Bjarne Stroustrup's C++ Glossary

Modified October 3, 2012

This is a glossary of C++ terms, organized alphabetically by concept.

The definitions/explanations of individual terms are necessarily very brief. To compensate, each entry includes one or more references to The C++ Programming language (Special Edition) (TC++PL) where you can find more comprehensive explanations and code examples. I use section references, rather than page numbers, so that this glossary can be used together with translations of my books. It is always wise to read a whole section rather than trying to glean information from a few semi-random sentences.

For people interested in the reasons behind the design of C++, most entries also refer to The Design and Evolution of C++ (D&E). Some entries contain links other parts of my home pages, especially to my FAQ and C++ Style and Technique FAQ. If I find the time, I'll add references to the ISO standard.

This glossary is specifically "C++ oriented". That is, it defines terms in the context of C++. For example, it defines generic programming in terms of templates and object-oriented programming in terms of virtual functions, rather than trying to be sufficiently abstract and general to cover all languages and all usages.

The entries are meant to be brief explanations, rather than precise definitions.

Suggestions for improved explanations, terms to add, or anything else that would make the glossary more useful, are most wellcome: bs at cs dot tamu dot edu.

Glossary

!= - the inequality operator; compares values for inequality returning a bool. TC++PL 2.3.1, 6.2, 16.3.10, 20.3.8, 22.4.3, 22.5.

#define - a directive that defines a macro.

#include - a mechanism for textual inclusion of one source file into another.

Typically used to compose a translation unit out of a .c file and the header files it needs to define its view if the rest of the program. TC++PL 2.7, 13, D&E 18.1.

+= - add-and-assign operator; a+=b is roughly equivalent to a=a+b. Often a useful operation for user-defined types. TC++PL 6.1.1, 6.2, 11,3,2, 20.3.9, 22.5.

.c file - file containing definitions.

.cpp file - see .c file

.cxx file - see .c file

.h file - see header file.

14882 - ISO/IEC 14882 - Standard for the C++ Programming Language.

<< - (1) iostream output operator. TC++PL 3.4, 21.2.1, D&E 8.3.1. (2) integer left-shift operator. TC++PL 6.2.

= - the assignment operator; *not* an equality operator. = can be used for non-const built-in types (except arrays), enumerations, strings, containers, iterators, complex, and valarray. For a class, = is by default defined member-wise assignment; if necessary, the writer of a class can define it differently. TC++PL 2.3.1, 6.2, 11.2, 16.3.4, 20.3.6, 22.4.3, 22.5, D&E 2.12.1,11.4.4.

=0 - curious notation indicating that a virtual function is a pure virtual function. TC++PL 12.3. D&E 13.2.1.

== - the equality operator; compares values for equality returning a bool. == can be used for built-in types, enumerations, strings, iterators, complex, and valarray. == is not by default defined for a class, but a user can define it for a user-defined type. Note that == doesn't have the naively expected meaning for C-style strings or arrays. TC++PL 2.3.1, 6.2, 16.3.10, 20.3.8, 22.4.3, 22.5.

>> - (1) iostream input operator. TC++PL 3.6, 21.3.2, D&E 8.3.1. (2) integer right-shift operator. TC++PL 6.2.

abstract class - a class defining an interface only; used as a base class. Declaring a member function pure virtual makes its class abstract and prevents creation of objects of the abstract class. Use of abstract classes is one of the most effective ways of minimizing the impact of changes in a C++ program and for minimizing compilation time. Example. TC++PL 2.5.4, 12.4.2, D&E 13.2.

abstract type - see abstract class.

abstraction - the act of specifying a general interface hiding implementation details. Classes, abstract classes, and templates are the primary abstraction

mechanisms in C++. See also: encapsulation.

access control - access to bases and members of a class can be controlled by declaring them public, protected, or private. TC++PL 15.3, D&E 2.3, 13.9.

ACCU - Association of C and C++ Users. A users group that among other things maintains a collection of professional book reviews.

adapter - a class that takes arguments producing a function object that performs an operation based on those arguments. A simple form of a higher-order function. For example, mem_fun() adapts a member function for use by the standard algorithms. See also: sequence adapter. TC++PL 18.4.4.

address - a memory location. TC++PL 5.1.

aggregate - an array or a struct without a constructor.

algorithm - a precise definition of a computation. The standard library provides about 60 standard algorithms, such as sort(), search(), and copy_unique(). TC++PL 3.8, 18.

alignment - placing objects in memory to suit hardware requirements. On many machines, an object must be aligned on a word boundary for acceptable performance.

allocator - object used by standard library containers to allocate and deallocate memory. TC++PL 19.4.

and - synonym for &&, the logical and operator. TC++PL C.3.1.

