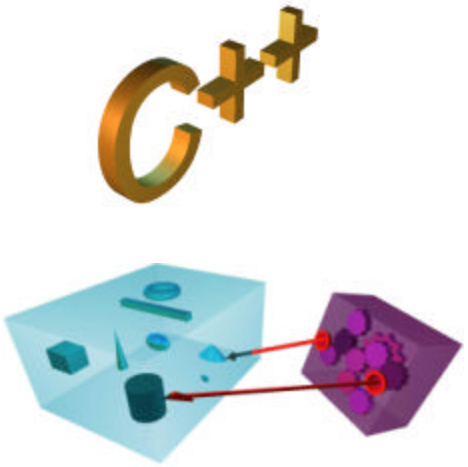
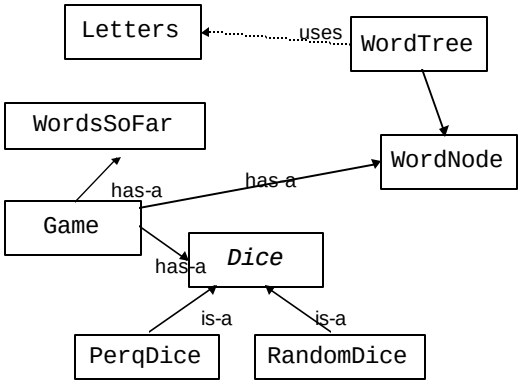
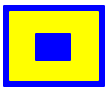
Introduction to C++

Language, Standard Library and OO Design





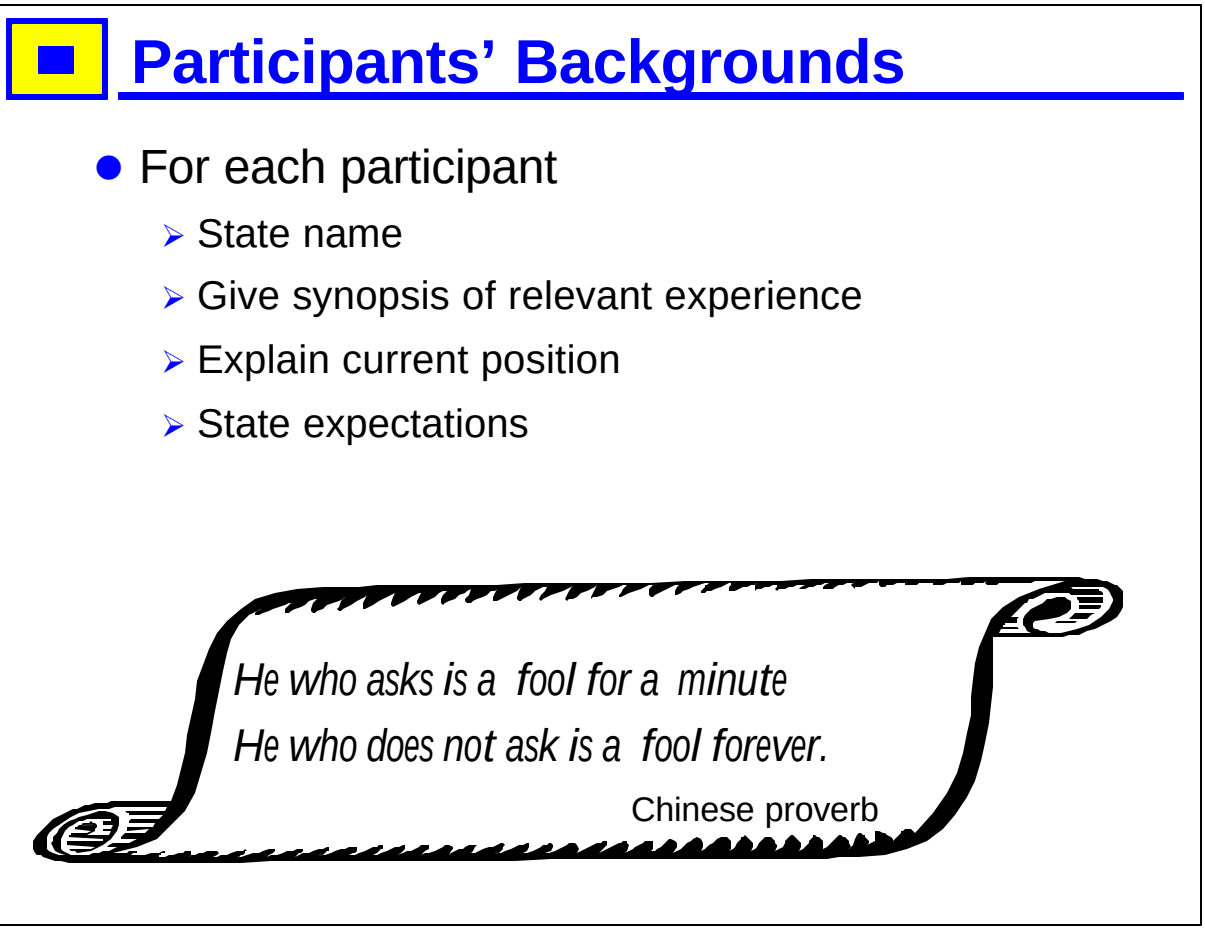
1. This course does not assume that you know any particular computer language or have any experience with software design, but it is aimed at people who have a university degree in a field related to computer science.



