessor, Cpre 27
 - with Classes run-time support 49
 - with Classes static type checking 40
 - with Classes strong type checking 40
 - with Classes, success of 64
 - with Classes support 29
 - with Classes, supporting 63
 - with Classes without virtual function 53
 - withdrawal symptoms 93
- C++ 147, 271
- .c file and template 376
- C84 64
- C85, ANSI C 64
- call
 - in constructor, virtual 282
 - notation, constructor 335

- of undeclared function 40
 - optimization of virtual 116
- call() and :before 57, 268
- callbacks 303
- call-by-reference 86
- call-by-value 86
- calling
 - Fortran 86
 - conventions 236
 - sequences, standard 135
- Cambridge 19, 64
- Computing Laboratory 22
- CAP computer, capability-based 19
- capabilities 22
- capability-based CAP computer 19
- Cargill, Tom 47, 105, 125, 269
- Carolan, John 164
- Carroll, Martin 269
- cast
 - RTTI and 309
 - alternative to 309
 - and const 331
 - and incomplete type 329
 - and interface 183
 - and pointer to function 331
 - away const 285, 332
 - eliminate 309
 - from virtual base 267, 312
 - from void* 333
 - implicit 350
 - new-style 327
 - notation, new 327
 - notation, old-style 333
 - overuse of 170
 - problems with 327
 - safe 309
 - syntax 309
 - to private base 328
 - transition to new 335
 - using new 335
- catastrophic failure, crashes and 93
- catch 385
- CBEMA 136
- Cedar Mesa 392
- Centerline ObjectCenter 167
- cerr 186
- Cfront 124
 - code quality 302
 - compile-time overhead 67
 - compiler front-end 66
 - first shipment 70
 - implementation of new 57
 - memory use 66
 - resource requirements 65
 - size of 66
 - structure of 66
- symbol tables 66
 - template 339
 - template implementation 365
 - to PC, port of 66
 - virtual table layout 320
 - warning 43
- C++
 - Journal, The 165
 - Release 1.0 deficiencies 107
 - Release 1.0 features 71
 - Report, The 165
 - Simula and 106
 - Smalltalk and 107
 - accessible 197
 - advertising, AT&T 177
 - aims of 1, 65, 109, 196
 - and Ada 192
 - and C 100, 176, 201
 - and Fortran 176
 - and Smalltalk, learning 170
 - and assembler 120, 201
 - and data abstraction 197
 - and generic programming 379
 - and object-oriented programming 197
 - and other languages 197
 - as a target language 204
 - as an implementation language 204
 - beauty and efficiency 379
 - better C subset of 171
 - committee charter 130
 - community, diversity of 130
 - compatibility, C and 120, 129
 - compilers sold, number of 164
 - compositional 188
 - concurrent 188
 - const, C and 90
 - design of 60
 - design using 171
 - developers, Bell Labs 101
 - development plan 124
 - exception handling, ideals for 385
 - features 131
 - from ARM, learning 127
 - general-purpose programming language 105
 - goals of 196
 - gradual approach to 171
 - is coherent 197
 - language definition 5
 - learning 168, 172
 - learning C and 169
 - libraries 65
 - linkage, C and 233
 - manual, size of initial 79
 - marketing, AT&T 177
 - micro 188

- model of protection 22
- multiple inheritance 271
- name 64
- not just a better C 204
- operating system influence on 22
- parsing 68
- people who contributed to 2
- prehistory of 19
- programming environment 167
- programming environment, future 167
- programming environment, ideals for 167
- projects, major 174
- purpose of 105, 163
- reference manual 60
- reference manual, C and 170
- separate from C 65
- serving current users 110
- source 207
- standard, ANSI 128
- standardization, ISO 129
- standardization, start of 128
- static type checking 197
- stream I/O 185
- strength 208
- syntax type-system 208
- teaching 168
- textbooks, C and 170
- time needed to learn 172
- timeline 4
- tools 65
- type checking 92
- use within AT&T 65
- users, counting 164
- users, number of 163
- versions of 71
- writing C in 170
- writing Fortran in 170
- writing Smalltalk in 170
- C++ Programming Language, the* 105
- C++'s contribution 196
- change
 - gradual 24
 - impact of 281
 - motivation for 110
 - rate of 24
- char
 - constant, type of 225
 - int and 224
 - literal, type of 225
 - overloading and 224
- character
 - 7-bit 158
 - set 158
 - set, 8-bit 161
 - set, unicode 161
 - sets, I/O of extended 161
 - sets, extended 161
 - sets, national 158
- character-oriented tools 207
- characters, limit on number of 69
- char_ref 88
- chart, language 5
- charter, C++ committee 130
- check
 - list, extension 149
 - run-time 42
 - run-time argument 296
- checked interfaces, statically 107
- checking
 - and error 378
 - of exceptions, dynamic 396
 - of exceptions, static 395
 - of multiple inheritance, static 263
 - of template, error 369
- checks, run-time 29
- Cheriton, David 384
- choice 113
- Choices 165
- choosing a programming language 175
- cin, initialization of 96
- CLAM 165
- Clamage, Steve 166
- clarifications 138
- clarity of definition 103
- class
 - helper 87
 - hierarchy, exceptions and 387
 - layout, derived 52
 - member, forward reference of 140
 - string 66
- class
 - Simula 6, 44
 - Simula style 21
 - abstract 35, 261, 277
 - access to base 301
 - and library, abstract 183
 - and macro, container 337
 - and template, container 337
 - as namespace 401
 - base 49
 - concept 30
 - derived 49, 301
 - forward declaration of nested 289
 - hierarchies, merging of 273
 - hierarchy 20
 - hierarchy, overloading and 226
 - is user-defined type 30
 - locality nested 118
 - member 289
 - member order dependencies 138
 - namespace and 417
 - nested 102, 128, 289

- ordinary base 258
- prefix, Simula 49
- root 261
- sibling 261
- struct and 76
- template 341
- template and 360
- template and abstract 379
- unit of protection 54
- universal root 184
- virtual base 260
- cleanup
 - deallocation and 212
 - of template design 340
- clone() 293
- CLOS 268, 272, 299
 - complexity 199
 - learning 169
- Clu 6, 44, 105, 107–108, 397
- Clu-style container 339
- CMP 359
- Cobol extensions 147
- code
 - bloat, curbing 347
 - exceptions and old 394
 - generation 302
 - quality, Cfront 302
 - replication 346
 - verbose error-prone 388
- coding, low-level 338
- coexistence 112
- coherence, maintaining 153
- coherent
 - C++ is 197
 - view 148
- collating sequence 358
- combination, method 268
- Comeau, Greg 166
- comments //, BCPL 44, 93
- commercial
 - availability 70
 - competition 175
- committee
 - charter, C++ 130
 - dangers of design by 148
 - design by 60, 147
 - disband 148
 - for a new language, design 148
 - members 137
 - standards 128
- communication between siblings 261
- community, diversity of C++ 130
- company language 178
- comparing programming languages 5
- comparison, language 270
- compatibility
 - C 131, 225, 227, 229, 240
 - C and C++ 120, 129
 - C with Classes C 38
 - C with Classes layout 28
 - concern for 153
 - hack, structure tag 48
 - link 120
 - vs. safety 334
 - wars 102
- competition, commercial 175
- compilation
 - model, template 376
 - one-pass 139
 - separate 21, 34
- compiler
 - AT&T 166
 - Borland 166
 - DEC 166
 - GNU 166
 - IBM 166
 - Microsoft 166
 - Oregon Software 166
 - PCC C 69
 - TauMetric 166
 - Zortech 166
- complexity 111
 - front-end, Cfront 66
 - has partial information 37
 - inlining linker 33
 - warning 42, 334
- compilers 166
 - sold, number of C++ 164
- compile-time 281
 - and template 366
 - overhead 233
 - overhead, Cfront 67
 - protection 54
 - type checking 31
 - type inquiry 353
- comp1 keyword 160
- comp.lang.c++ 231
 - : 168
 - usenet 166
- complex 79, 194, 370
 - library 66
 - template 362
- complexity
 - Ada 199
 - CLOS 199
 - Eiffel 199
 - RTTI implementation 306
 - Smalltalk 199
- compiler 111
 - environment 199
 - language 199
 - library 199

- multiple inheritance 270
- of use 111
- simplicity and 112
- components industry, software 191
- composite operator 251
- composition 116
 - interface 409
 - techniques, template 356
- compositional C++ 188
- computer, high-performance 176
- Computing Laboratory, Cambridge 22
- computing
 - engineering 176
 - numeric 203
 - numerical 158
 - scientific 158, 176, 203
- concepts
 - and language constructs 19
 - focus on 168
- concern for compatibility 153
- concurrency 20, 27
 - and exceptions 385
 - support 188
 - support extensions 188
 - support libraries 188
- concurrent C++ 188
- condition, declaration in 99
- conditionals in template 353
- conferences 164
- conservative garbage collection 217
- consistency 37
- const 22
 - C and C++ 90
 - Cpp and 425
 - as symbolic constant 90
 - cast and 331
 - cast away 285, 332
 - constants 89
 - enforcement of 284
 - for specifying interface 90
 - ideal 286
 - initialization of 286
 - member function 284
 - member initializer 151
 - overloading and 225
 - references 86
 - static_cast and 329
 - ~const 287
- constant
 - const as symbolic 90
 - type of char 225
- constants const 89
- const_cast 332
- constraint
 - and error detection 344
 - and readability 344
 - on template argument 343
 - through derivation 344
 - through inheritance 344
 - through use 345
- constraints 346
- construction, base first 282
- constructor 94
 - access control for 94
 - and library 183
 - call notation 335
 - copy 240
 - default 59, 381
 - exception and 388
 - for built-in type 380
 - new allocation and 56
 - new and 213
 - new() 56
 - notation 94
 - template and 379
 - virtual call in 282
- contain using macro 50
- container
 - Clu-style 339
 - Smalltalk-style 339
 - class and macro 337
 - class and template 337
 - without template 50
- Container template 315
- containers, standard 194
- context
 - of template definition 369
 - of template instantiation 369
- contract
 - between implementer and programmer 133
 - standard as a 133
- contributed to C++, people who 2
- contribution, C++'s 196
- control
 - access 31, 301
 - manual 121
 - of copy 237
 - of memory exhaustion 218
 - of memory management, fine-grain 211
- Controlled_container 357
- controversy, multiple inheritance 269
- convenience vs. safety 411
- conventions, calling 236
- conversion
 - and C compatibility 227
 - and template 362
 - floating to integral 42
 - function 83
 - graph 228
 - implicit 80, 223
 - implicit narrowing 334
 - lattice 228