ANSI - The American national standards organization. Cooperates closely with ISO over the C++ standard.

ANSI C - see C.

ANSI C++ - C++

application - a collection of programs seen as serving a common purpose (usually providing a common interface to their users).

argument - a value passed to a function or a template. In the case of templates, an argument is often a type.

argument passing - The semantics of function call is to pass a copy of an argument. The copy operation is defined by the argument type's copy constructor or by binding to a reference. In either case the semantics is those of initialization. TC++PL 7.2.

argument-based lookup - lookup of a function name or operator based on

the namespace of the arguments or operands. Often called Koenig lookup after Andrew Koenig who proposed the scheme to the standards committee. TC++PL 8.2.6, 11.2.4, C.13.8.4.

ARM - The Annotated C++ Reference Manual by Margaret Ellis and Bjarne Stroustrup. The 1990 C++ reference manual with detailed comments about design details and implementation techniques. Now outdated. See also: C++ standard.

array - contiguous sequence of elements. An array doesn't know its own size; the programmer must take care to avoid range errors. Where possible use the standard library vector. TC++PL 5.2-3, C.7.

assignment operator - see =.

AT&T Bell Labs. - the industrial research and development labs where C and C++ were invented, initially developed, and initially used. D&E 2.14.

auto - In C and C++98 a largely useless keyword redundantly indicating stack allocation for local variables. In C++0x a keyword indicating that a variable gets its type from its initializer. For example: double d1 = 2; auto d2 = 3*d1; (d2 will have type double). Primarily useful in generic programming.

automatic garbage collection - see garbage collection.

auto_ptr - standard library class template for representing ownership of an object in a way that guarantees proper release (delete) even when an exception is thrown. See also: resource management, resource acquisition is initialization. TC++PL 14.4.2.

back-end - the parts of a compiler that generates code given an internal representation of a correct program. This representation is produced by a compiler front-end. See also: front-end.

backslash - see escape character.

back_inserter() - returns an iterator that can be used to add elements at the back of a container. TC++PL 19.2.4.

bad_alloc - standard exception thrown by new in case of failure to allocate free store. TC++PL 6.2.6.2, 19.4.5.

bad_cast - standard exception thrown if a dynamic_cast to a reference fails. TC++PL 15.4.1.1, D&E 14.2.2.

base class - a class from which another is derived. TC++PL 2.6.2, 12, 15, D&E 2.9.

base initializer - initializer for a base class specified in the constructor for a derived class. TC++PL 12.2.2, 15.2.4.1, D&E 12.9.

basic guarantee - the guarantee that basic invariants are maintained if an exception is thrown and that no resources are leaked/lost. Provided by all standard library operations. See also exception safety, nothrow guarantee, and strong guarantee. TC++PL E.2.

basic_string - general standard-library string template parameterized by character type. See also: string, C-style string. TC++PL 20.3.

BCPL - ancestor to C and C++ designed and implemented by Martin Richards. TC++PL 1.4, D&E 1.1, 3.1.

Bell labs - see AT&T Bell Labs.

binary operator - an operator taking two operands, such as /, &&, and binary *.

binder - a function taking a function and a value, returning a function object; when called, that function object will invoke the function with the value as an argument in addition to other arguments supplied in the call. The standard library provides bind1st() and bind2nd() for binding the first and second argument of a binary function, respectively. TC++PL 18.4.4.

bit - a unit of memory that can hold 0 or 1. An individual bit cannot be directly accessed in C++ (the unit of addressing is a byte), but a bit can be accessed through a bitfield or by using the bitwise logical operators & and |. TC++PL 6.2.4.

bitand - synonym for &, the bitwise and operator. TC++PL C.3.1.

bitfield - a number of bits in a word made accessible as a struct member. TC++PL C.8.1

bitor - synonym for |, the bitwise or operator TC++PL C.3.1.

bitset - a standard library "almost container" holding N bits and providing logical operations on those. TC++PL 17.5.3.

Bjarne Stroustrup - the designer and original implementor of C++. The author of this glossary. See also: my home page.

block - see compound statement. See also: try-block.

block comment - comment started by /* and terminated by */. TC++PL 6.4, D&E 3.11.1.

bool - the built-in Boolean type. A bool can have the values true and false. TC++PL 4.2, D&E 11.7.2.

boost.org - a collection of people - many with ties to the C++ standards committee - devoted to creating a body of quality - peer reviewed - open

source libraries designed to interoperate with the standard library. Their central "home" is their website.