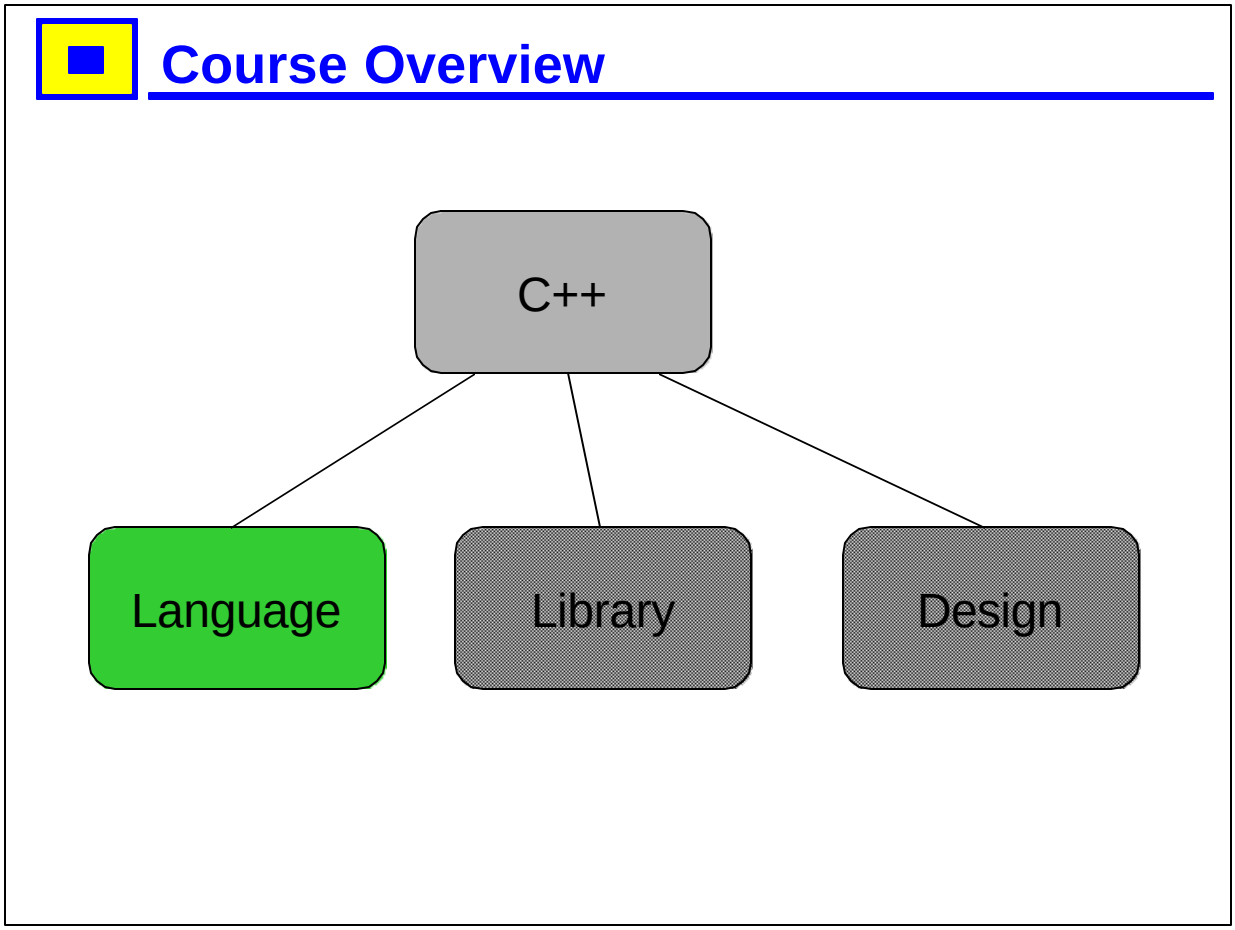
Outline

- Participants' backgrounds
- Course overview
- The C++ Language
- The C++ Standard Library
- The Design Process
- History of C++
- C++'s place in the world
- Resources

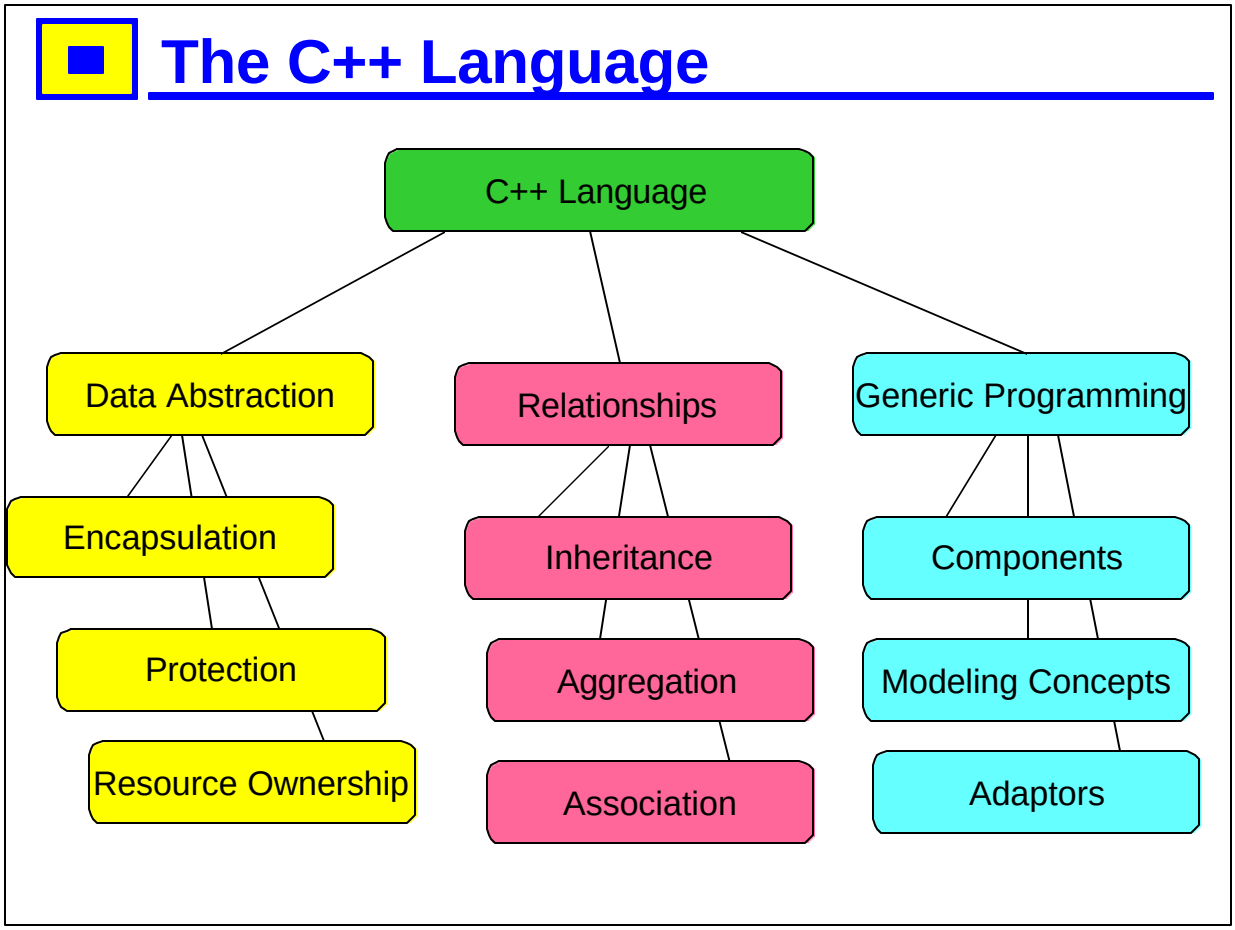
1. At the end of this chapter we will look at an overview of the exercises that will be done after each of the other chapters.