- narrowing 41, 227
- natural 227
- overloading and 223
- safe 227
- standard 228
- static_cast and implicit 329
- to void* 226
- convert() 350
- Coplien, Jim 66, 105, 125
- copy
 - bitwise 239
 - constructor 240
 - control of 237
 - deep 240
 - memberwise 239
 - of object 293
 - of pointer 240
 - prohibit 237
 - shallow 240
 - virtual 241
- copy() 77
- Coral 22
- CORBA 206
- Corfield, Sean 255
- cost of exceptions 385
- counting C++ users 164
- coupling 281
- cout 186
 - initialization of 96
- CPL 64
- c.plus.plus, BIX 166
- Cpp
 - C preprocessor 119, 423
 - abolish 426
 - alternative to 425
 - and const 425
 - and error 424
 - and inline 425
 - and namespace 425
 - and programming environment 424
 - and scope 424
 - and templates 425
 - and tools 424
 - in ARM 127
 - macro 118, 424
- Cpre
 - C with Classes preprocessor 27
 - first shipment 70
- crashes and catastrophic failure 93
- Cray 157
- Cristian, Flaviu 392
- criteria
 - extension 149
 - feature acceptance 61
 - for templates, design 338
 - tools 21

- Crockett, Beth 274
- crossing protection barrier 32
- CSRC, Bell Labs 60
- cult
 - language 229
 - of orthogonality 104
- curbing code bloat 347
- current users, C++ serving 110

D

- DAG, dominance 263
- Dahl, Ole-Johan 258
- dangerous
 - C features 29
 - feature 315
- dangers of design by committee 148
- data
 - abstraction, C++ and 197
 - member, pointer to 304
 - virtual 266
- databases, object-oriented 192
- deallocating array 217
- deallocation
 - and cleanup 212
 - delete and 212
 - problems 216
- debugger, standard 135
- debugging support 42
- DEC 128
 - VMS 392
- compiler 166
- declaration
 - Algol68 99
 - implementation 35
 - in condition 99
 - in *for-statement* 99
 - of nested class, forward 289
 - overload 231
 - specialization 375
 - syntax, C 45
- declarators, linear notation for 46
- decrement operator -- 245
- deducing template argument 349
- deep copy 240
- default
 - argument 59
 - arguments, overloading and 59
 - constructor 59, 381
 - local 101
 - nested 101
 - operator 237
 - private 101
 - type checked 101
- deficiencies, C++ Release 1.0 107
- #define 424

- definition
 - C++ language 5
 - clarity of 103
 - context of `template` 369
 - error, double 377
 - finding `template` 376
 - formal 103
 - function 31
 - of built-in types 380
 - of simple 403
- delay causes dialects 129
- delegation 242
 - and `->` 242
 - experiment 272
 - multiple inheritance 270
- `delete`
 - and deallocation 212
 - and destructor 213
 - `free()` 57
 - operator 32, 212, 216
- `delete()`, destructor 56
- `delete[]` 217
 - operator 214
- depend on `template` argument 370
- dependencies
 - implementation 138
 - on library 134
- deprecate implicit `int` 45
- deprecated
 - access-declaration* 419
 - global static* 419
- derivation
 - constraint through 344
 - from built-in type 380
 - from `int` 380
 - prohibit 238
- derived
 - base and 74
 - `class` 49, 301
 - class layout 52
 - hiding base 417
 - overload base and 417
 - overloading, base and 77
- design
 - aims of namespace 402
 - alternative RTTI 323
 - and problems 105
 - and teaching 119
 - and type checking 107
 - by committee 60, 147
 - by committee, dangers of 148
 - cleanup of `template` 340
 - committee for a new language 148
 - criteria for templates 338
 - from first principles 44
 - hybrid 201
 - ideals for namespace 402
 - language 114
 - library 182
 - non-aims of namespace 403
 - object-oriented 114, 172
 - of C++ 60
 - of fault-tolerant system 385
 - on blackboards 61
 - onion 323
 - paper 60
 - principles, language 109
 - rules, language 109
 - static type checking and 107
 - styles of 24
 - support rules 114
 - teaching and 168
 - `template` and library 378
 - too cautious, `template` 340
 - tools for language 103
 - tradeoffs, library 182
 - using C++ 171
 - warped 338
- designer and implementer 60
- destruction of temporary, alternative points
 - of 145
- destructor 94
 - and garbage collection 222
 - `delete` and 213
 - `delete()` 56
 - exception and 388
 - explicit call of 217
 - for built-in type 380
- destructors, `virtual` 216
- details, language-technical 3
- detectable, extensions are 134
- Deutsch, Peter 387
- development
 - environment, ideal program 207
 - management 124
 - plan, C++ 124
- Dewhurst, Steve 71, 125, 166, 304
- dialects
 - C 101
 - delay causes 129
- `dialog_box` 307
- digraph
 - `%:` 160
 - `%%:` 160
 - `%>` 160
 - `>` 160
 - `<%` 160
 - `<:` 160
- dilemma, C with Classes 63
- direct expression 115
- directive, `template` instantiation 367
- disadvantages, C advantages and 44

- disband committee 148
- dispatch
 - double 300
 - template, double 364
- displayed 258–259
- disruption of standards process 137
- distributed
 - fat 121, 199
 - system simulator 19
- diversity of C++ community 130
- documentation, C with Classes 29
- dominance 263
 - DAG 263
- double
 - definition error 377
 - dispatch 300
 - dispatch template 364
- double, float and 224
- Dovich, Steve 400
- dynamic
 - allocation and real-time 211
 - checking of exceptions 396
 - initialization 98, 286
 - linking 206
 - storage 32
 - store 56
 - type checking, static and 107
- dynamic_cast 308
 - static_cast and 330
 - syntax of 309
 - use of 296, 320
- dynarray 194

E

- early
 - libraries, aims of 184
 - users 174
- Eaves, Laura 71, 125
- EBCDIC, IBM 160
- Eckel, Bruce 154, 191
- education and technique 205
- efficiency
 - C with Classes 28
 - C++ beauty and 379
 - Fortran 157
 - and overloading 84
 - exceptions and 397
 - flexibility and 379
 - organization, flexibility 1
 - overloading and 78
 - run-time 32, 179, 324
 - virtual function 49, 75
- efficient
 - C is 43
 - I/O 185
 - Matrix 252
 - String, simple and 185
- Eick, Stephen 193
- Eiffel 6, 67, 176–177, 202, 326, 397
 - complexity 199
 - learning 169
 - elegance, simplicity and 229
 - elegant language 179
- eliminate cast 309
- elimination of temporaries 252
- ellipsis . . . 333
- Ellis, Margaret 127, 247
- email internet, newsgroups 164
- embedded system 203
- employee 292
- encapsulation and friend 53
- encoding, type 234
- Energize, Lucid 167
- enforcement 24
 - of const 284
- engineering
 - computing 176
 - language design and 104
- English as a common language 161
- enum
 - C 253
 - overload based on 254
 - type 253
- enumeration 253
 - overload based on 254
- environment
 - C++ programming 167
 - Cpp and programming 424
 - complexity 199
 - execution 221
 - future C++ programming 167
 - ideal program development 207
 - ideals for C++ programming 167
 - separation between language and 197
 - standard application 135
 - tool poor 65, 166
- environments, many 167
- EOS 145
- equivalence
 - name 35
 - operator 242
 - structural 35
- Eratosthenes' sieve 188
- error
 - Cpp and 424
 - checking and 378
 - checking of template 369
 - detection, constraint and 344
 - double definition 377
 - handling 218
 - handling, C 338

- handling, multi-level 385
 - potential 113
 - error-prone code, verbose 388
 - Estes Park workshop 165
 - ET++ 167
 - evaluating a programming language 175
 - events
 - Release 2.0 124
 - asynchronous 393
 - sequence of 123
 - evidence and skeptics 24
 - evolution, interface 206
 - example
 - member `template` 361–362
 - of RTTI 320
 - of object I/O 320
 - task 188
 - examples, pathological 141
 - exception
 - and constructor 388
 - and destructor 388
 - and library 183
 - handling 108, 383
 - handling, aims for 384
 - handling and reliability 384
 - handling, assumptions for 384
 - handling, ideals for C++ 385
 - handling, multiple inheritance and 387
 - handling syntax 385
 - specification 396
 - exceptions 22
 - and C 385
 - and class hierarchy 387
 - and efficiency 397
 - and layout 397
 - and levels of abstraction 392
 - and old code 394
 - and other languages 394
 - and recompilation 396
 - and resource management 388
 - concurrency and 385
 - cost of 385
 - dynamic checking of 396
 - experience with 392
 - grouping of 385–386, 396
 - guaranteed catching of 385
 - large system and 392
 - memory exhaustion and 390
 - multi-level propagation of 394
 - overflow and 394
 - static checking of 395
 - `template` and 379
 - type safety and 385
 - execution environment 221
 - experience 126
 - idealists and 23
 - rules and 109
 - `template` implementation 339, 368
 - `template` use 339
 - value of 301
 - with exceptions 392
 - with type-safe linkage 234
 - experiment 104
 - delegation 272
 - explicit
 - call of destructor 217
 - qualification 406
 - qualification :: 412
 - `template` argument 350
 - `template` instantiation 366
 - type field 50
 - `explicit` 80
 - exponentiation
 - operator `**` 247
 - operator `**^` 249
 - expression, direct 115
 - extended
 - character sets 161
 - character sets, I/O of 161
 - type information 318
 - extending vs. mutating 85
 - extensible I/O 185
 - extension
 - architecture-specific 158
 - check list 149
 - criteria 149
 - major and minor 305
 - stability and 125
 - extensions 269
 - Ada 147
 - C 147
 - Cobol 147
 - Fortran 147
 - Modula-2 147
 - Pascal 147
 - accepted 151
 - and standard 134
 - are detectable 134
 - concurrency support 188
 - need for 129
 - problems with good 152
 - proposals, management of 147
 - suggested 150
 - extern C 234
 - extreme portability 65
- ## F
- `f()`, `f(void)` and 41
 - facts 1
 - respect for 179
 - reviewed 2