Borland C++ Builder - Borland's implementation of C++ together with proprietary libraries for Windows programming in an IDE.

bug - colloquial term for error.

built-in type - A type provided directly by C++, such as int, double, and char*. See also: integral types, floating-point type, pointer, reference. TC++PL 4.1.1, 5.1, 5.2, 5.5, D&E 4.4, 15.11.3.

byte - a unit of memory that can hold a character of the C++ representation character set. The smallest unit of memory that can be directly addressed in C++. Usually, a byte is 8 bits. TC++PL 4.6.

C - programming language designed and originally implemented by Dennis Ritchie. C++ is based on C and maintains a high degree of compatibility with C. See also: K&R C, C89, C99, ANSI C. TC++PL B, D&E 3.12.

C standard library - the library defined for C in the C standard. Inherited by C++. Most C standard library functions have safer and more convenient alternatived in the C++ standard library. See also: algorithm, container, stream I/O, string, locale.

C++ - a general-purpose programming language with a bias towards systems programming that supports procedural programming, data abstraction, object-oriented programming, and generic programming. C++ was designed and originally implemented by Bjarne Stroustrup. C++ is defined by ISO/IEC 14882 - Standard for the C++ Programming Language. TC++PL describes C++ and the fundamental techniques for its use. A description of the design considerations for C++ can be found in D&E. Many commercial and free implementations exist. TC++PL 1.3,-5, 2.1, D&E 0.

C++ standard - the definition of C++ provided by ISO. Available from ANSI; see my C++ page. TC++PL 1.4, B.1. D&E 6.1.

C++ standards committees - the ISO committee for C++ (WG21) and the various national standards committees that closely cooperate with it (BIS, AFNOR, DIN, etc.). Did the ANSI/ISO standards committee spoil C++?. See also: C++ Standard. D&E 6.2.

C++ standards process - see C++ standards committees

C++/CLI - A set of Microsoft extensions to C++ for use with their .Net system. See FAQ comments.

C++03 - name for the minor revision of the C++ standard represented by the 2003 corrigenda ("a bug fix release").

C++0x - the upcoming revision of the ISO C++ standard; 'x' is scheduled to be '9'. See my publicatons page.

C++98 - the ISO C++ standard. See my C++ page.

C-style cast - dangerous form of explicit type conversion; prefer new-style cast if you must use explicit type conversion. TC++PL 6.2.7, D&E 14.3.5.1.

C-style string - zero-terminated array of characters, supported by C standard library functions. A low-level and error-prone mechanism; where possible prefer strings. TC++PL 3.5.1, 20.3.7, 20.4.

C/C++ - (1) an abbreviation used when discussing similarities, differences, and compatibility issues of C and C++. (2) a mythical language referred to by people who cannot or do not want to recognize the magnitude of differences between the facilities offered by C and C++ or the significant differences in the programming styles supported by the two language. See also: multi-paradigm programming, object-oriented programming, generic programming, exception, template, user-defined type, C++ standard library.

C/C++ compatibility - C++ was designed to be as compatible as possible to C, but no more. This basically means as compatible as can be without compromising C++'s level of type safety. You can download Appendix B of TC++PL,. Compatibility, which describes incompatibilities and differences in facilities offered by C and C++. TC++PL B. D&E 2.7, 3.12, 4.5.

C89 - The 1989 ANSI standard for C based on K&R C with a few additions borrowed from C++, such as function prototypes and const. See also: K&R C, C99.

C99 - The 1999 ISO standard for C based on C89 with additions to support Fortran-style numeric computation. It also borrows a few more features, such as line comments (// comments) and declarations as statements, from C++.

call-by-reference - declaring a function argument type to be a reference, thus passing a reference rather than a value to the called function. See Also: call-by-value. TC++PL 5.5, D&E 3.7.

call-by-value - passing a copy of an argument to the called function. The semantics of function call is to pass a copy of an argument. The copy operation is defined by the argument type's copy constructor. See Also: call-by-reference. TC++PL 7.2.

cast - operator for explicit type conversion; most often best avoided. See also dynamic_cast, C-style cast, new-style cast. TC++PL 6.2.7, D&E 7.2, 14.2.2.1.

catch - keyword used to introduce a catch-clause.

catch(...) - catch every exception. TC++PL 14.3.2, D&E 16.5.

catch-clause - a part of a try-block that handles exceptions of a specified type. Also called a handler or an exception handler. TC++PL 8.3.1, 14.3, D&E 16.3-4.

cerr - standard unbuffered ostream for error or diagnostic output. TC++PL 21.2.1.