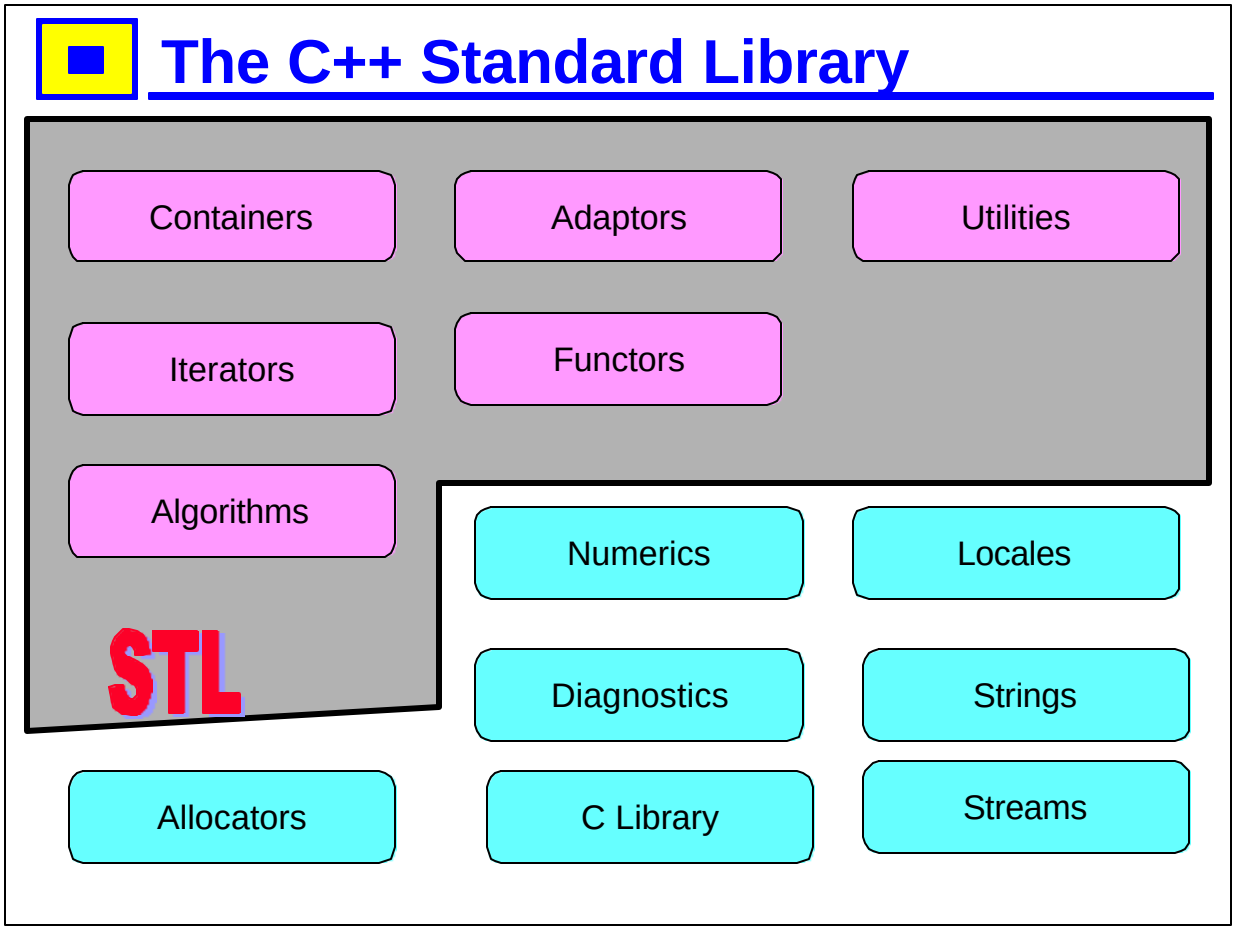
1. Please use the name-cards provided; some instructors are good at remembering peoples' names, but some have a hard time remembering their own :-)
2. If you have taken any courses in programming languages or analysis and design, mention them.
3. If you have had any experience programming in any language, summarise how much and what type of language.
4. If you have read any books related to C++ or object-oriented programming, this would also be a good thing to point out.
5. Your current position is important --- are you going to be using C++ right away? Are you already using it? Are you just learning it to see if it might be useful to you? Are you here because you want to be or because you were told to come?
6. The most important thing here is to tell the instructor what you want to get out of the course. The instructor can try to shift the focus of the course to best suit the needs of most of the participants, but only if you are clear and open about what you want to hear about and what you are not interested in. Don't expect that everyone will get exactly what they want, but please provide feedback as the course progresses on what is most helpful and what is less helpful for you.
7. Most instructors will not mind if, for part of the course you already know, you do something else like read a book or answer your mail. You probably want to avoid guffawing loudly at the daily comics, though :-)
8. If you have terminals in the classroom, feel free to use them during the classes as long as it doesn't bother the other students. If you expect that you will be doing this a lot, consider sitting at the rear of the room.
9. Students are sometimes called away unexpectedly for personal reasons or to put out fires in their offices... This is not a problem, but it is polite to tell the instructor either before or afterwards why you are leaving.



1. The topics we cover in this course fall into three main categories: the C++ language, the standard C++ library, and general design issues.
2. Since this is a hands-on course, in order to facilitate doing the exercises, we will not cover these topics directly in order; instead, we will cover some language issues, some of the library components and some design issues in most of the chapters.
3. To get an idea of the topics that will be covered, however, we will present them here broken down into language, library and design.