- vs. philosophy 3
- `false` 0 254
- `fancyString` 185
- fat, distributed 121, 199
- fatalism status quo 24
- fault-tolerant system, design of 385
- fear of overloading 231
- feature
 - acceptance criteria 61
 - acceptance process 61
 - dangerous 315
 - library vs. language 181
 - major and minor 305
- features
 - ARM 128
 - C with Classes 29
 - C++ 131
 - C++ Release 1.0 71
 - Release 2.0 126
 - accepted 151
 - affordable 115
 - and techniques, language 170
 - language design and 104
 - necessary 149
 - new 269
 - removing 269
- featurism 60
- feedback, perfection and 111
- Feldman, Stu 22, 39, 78, 105
- `Fig` 295
- file
 - and `template, .c` 376
 - and `template, .h` 376
 - and `template, source` 376
 - header 34, 207
 - source 207
- `FilePtr` 389
- files, header 235
- finalization 388
- finding `template` definition 376
- fine-grain
 - control of memory management 211
 - overload resolution 224
- fire walls 394
- first
 - principles, design from 44
 - shipment, Cfront 70
 - shipment, Cpre 70
- first-time switch 97
- fixing C 45
- flame war 231
- flexibility
 - and efficiency 379
 - efficiency organization 1
 - vs. simplicity 224
- flexible, C is 43
- `float`
 - and `double` 224
 - overloading and 224
- floating
 - point, single precision 224
 - to integral conversion 42
- focus on concepts 168
- Fontana, Mary 339, 392
- fooling `static_cast` 330
- `fopen()` 388
- force 113
 - for instability, standardization as 148
- forcing people 23, 171
- foreign language interface 236
- foreman 292
- formal
 - definition 103
 - methods 103
- for-statement*, declaration in 99
- Fortran 34, 120, 203
 - C++ and 176
 - a weed 37
 - calling 86
 - efficiency 157
 - extensions 147
 - in C++, writing 170
 - linkage 236
- forward
 - declaration of nested `class` 289
 - reference of class member 140
- forwarding function 244
- foundation
 - libraries 191
 - library, horizontal 191
 - library, vertical 191
- fragmentation 211
- Fraser, Sandy 61, 105
- fraud, protection against 116
- free
 - implementations 70
 - storage 32
 - store 56
 - store allocation, prohibit 238
- `free()`, `delete` 57
- friend
 - and member 81
 - encapsulation and 53
 - misuse of 301
- front-end, Cfront compiler 66
- function
 - C with Classes without `virtual` 53
 - and multiple inheritance, `virtual` 265
 - call of undeclared 40
 - cast and pointer to 331
 - `const` member 284
 - conversion 83

- definition 31
- efficiency, `virtual` 49, 75
- forwarding 244
- implementation of member 39
- implementation, `virtual` 75
- `inline` 32
- `inline member` 139
- layout and `virtual` 74
- member 30
- nested 152
- `new()` 31
- only, instantiate used 347
- operator 83
- optimization, `virtual` 239
- pointer to 74
- pointer, `virtual` 74
- polymorphism without `virtual` 49
- pure `virtual` 279
- recompilation, `virtual` 75
- static member 288
- table, `virtual` 74
- template 348
- template and `virtual` 342
- template argument 350
- template overloading 351
- `virtual` 72, 74
- future C++ programming environment 167
- `f(void)`
 - abomination 41
 - and `f()` 41

G

- garbage
 - collection 143, 203
 - collection, automatic 219
 - collection, conservative 217
 - collection, destructor and 222
 - collection, multiple inheritance 271
 - collection, optional 198, 220
 - collection, plug-in 217
 - collection, reasons against 220
 - collection, reasons for 220
 - collection, standard 221
- Gautron, Philippe 344, 400
- GDB_root 184
- general
 - application programming 204
 - rules 110
- general-purpose 125
- language 28, 201
- programming language C++ 105
- generated
 - C 67
 - C code 39
- generic

- programming, C++ and 379
- types, macros for 51
- Gibbons, Bill 139, 155, 294, 373, 400
- global
 - allocation, prohibit 237
 - name 288
 - namespace 399, 412
 - optimization 116
 - scope 412
 - scope name clashes 400
 - static deprecated 419
 - variable 32
- Glockenspiel 164
- library 191
- GNU
 - compiler 166
 - library 191
- goals of C++ 196
- Goldstein, Ted 294, 387, 391
- good
 - extensions, problems with 152
 - languages vs good software 149
- Gorlen, Keith 125, 165, 191, 276, 345
- `grab_x()` 393
- gradual
 - adoption 168
 - approach to C++ 171
 - change 24
- gradually, learning 113, 169
- grammar, ANSI C 69
- granting access 53–54
- graph, conversion 228
- Green Hills 164
- grouping of exceptions 385–386, 396
- guaranteed catching of exceptions 385
- guarantees, run-time 56
- gummy bear 261

H

- `.h` file and `template` 376
- hack
 - macro 401
 - structure tag compatibility 48
- hand optimization 211
- handle 243
- handler, `new` 219
- Hansen, Tony 289, 339, 400
- Hanson, David 253
- Haradhvala, Sam 71, 339
- hard
 - to implement, multiple inheritance 270
 - to use, multiple inheritance 270
- hardware, special 134
- Hartinger, Roland 153, 400
- `hchar` 161

- header
 - file 34, 207
 - files 235
 - precompiled 425
- heap storage 32
- helper class 87
- Hewlett-Packard 71, 128–129
- Softbench 167
- hiding
 - base, derived 417
 - of base member 77
 - of name 77
 - of representation 279
 - of template implementation 366
 - overriding and 76
- hierarchy
 - class 20
 - exceptions and class 387
 - single-rooted 170, 184
- higher-level programming 202
- high-performance computer 176
- hijack names 406
- hijacking of names in template 373
- hindsight, retrospective 196
- histogram 62
- history
 - and philosophy 23
 - revisionist 1
- Holly, Mike 157
- honesty, intellectual 179
- Hopkins, Bill 272
- HOPL-2 paper 2
- horizontal foundation library 191
- Howard, Jim 241
- Hstring 161
- hybrid
 - design 201
 - styles 112
- hype, marketing 63

I

- IBM 128, 392
- EBCDIC 160
- compiler 166
- ideal
 - const 286
 - program development environment 207
- idealism 24
- idealists and experience 23
- ideals
 - for C++ exception handling 385
 - for C++ programming environment 167
 - for namespace design 402
- identity, type 316
- if alternative to `#ifdef` 425

- `#ifdef` 424
- if alternative to 425
- ignoring
 - standards 135
 - warning 42
- impact of change 281
- implementation
 - C with Classes 29
 - Cfront template 365
 - and interface 281
 - and interface, template 379
 - complexity, RTTI 306
 - declaration 35
 - dependencies 138
 - experience, template 339, 368
 - hiding of template 366
 - inheritance private 53
 - language, C++ as an 204
 - multiple inheritance 270
 - of Simula 20
 - of member function 39
 - of namespace 403
 - of new, Cfront 57
 - portable 21
 - problems, template 338
 - simple-minded 36
 - simplicity 111
 - supplier lock-in 138
 - system and language 206
 - virtual function 75
- implementation-dependent behavior
 - 133–134
- implementations
 - free 70
 - language and 127
- implementer
 - and programmer, contract between 133
 - designer and 60
- implicit
 - cast 350
 - conversion 80, 223
 - conversion, `static_cast` and 329
 - instantiation 365
 - int, deprecate 45
 - narrowing conversion 334
 - use in template 370
- importance
 - multiple inheritance 271
 - of placement 215
 - of syntax 48
 - of time order 123
- impression of instability 153
- `#include` 407, 424
- include alternative to 425
- include alternative to `#include` 425
- incomplete type, cast and 329