Cfront - the front-end of Bjarne Stroustrup's original C++ compiler. D&E 3.3.

char - character type; typically an 8-bit byte. See also: wchar_t. TC++PL 4.3, C.3.4.

char* - pointer to a char or an array of char. Typically assumed to point to a C-style string. Prefer a standard library string over a C-style string when you can. TC++PL 2.3.3, 13.5.2.

character set - a set of integer values with a mapping to character representations; for example, ASCII (ANSI13.4-1968) gives meaning to the values 0-127. ASCII is C++'s representation character set, the character set used to represent program source text. TC++PL C.3. D&E 6.5.3.

character type - char, unsigned char, and signed char. These are three distinct types. See also: wchar_t. TC++PL 2.3.1, 4.3, C.3.4.

cin - standard istream. TC++PL 3.6, 21.3.1 D&E 8.3.1.

class - a user-defined type. A class can have member functions, member data, member constants, and member types. A class is thee primary mechanism for representing concepts in C++. See also: template class. TC++PL 2.5.2, 10, D&E 2.3.

class hierarchy - a collection of classes organized into a directed acyclic graph (DAG) by derived/base relationships. TC++PL 2.6.2, 12, 15, D&E 1.1, 7.2, 8.2.3.

class template - see template class.

Classic C - see K&R C.

clone - a function that makes a copy of an object; usually a clone function relies on run-time information (e.g. a virtual function call) to correctly copy an object given only a pointer or reference to a sub-object.

closure - object representing a context. C++ does not have general closures, but function objects can be efficiently used to hold specific parts of a context relevant to a computation. TC++PL 22.4.7, 18.4.

co-variant return type - see return type relaxation.

code generator - the part of a compiler that takes the output from the front-end and generates code from it. See also: back-end, optimizer.

collection - a term sometimes used as a synonym for container.

Comeau C++ - a family of ports of the EDG C++ front-end.

comment - block comment /* ... */ or line comment // ...

compatibility - see C/C++ compatibility.

compiler - the part of a C++ implementation that produces object code from a translation unit. See also: front-end, back-end.

complex - standard library complex number template parameterized by scalar type. TC++PL 11.3, 22.5, D&E 3.6.1, 8.5, 15.10.2.1.

compound statement - sequence of statements enclosed in curly braces: { ... } See also: try-block. TC++PL 2.3, 6.3.

concept - a C++ language construct, providing type chaecking for template arguments.

concept checking - see constraint.

concrete type - a type without virtual functions, so that objects of the type can be allocated on the stack and manipulated directly (without a need to use pointers or references to allow the possibility for derived classes). Often, small self-contained classes. See also abstract class, vector, list, string, complex. TC++PL 25.2.

const - attribute of a declaration that makes the entity to which it refers readonly. See also: const member function. TC++PL 5.4, D&E 3.8.

const definition - declaration of a const including an initializer.

const member function - member function declared not to modify the state of the object for which it is called. Can be called for const objects only. TC++PL 10.2.6, D&E 13.3.

constant - literal, object or value declared const, or enumerator.

constant expression - expression of integral type that is evaluated at compile time. TC++PL C.5.

constraint - rule that restricts the set of acceptable arguments for a template parameter. For example "the argument must have + and - operators". Examples. D&E 15.4.

constructor - member function with the same name as its class, used to

initialize objects of its class. Often used to establish an invariant for the class. Often used to acquire resources. A constructor establishes a local environment in which member functions execute. See also: order of construction, destructor. TC++PL 10.2.3, D&E 2.11.1.

const_cast - a type conversion operation that conversion between types that differ in const and volatile type modifiers only. See also: cast. TC++PL 15.4.2.1, D&E 14.3.4.

container - (1) object that holds other objects. (2) type of object that holds other objects. (3) template that generates types of objects that hold other objects. (4) standard library template such as vector, list, and map. TC++PL 16.2, 16.2.3, 17, D&E 15.3.

controlled variable - a variable used to express the part of the exit condition of a loop that varies each time around the loop. For example ``i'' in for (int i=0; i<max; ++i) f(i);

conversion - explicit type conversion or implicit type conversion. See also: user-defined type conversion.