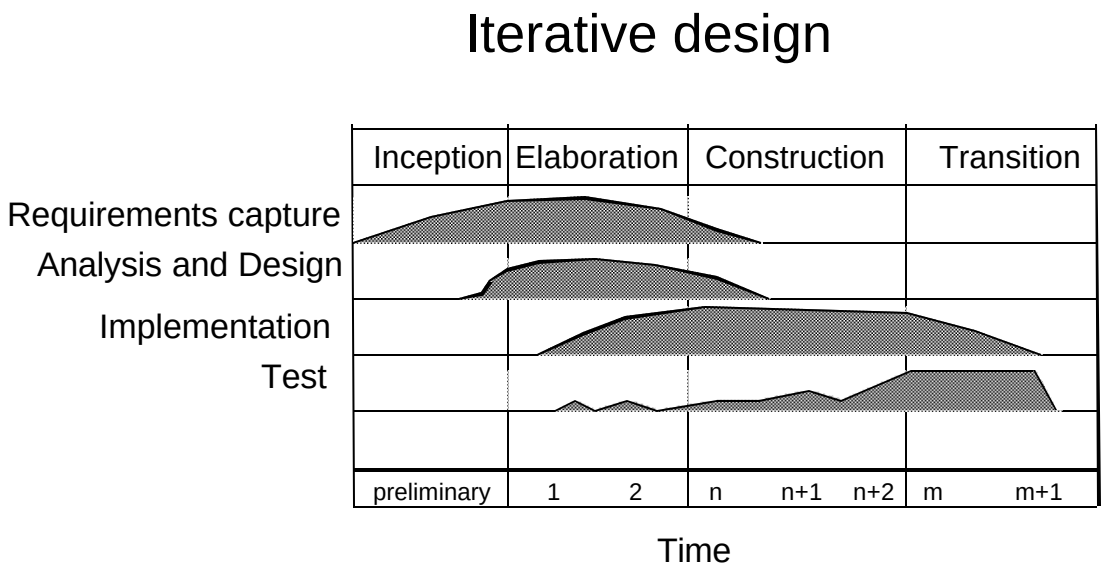


1. Data abstraction separates the interface (how the object is used) from the implementation (how it works inside). It is one of the oldest design techniques in computer programming, and it has been recognized as important for a number of decades. Traditional (procedural) languages like C and Fortran do not provide any support for data abstraction, although programming conventions can be used in those languages. C++, like all object-oriented languages, provides explicit support for data abstraction, through classes that define both data and methods (encapsulation), protection (compiler enforces access only to the public part of a class), and resource ownership (classes can manage resources by prohibiting outside access to them).
2. Relationships between objects and types are fundamental in object-oriented programming; inheritance defines a relationship between types corresponding to “is-a”, aggregation corresponds to “has-a”, and associations are more general relationships. We will look at the C++ mechanisms for representing these various relationships, and how they are used in the design process.
3. Generic programming is a notion that has gained a lot of popularity in the last few years; originally used in Ada, the Standard Template Library (roughly a subset of the standard C++ library) made it popular in C++. Basically, generic programming involves defining small, simple components that can be fit together in many different and useful ways, somewhat similar to the idea behind Lego(TM). We will look at the components that make up the standard library, how they are defined in terms of concepts and modeling, and one of the essential concepts of generic programming: adapters that provide different interfaces to existing components.

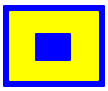


1. Containers are objects that contain other objects, e.g., vector, set, map.
2. Iterators represent locations in a container. Each container has its own iterator type.
3. Algorithms are operations on containers, e.g., find, sort, random_shuffle.
4. Functors are operations on objects, e.g., less, plus.
5. Adaptors are objects that change an interface, e.g., not1. (That's a one at the end, not an ell; there is also a not2.)
6. Utilities are components such as pairs, operations like comparison, etc. In the ANSI standard, allocators are included in the utilities section.
7. Diagnostics are provided to deal with exceptions.
8. Locales facilitate internationalization.
9. Numerics are container types that are optimized for speed, less general than containers, e.g., valarray, complex.
10. Strings replace C's character arrays.
11. Streams are used for input and output.
12. Allocators customize memory allocation, e.g., malloc_alloc.
13. We will not have time to cover every type in the standard library, as it is very large. However, we will cover the most important concepts, which will allow you to use standard documentation to use the other types.
14. A large part of the standard C++ library (ANSI and ISO ratified the standard library along with the language definition in November 1997) consists of components taken from the STL (Standard Template Library), written by Alexander Stepanov while at HP (now at SGI). The STL has continued to grow since the standard was defined, so it is no longer a strict subset of the standard library. For example, it now includes hashed containers, ropes and singly-linked lists.