- increment operator ++ 245
- incremental linking 206
- independent multiple inheritance 260
- indirection 242
- individual, respect for 23
- individuals, learning and 169
- industry
 - software components 191
 - standards 206
- influence
 - on C++, operating system 22
 - operating system 55
- informal standardization 126
- information, compiler has partial 37
- inheritance
 - and template 361
 - constraint through 344
 - multiple 108, 257
 - of operator delete() 213
 - of operator new() 213
 - overloading and 225
 - overuse of 53, 170
 - private, implementation 53
 - public, interface 53
 - timing of multiple 257
- inherited 290
- initial C++ manual, size of 79
- initialization 37
 - Simula 56
 - allocation and 212–213
 - and assignment 22
 - and paging 98
 - and virtual memory 98
 - dynamic 98, 286
 - library 96
 - of cin 96
 - of const 286
 - of cout 96
 - of stdin 96
 - of stdout 96
 - order of 96
 - order of member 275
 - problems with run-time 96
 - resource acquisition 389
 - run-time 96, 286
 - run-time checked 326
 - static 96, 286
- initialize-only style 84, 99
- initializer 96
 - base 275
 - const member 151
 - member 275
 - syntax 380
- inline
 - C++ and 425
 - function 32
 - keyword 33
 - member function 139
 - template and 342
- inlining linker compiler 33
- inquiry, compile-time type 353
- Insinga, Aron 392
- INSPECT, Simula 74, 309
- instability
 - impression of 153
 - standardization as force for 148
- instantiate used function only 347
- instantiate keyword 367
- instantiation
 - #pragma, template 366
 - automatic template 365
 - context of template 369
 - directive, template 367
 - explicit template 366
 - implicit 365
 - late 346, 365
 - manual optimization of 368
 - multiple 367
 - point of 368
- instruction set, standard 135
- int
 - and char 224
 - deprecate implicit 45
 - derivation from 380
- integral
 - conversion, floating to 42
 - promotions 228
- integrated system 112
- intellectual honesty 179
- interface 31
 - cast and 183
 - composition 409
 - const for specifying 90
 - evolution 206
 - foreign language 236
 - implementation and 281
 - inheritance public 53
 - separate 35
 - template implementation and 379
- interfaces 118
 - statically checked 107
- inter-language linkage 236
- international standard 136
- internet, newsgroups email 164
- interrupts 393
- intersect rule 229
- intersect Shape 297
- Interviews 301
 - library 165, 191
- inventing terminology 31
- I/O
 - Ada 185

- C 96
- C++ stream 185
- Stream 93, 96
- efficient 185
- example of object 320
- extensible 185
- library, stream 185
- object 276
- of extended character sets 161
- printf, C 185
- terse 185
- type-safe 185
- io 322
- iocircle 322
- io_counter 97
- io_obj 321
- iostream 260
- isKindof, Smalltalk 325
- ISO
 - ANSI and 136
 - C++ standardization 129
 - WG-21 136
 - standard library 194
- ISO-646-1983 ASCII ANSI3.4-1968 158
- issues, system-related 206
- iterators, standard 194

J

- Jchar 161
- Johnson, Steve 22, 35, 62, 68, 70, 101, 105, 124
- jokes, puns and 24
- JOOP 166
- Jordan, David 361
- Journal, The, C++ 165
- journals and books 165
- Jstring 161
- Juhl, Peter 400, 403

K

- Kallman, Dave 70
- Keffer, Thomas 191
- Kelley, Bob 125
- Kernighan, Brian 4, 22, 70, 91, 105, 125, 127, 245, 253
- keyword
 - and 160
 - and_eq 160
 - argument 153
 - argument workaround 156
 - bitand 160
 - bitor 160
 - compl 160
 - inline 33

- instantiate 367
- not 160
- not_eq 160
- or 160
- or_eq 160
- postfix 246
- prefix 246
- specialise 375
- specialize 375
- template 341
- xor 160
- xor_eq 160
- keywords, new 152
- kinds of rules 110
- Knuttila, Kim 392
- Koenig, Andrew 4, 46, 71, 81, 101, 125, 127, 131, 134, 139, 145, 165–166, 187, 227, 229, 239–240, 251, 253, 255, 263, 303, 322, 327, 339, 344, 348, 352, 368, 383, 387, 391, 400
- Kroghdahl, Stein 258
- Krohn, Eric 400

L

- Lachner, Archie 4, 125, 212
- lack of template 184
- LALR(1) parser, YACC 68
- language
 - C++ as a target 204
 - C++ as an implementation 204
 - C++, general-purpose programming 105
 - English as a common 161
 - and environment, separation between 197
 - and implementations 127
 - and system 37
- chart 5
- choosing a programming 175
- company 178
- comparison 270
- complexity 199
- constructs, concepts and 19
- cult 229
- definition, C++ 5
- design 114
- design and engineering 104
- design and features 104
- design and mathematics 104
- design and philosophy 104
- design and sociology 104
- design committee for a new 148
- design, motivation for 105
- design principles 109
- design rules 109
- design, tools for 103
- elegant 179
- evaluating a programming 175

- feature, library vs. 181
- features and techniques 170
- general-purpose 28, 201
- implementation, system and 206
- interface, foreign 236
- lawyers 5
- programming 112, 114
- purposes of programming 44
- specialized 201, 204
- support, basic 194
- support for library 183
- wars 179
- wars and bigotry 299
- what is a programming 7
- languages
 - C++ and other 197
 - comparing programming 5
 - exceptions and other 394
 - vs good software, good 149
- language-technical
 - details 3
 - rules 117
- large
 - program, naming in a 406
 - system 116
 - system and exceptions 392
- late instantiation 346, 365
- later users 174
- lattice, conversion 228
- lawyers
 - language 5
 - users vs. 137
- layout
 - C with Classes object 38
 - Cfront virtual table 320
 - RTTI 320
 - and virtual function 74
 - compatibility, C with Classes 28
 - derived class 52
 - exceptions and 397
 - multiple inheritance 264
 - object 304
 - standard virtual table 135
 - template and 342
 - virtual base 266
- Lea, Doug 191, 298, 344
- learn C++, time needed to 172
- learning
 - C 169
 - C and C++ 169
 - CLOS 169
 - C++ 168, 172
 - C++ and Smalltalk 170
 - C++ from ARM 127
 - Eiffel 169
 - ML 169
 - OODBMS 169
 - OOP 170
 - Smalltalk 169
 - academic pursuit of 70
 - and individuals 169
 - assembler 169
 - gradually 113, 169
 - object-oriented programming 170
 - on the job 168
 - verification 169
- legal
 - program 133
 - source text 133
- Lenkov, Dmitry 129, 306, 384, 391
- levels of abstraction, exceptions and 392
- libraries
 - C++ 65
 - aims of early 184
 - and static type checking 183
 - and type system 183
 - concurrency support 188
 - foundation 191
 - many 167
 - numeric 192
 - persistence 192
 - specialized 193
 - support for 108
 - use of multiple 183
- library
 - ANSI standard 194
 - AT&T 184
 - Bell Labs 184
 - Booch components 192
 - GNU 191
 - Glockenspiel 191
 - ISO standard 194
 - Interviews 165, 191
 - NIH 165, 191
 - RHALE++ 192
 - RTTI and 183
 - Rogue Wave 191
 - Smalltalk-inspired 184
 - Standard Components 184
 - USL 184, 191
 - abstract class and 183
 - complex 66
 - complexity 199
 - constructor and 183
 - dependencies on 134
 - design 182
 - design, template and 378
 - design tradeoffs 182
 - exception and 183
 - horizontal foundation 191
 - initialization 96
 - language support for 183

- missing, standard 124
 - multiple inheritance and 184, 258
 - namespace and 183
 - namespaces in standard 400
 - naming in standard 316
 - providers and namespaces 403
 - run-time type information and 183
 - standard 407
 - standard numeric 194
 - stream I/O 185
 - support 306
 - task 27, 184, 188
 - vertical foundation 191
 - vs. language feature 181
 - lifetime of temporaries 143
 - limit on number of characters 69
 - limits
 - to perfection 108
 - translation 136
 - Lindsey, Charles 100
 - linear notation for declarators 46
 - link compatibility 120
 - link 50, 62, 259
 - linkage
 - C 232
 - C and C++ 233
 - Fortran 236
 - PL/I 236
 - Pascal 236
 - TSL see type-safe
 - and overloading 233
 - experience with type-safe 234
 - inter-language 236
 - model 34
 - name 233
 - overloading and 232
 - pointer to function 236
 - problem, type-safe 236
 - problems 69
 - syntax, type-safe 399
 - to other languages 34
 - type-safe 34, 232
 - linkage-specifier* 234
 - linker 120, 232
 - compiler, inlining 33
 - names 403
 - problem 98
 - problems with 303
 - linking
 - dynamic 206
 - incremental 206
 - link-time
 - and template 366
 - overhead 233
 - Linton, Mark 191, 301
 - lip service to minimalism 147
 - Lippman, Stan 71, 125, 303, 339
 - Liskov, Barbara 53, 301
 - Lisp 37, 67, 175
 - model 197
 - list 62
 - lists, standard 194
 - literal
 - binary 152
 - type of char 225
 - literature 23
 - on programming languages 105
 - L.M.Ericsson 392
 - lobbying for long names 70
 - local
 - default 101
 - static arrays, ANSI C 128
 - variable 32
 - variables, true 198
 - locality 117
 - nested class 118
 - Locanthi, Bart 105
 - lock-in, implementation supplier 138
 - locking 190
 - in users 167
 - logic, theory and 23
 - Logothetis, George 71
 - long
 - names, lobbying for 70
 - namespace name 408
 - lookup
 - ANSI/ISO name 141
 - ARM name 139
 - name 138
 - template name 371
 - LotterySimulation 273
 - low-level
 - coding 338
 - features, C with Classes 29
 - programming 202
 - programming rules 120
 - Lucid Energize 167
 - lvalue vs. rvalue 87
- ## M
- Macintosh, Apple 165
 - macro
 - Cpp 118, 424
 - contain using 50
 - container class and 337
 - hack 401
 - template style 339
 - macros for generic types 51
 - maintainable program 406
 - maintaining coherence 153
 - major