conversion operator - operator function specifying a conversion from a user-defined type to either another user-defined type or a built-in type. Note that constructors cannot define conversions to built-in types. TC++PL 11.4, D&E 3.6.3.

copy assignment - an assignment accepting an object of the class itself as its argument, typically Z::operator=(const Z&). A copy assignment is used for assignment of an object of type T with an object of type T. If a copy assignment is not declared for a class, memberwise copy is used. See also: copy constructor. TC++PL 10.4.4.1, 10.4.6.3 D&E 11.4.

copy constructor - a constructor accepting an object of the class itself as its argument, typically Z::Z(const Z&). A copy constructor is used for initialization of objects of type T with objects of type T. If a copy constructor is not declared for a class, memberwise initialization is used. See also: call-by-value, argument passing, value return, copy assignment. TC++PL 10.4.4.1, 10.4.6.3, D&E 11.4.

copy() - standard algorithm for copying one sequence into another. The two sequences need not be of the same type. TC++PL 18.6.1.

copying class object - an object of a class is copied by the class' copy assignment and copy constructors. The default meaning of these operations is memberwise copy. TC++PL 10.4.4.1, 10.4.6.3 D&E 11.4.

cout - standard ostream. TC++PL 3.4, 21.2.1, D&E 8.3.1.

cpp - see preprocessor.

crosscast - a cast from a class to a sibling class. See also: dynamic_cast, upcast, downcast. TC++PL 15.4.

Currying - producing a function of N-M arguments by specifying M arguments for a function of N arguments. See also: binder, default argument. TC++PL 18.4.4.1.

D&E - Bjarne Stroustrup: The Design and Evolution of C++. Addison Wesley. 1994. A book describing why C++ looks the way it does - the closest to a design rationale that we have for C++.

data abstraction - programming relying on user-defined types with well-defined interfaces. See also: generic programming and object-oriented programming. TC++PL 2.5, 24.2.2, D&E 9.2.1.

data hiding - see information hiding

data member - member of a class that can hold a value. A memer can be a static member or a non-static member. TC++PL 2.5.2-3, 10.2, D&E 2.3, 2.5.2.

declaration - an introduction of a name into a scope. The type of the name must be specified. If the declaration also specifies the entity to which the name refers, the declaration is also a definition. TC++PL 4.9, D&E 3.11.5.

decltype - C++ox operator meaning the type of its operand. For example: const double& d1 =2.0; decltype(d1) d2; (d2 will also be a const double&). Primarily useful for writing forwarding functions in generic programming.

default argument - a value specified for an argument in a function declaration, to be used if a call of the function doesn't specify a value for that argument. This is commonly used to allow a simple interface for common uses while making it easy to use less common facilities by specifying more arguments. See also: default template argument, binder. TC++PL 7.5, 10.2.3, D&E 2.12.2.

default constructor - constructor requiring no arguments. Used for default initialization. TC++PL 10.4.2, 10.4.6, D&E 2.12.2, 15.11.3.

default template argument - a type or value specified for an argument in a template declaration, to be used if a use of the template doesn't provide a type or value for that argument. This is commonly used to allow a simple interface for common uses while making it easy to use less common facilities by specifying more arguments. See also: default argument. TC++PL 13.4.1, B.3.5.

default value - value defined by a default constructor. For built-in types, the default value is defined to be 0. TC++PL 4.9.5, 10.3.1, 10.4.2 D&E 15.11.3.

definition - a declaration that specifies the entity to which the declared name refers. See also: one definition rule, variable definition, const definition, template definition, function definition. TC++PL 4.9, D&E 15.11.3.

delayed evaluation - technique for eliminating temporary values, and in general to delay a computation until sufficient information is available to do it well. TC++PL 21.4.6.3, 22.4.7.

delete - object destruction operator. Invokes destructor, if any. See also: resource management, memory management, garbage collection, operator delete(). TC++PL 6.2.6, D&E 2.3, 10.2.

deprecated feature - feature left in a programming language for historical reasons only. The standard s committee recommends against its use and warns that it may be removed in future revisions of the standard.

deque - double-ended queue (pronounced "deck"). A standard library template alowing insertions and deletions at both ends. Use a vector if you need insertions and deletions only at one end (as is typical). Use a list if you need frequent insertions and deletions in the middle. TC++PL 17.2.3.

derived class - a class with one or more base classes TC++PL 2.6.2, 12, 15, D&E 3.5.