The Design Process



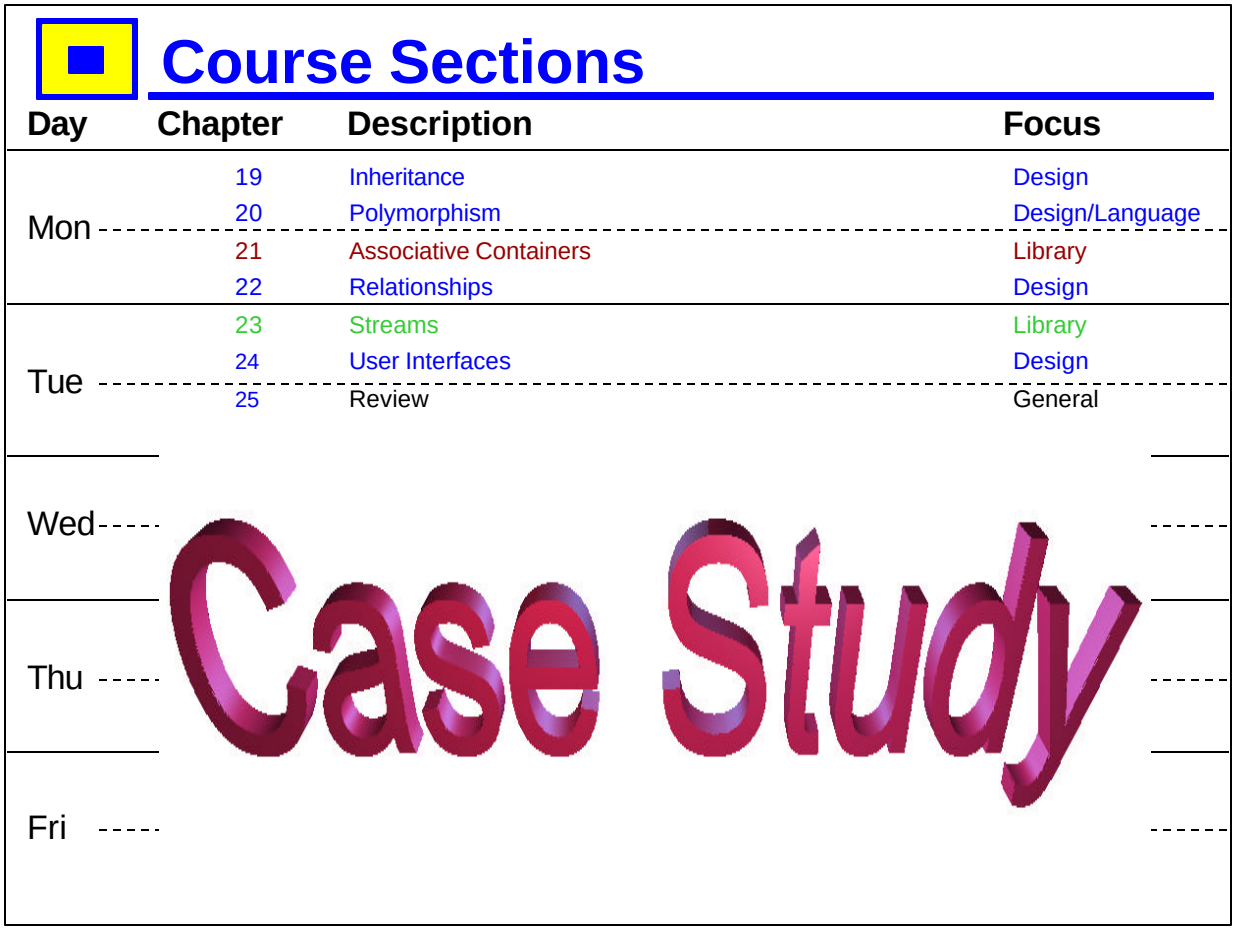
1. This graph is adapted from the Rational Rose book "Visual Modeling with The Unified Modeling Language" by Terry Quatrani, with permission from Addison Wesley Longman.
2. As the graph shows, object-oriented design is an iterative process. In the early phases of the design process, we concentrate on requirements capture: determining, with the aid of domain experts, exactly what the requirements of the system are. Next come analysis and design, which continue while implementation starts. Testing is done concurrently with implementation throughout the design process. Typically, the analysis and design phases will illustrate some shortcomings in the requirement capture, and the implementation will illustrate some shortcomings in the analysis and design. Therefore it is imperative that these phases continue in parallel, unlike the old waterfall method, where each phase was completed before the next phase was begun.
3. Analysis is typically used to refer to a high-level description of the project, describing logical components and their interactions. Design is a description of an actual system to perform the required tasks, taking into account things like the programming language(s) to be used, the architecture of the machine(s) the software will be running on, the communication infrastructure, and so on.
4. An important part of object-oriented design is communication between the different members of the design and implementation team, as well as between the members of the team and the domain experts who understand the goals of the system, though usually not the details of OO design and implementation. In order to facilitate this communication, using standard design diagrams and tools can be very helpful. The software industry is gradually moving towards the standard UML (unified modeling language) design notation, which is supported by such tools as Rational Rose.
5. Round-trip engineering tools, which allow iterative design and implementation (that is, the design notation can be converted to code and vice-versa) are beginning to become available, though in practice many of them are still difficult to use or provide only a subset of the functionality one would want.