- C++ projects 174
- and minor extension 305
- and minor feature 305
- `malloc()`, new and 56, 91
- management 277
 - development 124
 - of extensions proposals 147
- manual
 - ARM reference 126
 - C and C++ reference 170
 - C++ reference 60
 - control 121
 - optimization of instantiation 368
 - size of initial C++ 79
- many
 - environments 167
 - libraries 167
- map, standard 194
- Map 322
- Marian 405
- marketing
 - AT&T C++ 177
 - hype 63
- marketing, lack of 164
- Mary 22
- Mascitti, Rick 64
- match, better 228
- matching
 - overload 228
 - rules, argument 228
- mathematics
 - as tool for problem solving 23
 - language design and 104
- Matherr 387
- Matrix, efficient 252
- maturity
 - personal 137
 - technical 137
- Mayka, Larry 125
- McCluskey, Glen 339
- McIlroy, Doug 4, 41, 46, 62, 78, 85, 91, 102, 105, 125, 127, 141, 186, 227, 229, 253, 274, 304, 384, 400
- member
 - access control and `static` 288
 - `class` 289
 - forward reference of `class` 140
 - `friend` and 81
 - function 30
 - function, `const` 284
 - function, implementation of 39
 - function, `inline` 139
 - function, method see
 - function, `static` 288
 - hiding of base 77
 - initialization, order of 275
 - initializer 275
 - initializer, `const` 151
 - order dependencies, `class` 138
 - pointer to 303
 - pointer to data 304
 - protected 301
 - template 364
 - template example 361–362
 - template, `virtual` 364
 - members, committee 137
 - memberwise copy 239
 - memory
 - arena 215
 - exhaustion and exceptions 390
 - exhaustion, control of 218
 - initialization and `virtual` 98
 - management 91, 203, 211
 - management, Release 2.0 and 212
 - management, fine-grain control of 211
 - management, tuning of 211
 - raw 213
 - read-only 286
 - special 211
 - use, Cfront 66
 - Memphis 115
 - Mentor Graphics 212, 241
 - merging of `class` hierarchies 273
 - Mesa 44
 - Cedar 392
 - meta-object 324
 - method
 - combination 268
 - see member function
 - unconstrained 325
 - methods, formal 103
 - micro C++ 188
 - Microsoft 129
 - compiler 166
 - migration path 201
 - Miller, William (Mike) 165, 384, 391
 - minimal run-time support 21
 - minimalism, lip service to 147
 - minimize surprises 227
 - Minner, Richard 400
 - missing, standard library 124
 - mistake, worst 200
 - misuse
 - of RTTI 313
 - of `friend` 301
 - of `typeid` 308
 - use and 115
 - Mitchell, Jim 392
 - mixed system 204
 - mixed-mode arithmetic 80, 223
 - mixin 261
 - ML 103, 105, 107, 387

- learning 169
- model
 - Lisp 197
 - Simula 197
 - Smalltalk 197
- linkage 34
- of protection, C++ 22
- protection 53
- template compilation 376
- Modula-2 35, 44, 103, 175, 204
 - extensions 147
- Modula-2+ 397
- Modula-3 6, 67, 176–177, 202
 - variables 198
- modularity 102
- module structure 27
- monitor 56
- Moo, Barbara 71, 125, 277
- motivation
 - for change 110
 - for language design 105
- MPW, Apple Mac 167
- multi-argument overloading 228
- multi-level
 - error handling 385
 - propagation of exceptions 394
- multi-method 325
 - workaround for 299
- multi-methods 297
- multi-paradigm 106
- multiple
 - inheritance 108, 257
 - inheritance C++ 271
 - inheritance and Simula 258
 - inheritance and Smalltalk 270
 - inheritance and ambiguity 258
 - inheritance and exception handling 387
 - inheritance and library 184, 258
 - inheritance bugs 270
 - inheritance complexity 270
 - inheritance controversy 269
 - inheritance delegation 270
 - inheritance garbage collection 271
 - inheritance hard to implement 270
 - inheritance hard to use 270
 - inheritance implementation 270
 - inheritance importance 271
 - inheritance, independent 260
 - inheritance layout 264
 - inheritance, name clash and 273
 - inheritance overhead 270
 - inheritance, single and 258
 - inheritance, static checking of 263
 - inheritance, timing of 257
 - inheritance tools 271
 - inheritance, use of 271, 322

- inheritance, virtual function and 265
- instantiation 367
- libraries, use of 183
- multi-processor system 188
- multi-threading 188
- Murray Hill 27
- Murray, Rob 165, 339
- mutable 287
- mutating, extending vs. 85
- Myers, Nathan 80

N

- name
 - C++ 64
 - binding rule 368
 - clash and multiple inheritance 273
 - clashes, global scope 400
 - encoding, namespaces and 420
 - encoding, standard 135
 - equivalence 35
 - global 288
 - hiding of 77
 - linkage 233
 - long namespace 408
 - lookup 138
 - lookup, ANSI/ISO 141
 - lookup, ARM 139
 - lookup, template 371
 - mangling 234
 - mangling, namespaces and 420
 - prefixes 401
 - short namespace 408
 - space pollution 288
- name() 317
- named argument 153
- names
 - access control of 54
 - argument 155
 - hijack 406
 - in template, hijacking of 373
 - linker 403
 - lobbying for long 70
 - source code 403
- namespace
 - as template argument 343
 - problem 400
- namespace 289, 404
 - C++ and 425
 - alias 404, 408
 - and class 417
 - and library 183
 - class as 401
 - design, aims of 402
 - design, ideals for 402
 - design, non-aims of 403

- global 399, 412
 - implementation of 403
 - name, long 408
 - name, short 408
 - syntax 405
 - template 360
 - unnamed 419
 - using 407
 - namespaces
 - and C compatibility 420
 - and name encoding 420
 - and name mangling 420
 - and releases 409
 - and versioning 409
 - in standard library 400
 - library providers and 403
 - nested 415
 - open 416
 - overloading and 414
 - transition to 407
 - naming
 - in a large program 406
 - in standard library 316
 - narrow() 335
 - narrowing
 - conversion 41, 227
 - conversion, implicit 334
 - national character sets 158
 - natural conversion 227
 - nature of programming 111
 - NCEG 157
 - NDEBUG 397
 - Neath, Martin 339
 - necessary features 149
 - need for extensions 129
 - needed for standard, time 129
 - Needham, Roger 22
 - nested
 - class 102, 128, 289
 - class, forward declaration of 289
 - class, locality 118
 - default 101
 - function 152
 - namespaces 415
 - template 364
 - network_file_err 387
 - networking 188
 - new
 - cast notation 327
 - cast, transition to 335
 - cast, using 335
 - features 269
 - keywords 152
 - language, design committee for a 148
 - new
 - Ada 365
 - Cfront implementation of 57
 - allocation and constructor 56
 - and allocation 212
 - and constructor 213
 - and malloc() 56, 91
 - handler 219
 - operator 32, 212
 - performance 91
 - new()
 - constructor 56
 - function 31
 - new[], operator 214
 - Newcastle 22
 - new_handler 91
 - newsgroups email internet 164
 - new-style cast 327
 - Ngo, Thomas 287
 - Nicholas 404
 - NIH library 165, 191
 - noalias 157
 - No_free_store 238
 - non-aims of namespace design 403
 - non-pointers, pointers and 31
 - non-type template argument 342
 - not keyword 160
 - notation
 - constructor 94
 - constructor call 335
 - for declarators, linear 46
 - new cast 327
 - old-style cast 333
 - notational
 - convenience 241
 - convenience, operators and 223
 - not_eq keyword 160
 - null pointer 0 229
 - NULL 0 230
 - Num 243
 - number
 - of C++ compilers sold 164
 - of C++ users 163
 - of arguments, average 156
 - of characters, limit on 69
 - numeric
 - computing 203
 - libraries 192
 - library, standard 194
 - numerical
 - C 157
 - computing 158
 - Nygaard, Kristen 4, 19
- ## O
- Oasys 164
 - Object Pascal 291

- object
 - I/O 276
 - I/O, example of 320
 - copy of 293
 - layout 304
 - layout, C with Classes 38
 - partially constructed 389
 - representation of 31, 279
 - stack 92
 - static 92
- Object 314
- ObjectCenter, Centerline 167
- Objective C 175, 257
- object-oriented
 - databases 192
 - design 114, 172
 - programming 73, 172
 - programming, C++ and 197
 - programming, learning 170
- ObjectWorks, ParcPlace 167
- old code, exceptions and 394
- old-style cast notation 333
- OLE2 206
- ON-conditions, PL/I 392
- one-definition
 - rule 36
 - rule and `template` 369
- one-pass
 - analysis 140
 - compilation 139
- Onfreestore 237
- onion design 323
- OOD 114, 172
- OODBMS 361
 - learning 169
- OOP 73, 172
 - learning 170
- OOPSLA 299
- open namespaces 416
- operating
 - system influence 55
 - system influence on C++ 22
 - system, standard 135
- operations, unsafe 222
- operator
 - , 247
 - `**`, exponentiation 247
 - `*^`, exponentiation 249
 - `++`, increment 245
 - `--`, decrement 245
 - `->` 241
 - `->*` 247
 - `.` 242
 - `.*` 247
 - `<<` 186
 - `<< output` 186
 - `=` 239
 - composite 251
 - default 237
 - `delete` 32, 212, 216
 - `delete[]` 214
 - equivalence 242
 - function 83
 - `new` 32, 212
 - `new[]` 214
 - overloading 78
 - precedence of user-defined 251
 - `private` 237
 - prohibit 237
 - type inquiry 324
 - type relation 325
 - user-defined 250
- operator
 - `delete()` 194
 - `delete()`, inheritance of 213
 - `new()` 194
 - `new()`, inheritance of 213
 - `new()` return value 218
- operators
 - Algol68 250
 - Simula 86
 - and notational convenience 223
- optimization
 - global 116
 - hand 211
 - of instantiation, manual 368
 - of virtual call 116
 - return value 302
 - return value 84
 - virtual function 239
 - virtual table 302
- optional garbage collection 198, 220
- or keyword 160
- order
 - dependence 118
 - dependencies, `class` member 138
 - dependencies, workarounds for 97
 - dependency 232
 - dependency and overloading 226
 - of initialization 96
 - of member initialization 275
- ordinary base class 258
- Oregon
 - Software 164
 - Software compiler 166
- `or_eq` keyword 160
- organization
 - flexibility efficiency 1
 - program 28, 115
- original standards schedule 129
- O'Riordan, Martin 254, 274, 288, 294, 330, 368, 391–392, 400