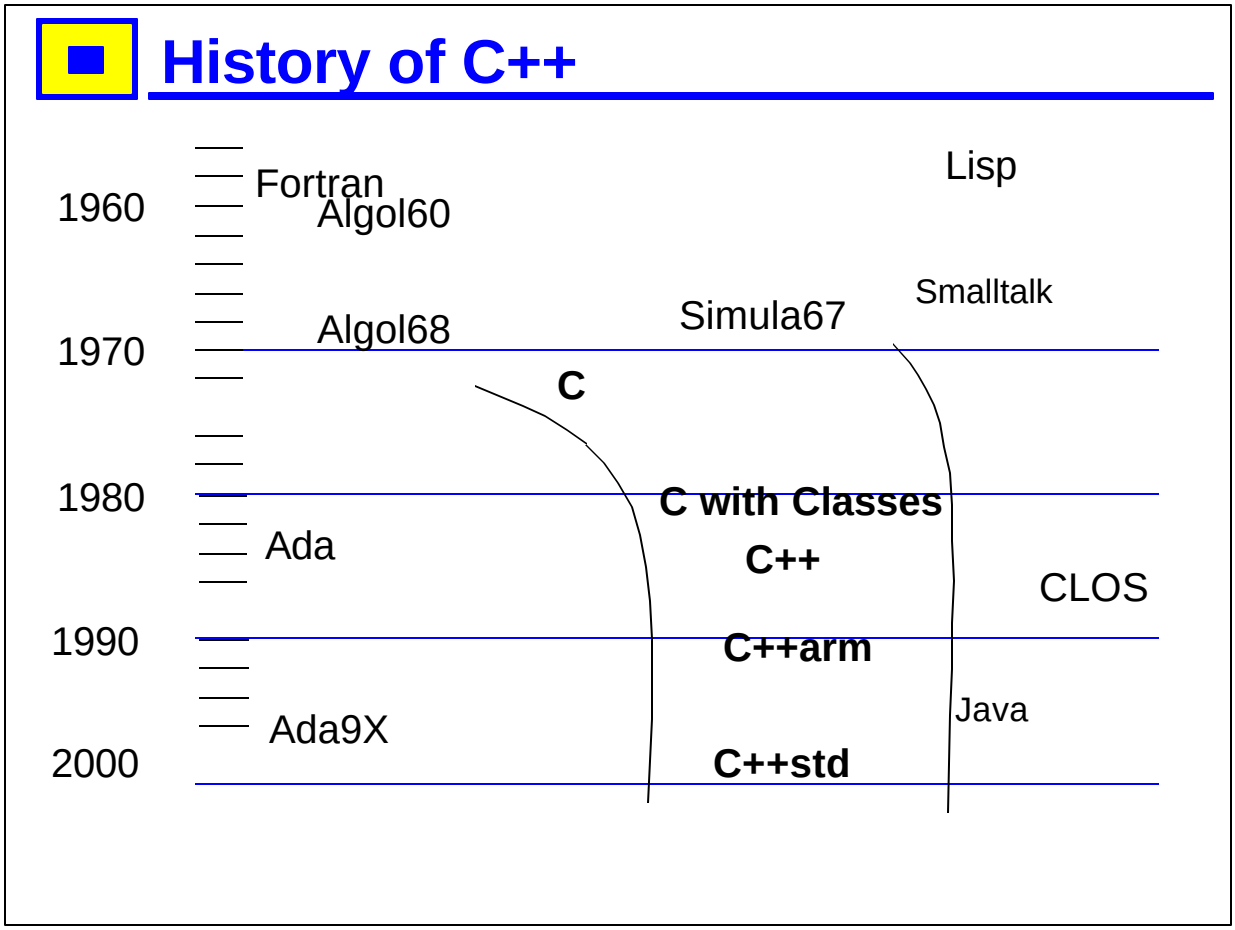
Course Sections

Day	Chapter	Description	Focus
Mon	0	Introduction	General
	1	Getting Started	General
	2	Variables and strings	Language/Library
	3	Functions	Language
Tue	4	Control Flow	Language
	5	Arrays	Language
	6	Pointers	Language
	7	Const and References	Language
Wed	8	Structures	Language
	9	Data Abstraction	Design
	10	Constructors	Design
	11	Accessor Functions	Language
Thu	12	Dynamic Allocation and Responsibilities	Design
	13	Exceptions	Design
	14	Operators	Language
	15	Templates and Generic Programming	Design/Language
Fri	16	Containers, Iterators and Algorithms	Library
	17	Functors and Adaptors	Library

1. The colors are not correct at the moment; eventually they will indicate whether the chapter is focusing on language, library, or design.
2. After each chapter will be an exercise; the exercises constitute about 50% of the course.
3. The exercises (apart from the first warm-up exercise) build on each other; by the end of the course we will have built a program based on the board game Perquackey.
4. Green is language, Red is library, Blue is design



1. After each chapter will be an exercise; the exercises constitute about 50% of the course.
2. The exercises (apart from the first warm-up exercise) build on each other; by the end of the course we will have built a program based on the board game Perquackey.

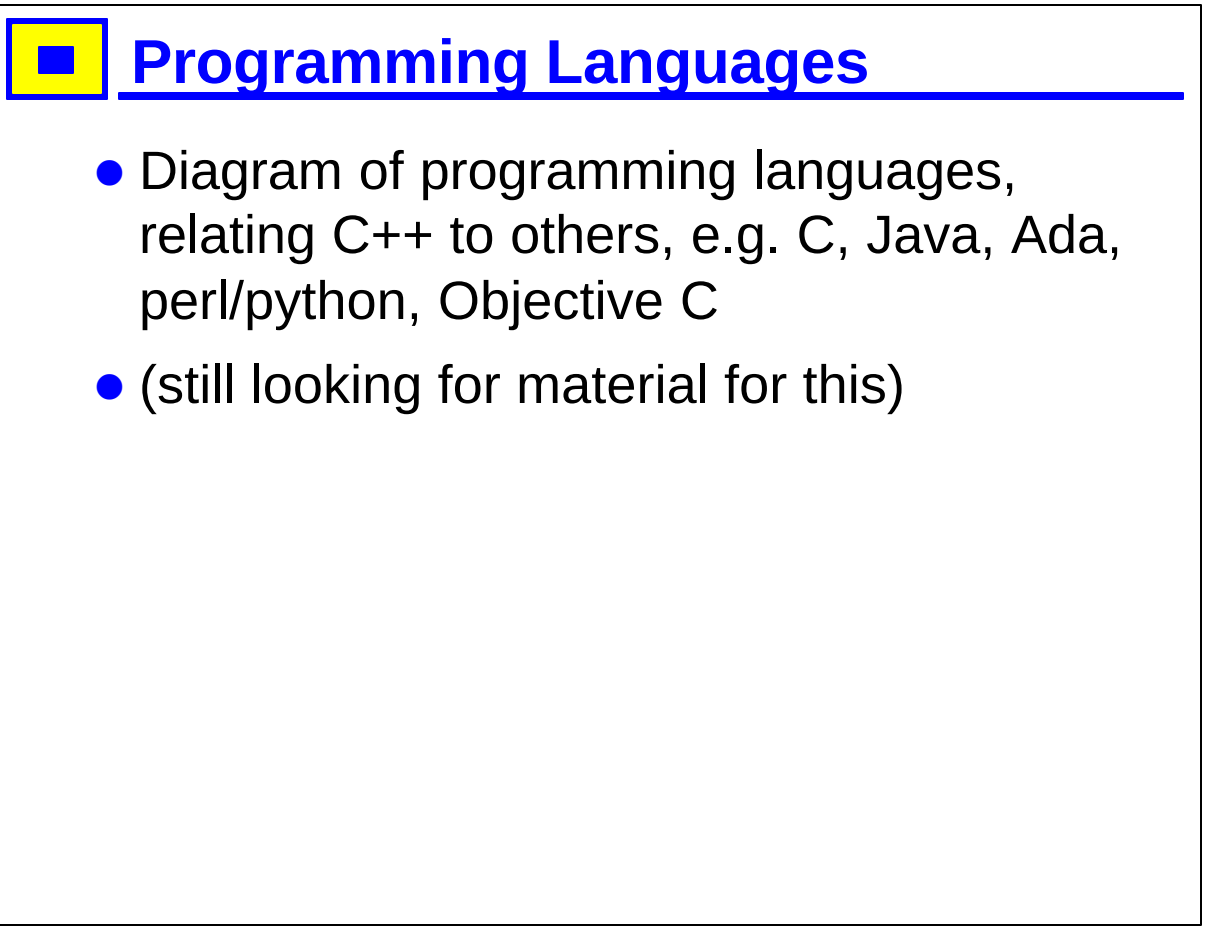


1. Source: Stroustrup, the Design and Evolution of C++.
2. C++ was developed by Bjarne Stroustrup at AT&T labs. It originated as an attempt to incorporate the Simula notion of class (providing explicit data abstraction) into C, which was at the time one of the most popular programming languages. Other features, such as inheritance, operator overloading and templates were added in successive versions of the language. The language continued to grow with the addition of exceptions, and refinements to the template mechanism, until November 1997, when the ANSI and ISO committees ratified the standard. This standard will remain unchanged for at least five years after the ratification, providing a solid foundation for users of the language.
3. Along with the ratification of the standard language definition, the standard library, a very large set of classes and functions, was ratified by the standard. As of this writing (February 2000), no compilers fully support either the standard or the library, but the major compilers are now close to full support.