- orthogonality 113
 - cult of 104
- OS/2 165, 392
- other
 - languages, exceptions and 394
 - languages, linkage to 34
- output operator, << 186
- over-constrained argument 344
- overflow and exceptions 394
- overhead
 - Cfront compile-time 67
 - compile-time 233
 - link-time 233
 - multiple inheritance 270
 - rule, zero 121
 - run-time 233
- overload
 - base and derived 417
 - based on `enum` 254
 - based on enumeration 254
 - matching 228
 - resolution 224
 - resolution, fine-grain 224
- `overload` declaration 231
- overloading 78, 224
 - Algol68 references 44
 - Release 2.0 and 224
 - and `char` 224
 - and `class` hierarchy 226
 - and `const` 225
 - and conversion 223
 - and default arguments 59
 - and efficiency 78
 - and `float` 224
 - and inheritance 225
 - and linkage 232
 - and namespaces 414
 - and transition 414
 - base and derived 77
 - efficiency and 84
 - fear of 231
 - function `template` 351
 - linkage and 233
 - multi-argument 228
 - of assignment 58
 - operator 78
 - order dependency and 226
 - reasons against 78
 - reasons for 78
- overriding
 - and hiding 76
 - relaxation of 293
- overstated weaknesses 153
- overuse
 - of `cast` 170
 - of inheritance 53, 170

P

- paging, initialization and 98
- paper
 - HOPL-2 2
 - design 60
- parameterized
 - type 107, 341
 - type `template` 338
- ParcPlace ObjectWorks 167
- parser
 - YACC LALR(1) 68
 - recursive descent 68
- parsing C++ 68
- partial information, compiler has 37
- partially constructed object 389
- Pascal 135, 204
 - and Simula 20
 - extensions 147
 - linkage 236
 - toy 44
 - variables 198
- pass-by-reference 86
- pass-by-value 85
- paternalism 106
- path, migration 201
- pathological examples 141
- PC, port of Cfront to 66
- PCC C compiler 69
- PCs personal computers 65
- pecompiled header 425
- Pennello, Tom 69, 139, 145
- people
 - forcing 23, 171
 - who contributed to C++ 2
- per-class allocator 91, 211
- perfection
 - and feedback 111
 - limits to 108
- performance 211
 - new 91
 - of Simula, run-time 20, 32
 - startup 98
- persistence libraries 192
- persistent symbol table 376
- personal
 - computers, PCs 65
 - maturity 137
- personal, preferences 106
- Philip, Pat 125
- philosophers 301, 392
- philosophy
 - facts vs. 3
 - history and 23
 - language design and 104
- PL360 22
- placement 211, 214

- importance of 215
- plan, C++ development 124
- PL/I 227
 - ON-conditions 392
 - linkage 236
- plug-in garbage collection 217
- Plum, Tom 103, 131
- point of instantiation 368
- pointer
 - 0, null 229
 - copy of 240
 - replace ordinary 245
 - semantics 198
 - smart 241, 361
 - syntax 304
 - this 39
 - to data member 304
 - to function 74
 - to function, cast and 331
 - to function linkage 236
 - to member 303
- pointers
 - and non-pointers 31
 - and references 86
 - restricted 157
- points of destruction of temporary,
 - alternative 145
- pollution, name space 288
- polymorphic type 312
- polymorphism without virtual function 49
- population, user 61
- port of Cfront to PC 66
- portability 125
 - extreme 65
- portable
 - C is 43
 - implementation 21
- porting problems 235
- positive rules 122
- postfix
 - ++, prefix and 245
 - , prefix and 245
- postfix keyword 246
- potential error 113
- Powell, Mike 395
- #pragma 424
 - template instantiation 366
- pragmatic approach 60
- precedence of user-defined operator 251
- prefix
 - Simula class 49
 - and postfix ++ 245
 - and postfix -- 245
- prefix keyword 246
- prefixes, name 401
- prehistory of C++ 19
- preprocessor
 - C 68
 - Cpp, C 119, 423
 - Cpre C with Classes 27
- Presotto, Dave 185
- principles
 - design from first 44
 - language design 109
- printf, C I/O 185
- private, default 101
- private 301
 - base, cast to 328
 - implementation inheritance 53
- operator 237
- public 53
- privileges 407
- problem
 - RTTI 307
 - linker 98
 - namespace 400
 - solving, mathematics as tool for 23
 - type-safe linkage 236
 - with C 374
 - with alias 275
 - with synonym 275
 - with template syntax, >> 356
- problems
 - archetypical 62
 - deallocation 216
 - design and 105
 - linkage 69
 - porting 235
 - real 110
 - with cast 327
 - with good extensions 152
 - with linker 303
 - with run-time initialization 96
- process, feature acceptance 61
- productive, staying 172
- program
 - development environment, ideal 207
 - legal 133
 - maintainable 406
 - naming in a large 406
 - organization 28, 115
 - reliable 406
 - startup 98
- programmer
 - bored 388
 - contract between implementer and 133
 - trust the 116, 316
- programmers 113
- training of 111
- programming
 - C++ and generic 379

- C++ and object-oriented 197
 - environment, C++ 167
 - environment, Cpp and 424
 - environment, future C++ 167
 - environment, ideals for C++ 167
 - general application 204
 - higher-level 202
 - language 112, 114
 - language, choosing a 175
 - language, evaluating a 175
 - language, purposes of 44
 - language, what is a 7
 - languages, comparing 5
 - languages, literature on 105
 - languages, systems 22
 - learning object-oriented 170
 - low-level 202
 - nature of 111
 - object-oriented 73, 172
 - rules, low-level 120
 - prohibit
 - auto allocation 237
 - copy 237
 - derivation 238
 - free store allocation 238
 - global allocation 237
 - operator 237
 - static allocation 237
 - prohibition 122
 - projects, major C++ 174
 - promises 62
 - promotions, integral 228
 - propagation of exceptions, multi-level 394
 - proposal
 - accepting a 147
 - rejecting a 147
 - proposals
 - accepted 151
 - rejected 151
 - Prosser, Dave 125, 253
 - protected 301
 - member 301
 - protection
 - C++ model of 22
 - Smalltalk 301
 - against accident 116
 - against fraud 116
 - barrier, crossing 32
 - class unit of 54
 - compile-time 54
 - model 53
 - prototype 40
 - providers and namespaces, library 403
 - Ptr 361
 - Ptr 242, 361
 - public 301
 - interface inheritance 53
 - private 53
 - public: 31
 - puns and jokes 24
 - pure virtual function 279
 - purpose
 - of ARM 127
 - of C++ 105, 163
 - purposes of programming language 44
 - pursuit of learning, academic 70
 - pvector 347
- ## Q
- qualification 95
 - ::, explicit 412
 - explicit 406
 - queue 62
 - Quinn, Peggy 125
- ## R
- raise 386
 - Randell, Brian 22
 - rate of change 24
 - raw memory 213
 - readability, constraint and 344
 - read-only memory 286
 - readonly 89
 - real problems 110
 - reality, votes and 90
 - real-time, dynamic allocation and 211
 - reasons
 - against garbage collection 220
 - against overloading 78
 - for garbage collection 220
 - for overloading 78
 - recompilation 31, 281
 - exceptions and 396
 - virtual function 75
 - reconsideration rule 142
 - recursive descent parser 68
 - redefinition rule 139
 - redundancy in syntax 386
 - reference 85
 - binding of 86
 - manual, ARM 126
 - manual, C and C++ 170
 - manual, C++ 60
 - return type 87
 - semantics 198
 - this 39
 - references
 - Algol68 86
 - const 86

- overloading, Algol68 44
- pointers and 86
- smart 242
- RefNum 243
- reinterpret_cast 331
- and static_cast 331
- rejected proposals 151
- rejecting a proposal 147
- relation operator, type 325
- relationships between templates 360
- relaxation
 - of argument rules 295
 - of overriding 293
- Release
 - 1.0 71
 - 1.0 deficiencies, C++ 107
 - 1.0 features, C++ 71
 - 1.1 71
 - 1.2 71, 302–303
 - 1.3 124
 - 2.0 187
 - 2.0 71, 124, 257, 288, 320
 - 2.0 = 82
 - 2.0 and memory management 212
 - 2.0 and overloading 224
 - 2.0 events 124
 - 2.0 features 126
 - 2.0 timing 124
 - 2.1 71, 128, 302
 - 3.0 71
 - E 70
- release1 409
- release2 410
- releases
 - namespaces and 409
 - to universities 70
- reliability, exception handling and 384
- reliable program 406
- religious conversions 24
- removal of restrictions 125
- removing features 269
- renaming 273, 403
- reordering rule 142
- replace ordinary pointer 245
- replicated virtual function table 302
- replication, code 346
- repository, template 376
- representation
 - hiding of 279
 - of object 31, 279
- resolution, overload 224
- resource
 - acquisition initialization 389
 - management 215
 - management, exceptions and 388
 - requirements, Cfront 65
- respect
 - for facts 179
 - for individual 23
- restrict 157
- restricted pointers 157
- restriction, specialization 375
- restrictions, removal of 125
- resumption
 - arguments for 391
 - semantics 390
 - termination vs. 390
 - workaround for 393
- retrospective hindsight 196
- return value optimization 84
- return
 - type 293
 - type, reference 87
 - value, operator new() 218
 - value optimization 302
- return() and :after 57, 268
- reviewed facts 2
- revisionist history 1
- revolutions 24
- rewrite rule 139
- RHALE++ library 192
- Richards, Martin 64
- Ritchie, Dennis 4, 22, 35, 41, 62, 64, 89–90, 105, 157, 163, 253
- Rogue Wave library 191
- ROM 286
- root
 - class 261
 - class, universal 184
- Rose, Leonie 66
- rose
 - Algol68 a 37
 - Simula a 37
- Rosler, Larry 64, 90, 103, 105, 128
- Rowe, Keith 400
- RTTI 74, 306
 - and cast 309
 - and library 183
 - and standardization 315
 - design, alternative 323
 - example of 320
 - implementation complexity 306
 - layout 320
 - misuse of 313
 - problem 307
 - use of 311, 313
- rule
 - intersect 229
 - name binding 368
 - one-definition 36
 - reconsideration 142
 - redefinition 139

- reordering 142
- rewrite 139
- specialization 375
- two week 274
- zero overhead 121
- rules
 - and experience 109
 - argument matching 228
 - design support 114
 - general 110
 - kinds of 110
 - language design 109
 - language-technical 117
 - low-level programming 120
 - positive 122
 - sociological 110
 - voting 136
- run-time
 - argument check 296
 - check 42
 - checked initialization 326
 - checks 29
 - efficiency 32, 179, 324
 - guarantees 56
 - initialization 96, 286
 - initialization, problems with 96
 - overhead 233
 - performance of Simula 20, 32
 - support, C with Classes 49
 - support, minimal 21
 - type information 266, 306
 - type information and library 183
 - type inquiry 74
- rvalue, lvalue vs. 87

S

- safe
 - cast 309
 - conversion 227
- safety 126
 - and exceptions, type 385
 - and type safety 93
 - compatibility vs. 334
 - convenience vs. 411
- Sakkinen, Markku 269
- Santa Fe USENIX workshop 165
- schedule, original standards 129
- Schwarz, Jerry 97, 105, 125, 187, 286–287, 327, 400
- scientific computing 158, 176, 203
- scope
 - C++ and 424
 - global 412
 - name clashes, global 400
- scopes, semi-transparent 54
- Scott, Roger 125
- Season 254
- self, Smalltalk 39
- self-assignment 59
- self-sufficiency, user 37
- semantics
 - C 103
 - pointer 198
 - reference 198
 - resumption 390
 - termination 390
 - value 198
- semantics-oriented tools 207
- semi-transparent scopes 54
- separate
 - compilation 21, 34
 - from C, C++ 65
 - interface 35
- separation between language and environment 197
- sequence of events 123
- serving current users, C++ 110
- Set of Shape* 360
- set 279
- Sethi, Ravi 46, 103, 240
- set_new_handler 219
- set_new_handler() 194
- shallow copy 240
- Shape, intersect 297
- shape 72, 278, 320
- Shape*, Set of 360
- Shapiro, Jonathan 4, 46, 61, 66, 78, 83, 87, 105, 125, 129, 134, 184, 206, 227, 229, 251, 288, 304, 339, 362, 368
- short namespace name 408
- sibling class 261
- siblings, communication between 261
- signal 386
- signals 393
- SIMLIB 193
- simple
 - and efficient String 185
 - definition of 403
- simple-minded implementation 36
- simplicity
 - and complexity 112
 - and elegance 229
 - flexibility vs. 224
 - implementation 111
- Simula 19, 31, 40, 62, 105
 - C and 1
 - INSPECT 74, 309
 - Pascal and 20
 - THIS 39
 - Users' group, ASU 106
 - a rose 37

- and C++ 106
- class 6, 44
- class prefix 49
- implementation of 20
- initialization 56
- model 197
- multiple inheritance and 258
- operators 86
- run-time performance of 20, 32
- style class 21
- variables 198
- virtual 73
- simulator, distributed system 19
- Singh, Livleen 165
- single
 - and multiple inheritance 258
 - precision floating point 224
- single-rooted hierarchy 170, 184
- size
 - of Cfront 66
 - of initial C++ manual 79
- sizet 213
- Skaller, John 400
- skeptics, evidence and 24
- slicing 240
- slist_set 280
- Smalltalk 37, 44, 67, 175, 325
 - and C++ 107
 - complexity 199
 - in C++, writing 170
 - isKindOf 325
 - learning 169
 - learning C++ and 170
 - model 197
 - multiple inheritance and 270
 - protection 301
 - self 39
 - style 170, 314
 - subclass 49
 - super 291
 - techniques, undesirable 378
 - variables 198
- Smalltalk-inspired library 184
- Smalltalk-style container 339
- smart
 - pointer 241, 361
 - references 242
- Snyder, Alan 53, 294
- sociological rules 110
- sociology, language design and 104
- Softbench, Hewlett-Packard 167
- software
 - components industry 191
 - good languages vs good 149
- SOM 206
- sort() 348
- sorting 358
- source
 - C++ 207
 - code names 403
 - control and template 366
 - file 207
 - file and template 376
 - text, legal 133
- special
 - address 214
 - hardware 134
 - memory 211
- specialise keyword 375
- specialization
 - declaration 375
 - of template 373
 - restriction 375
 - rule 375
- specialize keyword 375
- specialized
 - language 201, 204
 - libraries 193
 - tools 204
- specification, exception 396
- Spicer, John 350
- stability
 - and extension 125
 - and standards 205
- stack
 - object 92
 - storage 32
- stack 30
- Standard Components library 184
- standard
 - ANSI C++ 128
 - American national 136
 - algorithms 194
 - application environment 135
 - as a contract 133
 - associative array 194
 - calling sequences 135
 - containers 194
 - conversion 228
 - debugger 135
 - extensions and 134
 - garbage collection 221
 - instruction set 135
 - international 136
 - iterators 194
 - library 407
 - library, ANSI 194
 - library, ISO 194
 - library missing 124
 - library, namespaces in 400
 - library, naming in 316
 - lists 194

- map 194
- name encoding 135
- numeric library 194
- operating system 135
- time needed for 129
- timeliness of 135
- vector 194
- virtual table layout 135
- what is a 133
- standardization 124
 - ISO C++ 129
 - RTTI and 315
 - aims for 130
 - aims of 135
 - as force for instability 148
 - informal 126
 - start of C++ 128
 - status of 150
- standards
 - committee 128
 - ignoring 135
 - industry 206
 - process, disruption of 137
 - schedule, original 129
 - stability and 205
- Stark, Kathy 125
- start of C++ standardization 128
- startup
 - performance 98
 - program 98
- static
 - and dynamic type checking 107
 - checking of exceptions 395
 - checking of multiple inheritance 263
 - initialization 96, 286
 - object 92
 - storage 32
 - store 56
 - type checking 31, 106, 324
 - type checking, C with Classes 40
 - type checking, C++ 197
 - type checking and design 107
 - type checking, libraries and 183
 - type system 117
- static
 - allocation, prohibit 237
 - deprecated, global 419
 - member, access control and 288
 - member function 288
 - statically checked interfaces 107
- static_cast 329
 - and const 329
 - and dynamic_cast 330
 - and implicit conversion 329
 - fooling 330
 - reinterpret_cast and 331
- status
 - of standardization 150
 - quo, fatalism 24
- staying productive 172
- stdin, initialization of 96
- stdio.h 407
- stdout, initialization of 96
- Stepanov, Alex 194, 339, 348, 354, 379
- STL 194
- storage
 - automatic 32
 - class 32
 - dynamic 32
 - free 32
 - heap 32
 - stack 32
 - static 32
- store
 - dynamic 56
 - free 56
 - static 56
- Strachey, Christopher 64
- Stream I/O 93, 96
- stream
 - I/O, C++ 185
 - I/O library 185
- strength, C++ 208
- String 81, 83, 87
 - fancy 185
 - simple and efficient 185
- string 194
 - class 66
- strong type checking, C with Classes 40
- strtok() 225
- struct and class 76
- structural equivalence 35
- structure
 - of Cfront 66
 - tag 47
 - tag compatibility hack 48
- style, initialize-only 84, 99
- styles
 - hybrid 112
 - of design 24
- subclass
 - Smalltalk 49
 - superclass and 74
- subset of C++, better C 171
- success of C with Classes 64
- suggested extensions 150
- sum() 368
- super, Smalltalk 291
- superclass and subclass 74
- support
 - C with Classes 29
 - concurrency 188

- debugging 42
- for libraries 108
- for library, language 183
- library 306
- rules, design 114
- supporting C with Classes 63
- surprises, minimize 227
- switch
 - on type 313
 - on type field 72
- symbol
 - table, persistent 376
 - tables, Cfront 66
- synonym 403
- problem with 275
- syntax 119
 - <...> 341, 355
 - =0 281
 - >> problem with template 356
- C declaration 45
- alternative template 355
- cast 309
- exception handling 385
- importance of 48
- initializer 380
- namespace 405
- of dynamic_cast 309
- pointer 304
- redundancy in 386
- template 350
- type-safe linkage 399
- type-system, C++ 208
- syntax-oriented tools 207
- system
 - and exceptions, large 392
 - and language implementation 206
 - design of fault-tolerant 385
 - embedded 203
 - integrated 112
 - language and 37
 - large 116
 - libraries and type 183
 - mixed 204
 - multi-processor 188
 - system-related issues 206
- systems programming languages 22

T

- tag, structure 47
- target language, C++ as a 204
- task
 - example 188
 - library 27, 184, 188
- task 62, 258–259, 288
- TauMetric compiler 166