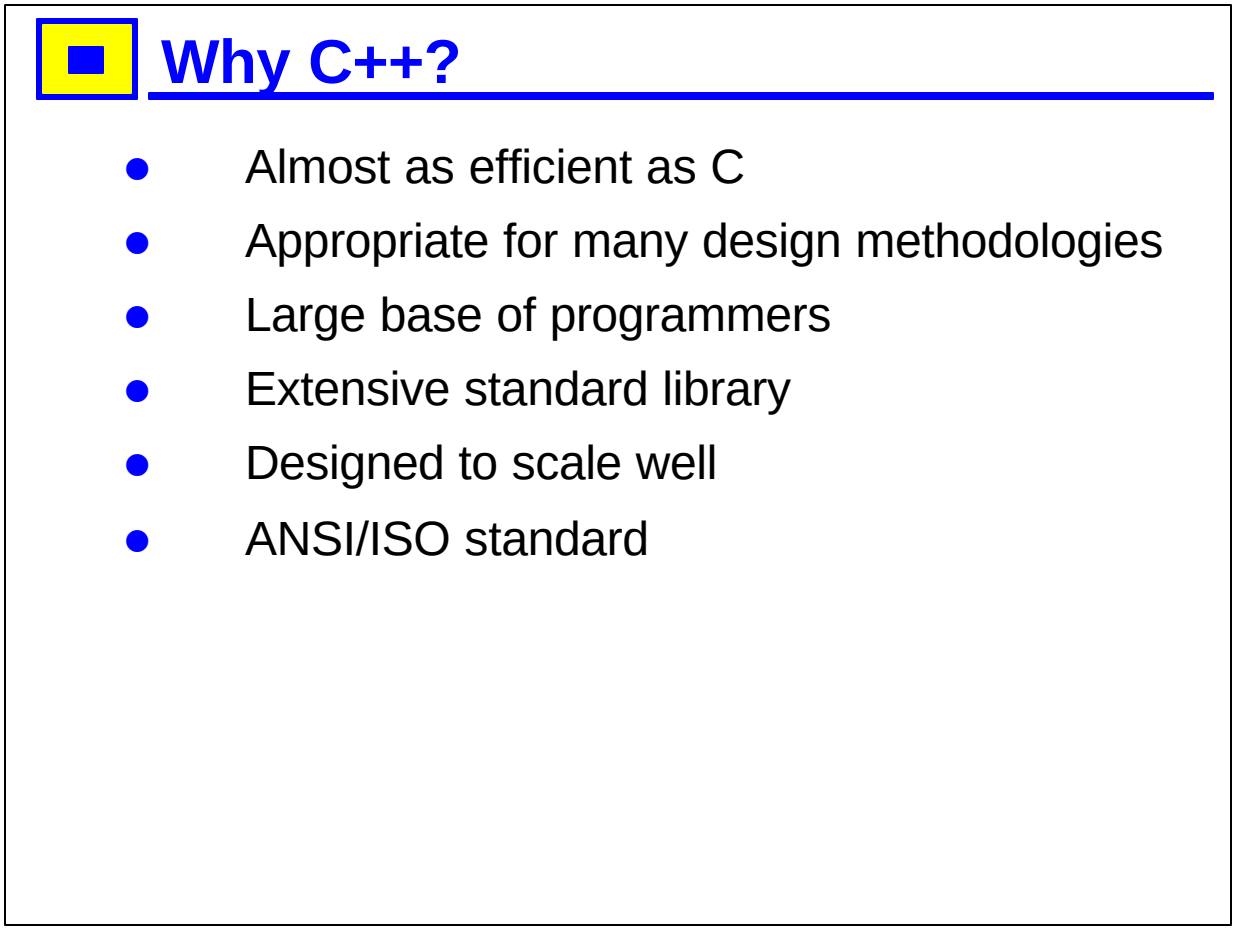
Programming Languages

- Characteristics of Languages
  - Efficiency
  - Flexibility
  - Ease of Design
  - Ease of Maintenance
  - Availability of Programmers
  - Availability of Libraries
- Many of these are subjective
- Different languages useful for different tasks

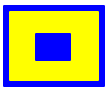
1. No one programming language is ideal for all tasks; it is important to choose a language appropriate for the kinds of applications you will be developing.
2. However, practical and political considerations often outweigh technical considerations.
3. General purpose languages have the advantages of more existing code and a larger base of programmers; these advantages often outweigh the disadvantages of a larger language and less specific language support for certain tasks.



1. Forth, Fortran, Python, Turbo Pascal, Prolog, Prolog++, OO-Cobol, Cobol, Haskel, Eiffel, Objective-C, Snobol, Algol, Ada
2. Which ones are standard and which are not
3. Try to put in many languages, make size represent importance, or market-share, perhaps.



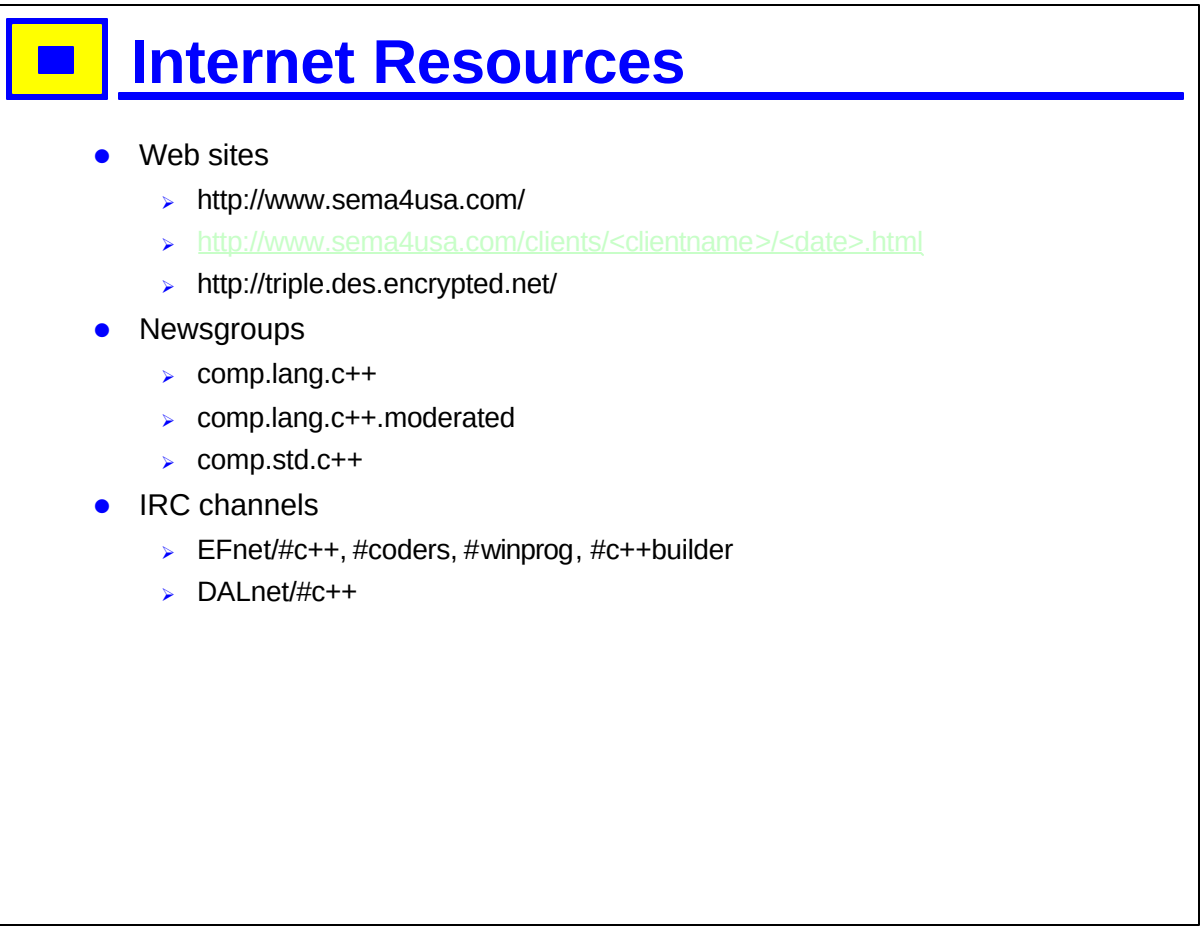
1. Well-written C++ code is almost as efficient as C code. Without a basic understanding of how C++ works, however, it is easy to write very inefficient C++ code.
2. C was designed with efficiency as one of its main goals. C code is typically almost as fast as assembler. C has existed for a long time, compared to most programming languages in common use, and therefore most platforms have highly optimized C compilers. C is the language used to write many operating systems and games, two types of applications where efficiency is critical. The biggest disadvantage of C is that it does not provide facilities that make object-oriented and generic programming easy.
3. C++ supports procedural, object-oriented and generic programming techniques.
4. There is support for development in C++ in many standard methodologies and their design tools (e.g., Software Through Pictures, Unified Modeling Language, Object Modeling Technique).
5. IDS estimated that 1.2 million C++ Implementations were sold in 1996. Stroustrup estimates the growth rate at 20% (+/- 10%) per year.
6. The Standard C++ library was ratified by the ISO and ANSI committees in November 1997.



Books

- Learning C++
  - Lippman: C++ Primer
  - Dewhurst and Stark: Programming in C++
  - Thinking in C++
- Reference
  - Stroustrup: The C++ Language, 3rd Edition
  - Austen: Generic Programming and the STL
  - Josuttis
- Advanced C++ and OO Design
  - Meyers: Effective C++, More Effective C++
  - Lippman: C++ Gems
  - Cline: C++ FAQs
  - Coplien: Advanced C++
  - Koenig: Ruminations on C++
  - Gamma et al: Design Patterns
  - Larman: Applying UML and Patterns
  - The Tao of Objects (oldish, though)

1. Stanley Lippman's C++ Primer (now in the 3rd edition) is one of the classic texts for learning C++, and still one of the best.
2. Thinking in C++ is recommended by many people; it is a free C++ tutorial,
3. Dewhurst and Stark's Programming in C++ does not cover the entire language, but has very good sections on design, something that is lacking in many other books on C++.
4. Scott Meyers has written several books that are considered essential reading by many C++ experts. Effective C++ provides 50 simple rules of thumb, with extensive explanations of the rationales behind them. More Effective C++ is not a replacement for Effective C++, but a continuation, with 35 more rules of thumb. There is now a second edition of Effective C++, about 80% rewritten from the first edition.
5. Stanley Lippman's C++ Gems is a collection of articles based on threads in the Usenet newsgroup comp.lang.c++. They cover important topics such as advanced template techniques, using exceptions, and design issues.
6. Cline's book C++ FAQs is based on the C++ Frequently Asked Questions posted monthly in the newsgroup comp.lang.c++, with many of the answers considerably expanded from the newsgroup.
7. James Coplien's Advanced C++ covers advanced techniques in excellent detail.
8. Andrew Koenig and Barbara Moo's Ruminations on C++ gives some very good insights into C++ development by two long-time practitioners from AT&T labs.
9. Negative recommendations: avoid Schildt. See accu.org's reviews for details.



1. There are more references and summaries on Semaphore's web site.
2. Comp.lang.c++ is a very-high volume newsgroup suitable for asking pretty much any question about C++. Comp.lang.c++.moderated gets about 100 messages a day. Read it for several months before posting there. Many of the world's most knowledgeable C++ people read and post to this newsgroup (e.g., Bjarne Stroustrup, Francis Glassboro, Scott Meyers, Andy Lippman, Andy Koenig)
3. This page needs a lot more added to it.
4. cscene, bytamin-c, accu.org
5. Should also mention C++ journals, Dr Dobbs.