- teaching
 - C++ 168
 - and design 168
 - design and 119
- technical maturity 137
- technique, education and 205
- techniques
 - language features and 170
 - template composition 356
 - undesirable C 378
 - undesirable Smalltalk 378
- template
 - Container 315
 - conditionals in 353
 - use of 322
- template
 - .c file and 376
 - .h file and 376
 - Cfront 339
 - absence of 378
 - ambiguity in 371
 - and abstract class 379
 - and built-in types 380
 - and class 360
 - and constructor 379
 - and exceptions 379
 - and inline 342
 - and layout 342
 - and library design 378
 - and virtual function 342
 - argument, constraint on 343
 - argument, deducing 349
 - argument, depend on 370
 - argument, explicit 350
 - argument, function 350
 - argument, namespace as 343
 - argument, non-type 342
 - argument, template as 343
 - array 342
 - as template argument 343
 - class 341
 - compilation model 376
 - compile-time and 366
 - complex 362
 - composition techniques 356
 - container class and 337
 - container without 50
 - conversion and 362
 - definition, context of 369
 - definition, finding 376
 - design, cleanup of 340
 - design too cautious 340
 - double dispatch 364
 - error checking of 369
 - example, member 361–362
 - function 348

- hijacking of names in 373
- implementation, Cfront 365
- implementation and interface 379
- implementation experience 339, 368
- implementation, hiding of 366
- implementation problems 338
- implicit use in 370
- inheritance and 361
- instantiation `#pragma` 366
- instantiation, automatic 365
- instantiation, context of 369
- instantiation directive 367
- instantiation, explicit 366
- keyword 341
- lack of 184
- link-time and 366
- member 364
- name lookup 371
- `namespace` 360
- nested 364
- one-definition rule and 369
- overloading, function 351
- parameterized type 338
- repository 376
- source control and 366
- source file and 376
- specialization of 373
- style macro 339
- syntax 350
- syntax, `>>` problem with 356
- syntax, alternative 355
- `typedef` 357
- use experience 339
- vector 342
- virtual member 364
- templates
 - design criteria for 338
 - relationships between 360
 - timing of 257
- templates, Cpp and 425
- temporaries
 - elimination of 252
 - lifetime of 143
- temporary, alternative points of destruction
 - of 145
- `terminate()` 194
- termination
 - arguments for 391
 - semantics 390
 - vs. resumption 390
- terminology 133
 - inventing 31
- Terrible, Mark 400
- terse I/O 185
- textbooks, C and C++ 170
- theory 103, 110
 - and logic 23
- `THIS`, Simula 39
- `this`
 - and `thunk` 265
 - assignment to 91, 212
 - pointer 39
 - reference 39
- Thompson, Ken 64, 161
- `throw` 385
- `thunk`, `this` and 265
- TI Explorer 392
- Tiemann, Mike 58, 166, 225, 292, 384, 395
- time, lack of 168
- time
 - needed for standard 129
 - needed to learn C++ 172
 - order, importance of 123
- timeline, C++ 4
- timeliness of standard 135
- timing
 - Release 2.0 124
 - of multiple inheritance 257
 - of templates 257
- tolerance 23
- tool
 - for problem solving, mathematics as 23
 - poor environment 65, 166
- tools
 - C++ 65
 - Cpp and 424
 - character-oriented 207
 - criteria 21
 - for language design 103
 - multiple inheritance 271
 - semantics-oriented 207
 - specialized 204
 - syntax-oriented 207
- toy, Pascal 44
- tradeoffs
 - for user 183
 - library design 182
- training of programmers 111
- transition 112–113, 276, 334
 - and *using-directive* 407
 - overloading and 414
 - path 153
 - to `namespaces` 407
 - to new cast 335
- translation
 - limits 136
 - unit of 57
- translations 165
- trigraphs 159
- true local variables 198
- `true` 1 254
- trust the programmer 116, 316

- try 385
- TSL see type-safe linkage
- tuning of memory management 211
- Turner, Scott 139, 141, 145
- two week rule 274
- type
 - abstract 279
 - bool, Boolean 254
 - cast and incomplete 329
 - checked, default 101
 - checking, C with Classes static 40
 - checking, C with Classes strong 40
 - checking, C++ 92
 - checking, C++ static 197
 - checking and design, static 107
 - checking, compile-time 31
 - checking, design and 107
 - checking, libraries and static 183
 - checking, static 31, 106, 324
 - checking, static and dynamic 107
 - class is user-defined 30
 - constructor for built-in 380
 - derivation from built-in 380
 - destructor for built-in 380
 - encoding 234
 - enum 253
 - field, explicit 50
 - field, switch on 72
 - identity 316
 - information and library, run-time 183
 - information, extended 318
 - information, run-time 266, 306
 - inquiry, compile-time 353
 - inquiry operator 324
 - inquiry, run-time 74
 - of char constant 225
 - of char literal 225
 - parameterized 107, 341
 - polymorphic 312
 - reference return 87
 - relation operator 325
 - return 293
 - safety and exceptions 385
 - safety, safety and 93
 - switch on 313
 - system 20
 - system, libraries and 183
 - system, static 117
 - template, parameterized 338
 - violation 117
- typedef 47
- template 357
- typeid 316
 - misuse of 308
 - use of 322
- type_info 194, 316

- types
 - definition of built-in 380
 - macros for generic 51
 - template and built-in 380
 - user-defined and built-in 32, 117
- type-safe
 - I/O 185
 - linkage 34, 232
 - linkage, TSL see
 - linkage, experience with 234
 - linkage problem 236
 - linkage syntax 399
- type-system, C++ syntax 208

U

- ugly C features 29
- unconstrained method 325
- undeclared function, call of 40
- undesirable
 - C techniques 378
 - Smalltalk techniques 378
- unexpected() 194, 395–396
- unicode character set 161
- uninitialized variables 99
- union 333
- uniqueness 37
- unit
 - of protection, class 54
 - of translation 57
- universal root class 184
- universities, releases to 70
- unnamed namespace 419
- Unruh, Erwin 367, 400
- unsafe operations 222
- Usable 239
- Usable_lock 239
- use
 - and misuse 115
 - complexity of 111
 - constraint through 345
 - experience, template 339
 - of RTTI 311, 313
 - of dynamic_cast 296, 320
 - of multiple inheritance 271, 322
 - of multiple libraries 183
 - of template 322
 - of typeid 322
 - within AT&T, C++ 65
- used function only, instantiate 347
- usenet comp.lang.c++ 166
- USENIX 164
 - workshop, Santa Fe 165
- user
 - population 61
 - self-sufficiency 37

- tradeoffs for 183
- user-defined
 - and built-in types 32, 117
 - operator 250
 - operator, precedence of 251
 - type, `class` is 30
- users
 - C++ serving current 110
 - book for 4
 - counting C++ 164
 - early 174
 - later 174
 - locking in 167
 - number of C++ 163
 - vs. lawyers 137
- Users' group, ASU, Simula 106
- using new cast 335
- using 404
 - `namespace` 407
 - `namespace` 404
 - `namespace`, using vs. 411
 - vs. using `namespace` 411
- using-declaration* 404
 - access-declaration* and 419
 - adjusting access 419
 - vs. *using-directive* 411
- using-directive* 404
 - transition and 407
 - using-declaration* vs. 411
- USL 127
 - library 184, 191

V

- value
 - of arguments 301
 - of experience 301
 - semantics 198
- variable
 - global 32
 - local 32
- variables
 - C 198
 - Modula-3 198
 - Pascal 198
 - Simula 198
 - Smalltalk 198
 - true local 198
 - uninitialized 99
- Vasa 147
- `vec` 49, 276
- vector
 - standard 194
 - `template` 342
- vector 49, 276
- verbose error-prone code 388

- verification, learning 169
- versioning, namespaces and 409
- versions of C++ 71
- vertical foundation library 191
- Vilot, Mike 192, 271, 391, 400
- violation, type 117
- virtual
 - function and multiple inheritance 265
 - function table, replicated 302
 - memory, initialization and 98
 - table layout, Cfront 320
 - table optimization 302
- virtual**
 - Simula 73
 - base, cast from 267, 312
 - `base class` 260
 - base layout 266
 - call in constructor 282
 - call, optimization of 116
 - copy 241
 - data 266
 - destructors 216
 - function 72, 74
 - function, C with Classes without 53
 - function efficiency 49, 75
 - function implementation 75
 - function, layout and 74
 - function optimization 239
 - function pointer 74
 - function, polymorphism without 49
 - function, pure 279
 - function recompilation 75
 - function table 74
 - function, `template` and 342
 - `member template` 364
 - table layout, standard 135
- visibility, access vs. 54
- VMS, DEC 392
- `void*` 213
 - assignment 230
 - cast from 333
 - conversion to 226
- `volatile`, ANSI C 128
- votes and reality 90
- voting rules 136
- `vptr` 74
- `vtbl` 74

W

- Waldo, Jim 269, 316
- Ward, Judy 71
- warning 240
 - Cfront 43
 - compiler 42, 334
 - ignoring 42

- warped design 338
- wars
 - compatibility 102
 - language 179
- wchar_t 161
- weaknesses, overstated 153
- weed
 - C a 37
 - C with Classes a 37
 - Fortran a 37
- Weinreb, Daniel 387
- WG-21, ISO 136
- what
 - is a programming language 7
 - is a standard 133
- whatis paper 106
- Wheeler, David 22
- why C 43
- Wilkes, Maurice 22
- Wilkinson, Nancy 71
- Williams, Tony 384
- window 154, 260
- withdrawal symptoms, C 93
- wordlink 50
- workaround
 - for multi-method 299
 - for resumption 393
 - keyword argument 156
- workarounds for order dependencies 97
- working groups 137
- workshop, Estes Park 165
- world view 23
- worst mistake 200
- wrappers 58
- writeonly 89
- writing
 - C in C++ 170
 - Fortran in C++ 170
 - Smalltalk in C++ 170
- wstring 194

X

- x() 94
- ~x() 94
- X3J16, ANSI 136
- Xerox PARC 309
- xor keyword 160
- xor_eq keyword 160

Y

- YACC 103
 - LALR(1) parser 68
- Yaker, Laura 214

- Yost, David 164

Z

- zero overhead rule 121
- Zortech 84, 164
 - compiler 166