design - creating a clean and reasonably simple structure of a system TC++PL 23.3.

design of C++ - see D&E.

destructor - member of a class used to clean up before deleting an object. It's name is its class' name prefixed by '~'. For example, Foo's destructor is ~Foo(). Often used to release resources. A destructor is implicitly called whenever an object goes out of scope or is deleted. See also: virtual destructor, order of destruction. TC++PL 10.4.2, D&E 2.11.1, 3.11.2.

digraph - alternative representation for C++ representation characters that doesn't exist in every national character set, such as {, }, [,], and #: <%, %., <:, :>, and %:. TC++PL C.3.1.

double - double-precision floating-point number. TC++PL 4.5.

double dispatch - a technique for selecting a function to be invoked on the dynamic type of two operands. TC++PL 21.2.3.1, D&E 13.8.

downcast - a cast from a base class to one of its derived classes. The name reflects the fact that in programming, trees tend to be drawn growing downwards from the roots. See also: dynamic_cast, upcast, crosscast. TC++PL 15.4.

dynamic memory - see free store.

dynamic type - the type of an object as determined at run-time; e.g. using dynamic_cast or typeid. Also known as most-derived type.

dynamic type safety - type safety enforced at run time (typically requiring a programmer to catch exceptions to deal with violations). An example is range checking for vectors.

dynamic_cast - a type conversion operation that performs safe conversions using on run time type information. Used for navigation of a class hierarchy. See also: downcast, crosscast, static_cast. TC++PL 15.4.1, D&E 14.2.2, 14.3.2.1.

EDG C++ front-end - a quality C++ compiler front-end, which is the core of several well-regarded C++ compilers.

element - an object in a container.

encapsulation - the enforcement of abstraction by mechanisms that prevent access to implementation details of an object or a group of objects except through a well-defined interface. C++ enforces encapsulation of private and proteced members of a class as long as users do not violate the type system using casts. See also: interface and access control. TC++PL 15.3, 24.3.7.4, D&E 2.10.

enum - keyword for declaring enumerations. TC++PL 4.8, D&E 11.7.

enumeration - a user-defined type consisting of a set of named values. TC++PL 4.8, D&E 11.7.

enumerator - a name identifying a value of an enumeration. TC++PL 4.8, D&E 11.7.

equality operator - see ==.

error handling - see exception handling.

escape character - the character \, also called backslash, sed an initial character in representations of characters that cannot be represented by a single ASCII character, such as newline ('\n') and horizontal tab ('\t'). TC++PL C.3.2.

exception - object thrown by a throw-statement and (potentially) caught by an exception handler associated by a try-block. See also: exception safety, termination semantics, catch. TC++PL 8.3, 14.2, D&E 16.

exception handler - a catch-clause associated with a try-block for handling exceptions of a specified type. TC++PL 8.3.1, 14.3, D&E 16.3-4.

exception handling - the primary way of reporting an error that cannot be handled locally. An exception is thrown and will be caught by an exception handler or terminate() will be called. See also: exception safety, termination semantics, try-block, throw, catch. TC++PL 8.3, 14, E, D&E 16.

exception safety - the notion that a program is structured so that throwing an exception doesn't cause unintended side effects. See also: basic guarantee, strong guarantee, and nothrow guarantee. You can download Appendix E Standard-Library Exception Safety of TC++PL describing techniques for exception handling. TC++PL E.2.

executable file - the result of linking the object files of a complete program. See also: compiler, linker.

explicit - keyword used to define a constructor so that it isn't used for implicit conversions. TC++PL 11.7.1.

explicit call of constructor - See placement new.

explicit call of destructor - destructors are implicitly called when an object goes out of scope or is deleted. However, if a user have taken over construction (using placement new) and destruction, a destructor must be explicitly called. Example. For example, explicit call of destructor is used in the implementation of standard library containers. See also: placement new. TC++PL 10.4.11, E.3.1, D&E 10.5.1.

explicit constructor - constructor so that will not be used for implicit conversions. TC++PL 11.7.1.

explicit instantiation - explicit request to instantiate a template in a specific context. See also: template instantiation. TC++PL C.13.10, D&E 15.10.1.

explicit qualification - (1) by namespace name, see qualified name. (2) by template argument. TCP++L 13.3.2.

explicit type conversion - type conversion (explicitly) requested by the use of a C-style cast, new-style cast, or functional notation. See also, implicit type conversion, user-defined type conversion. TC++PL 6.2.7, D&E 14.3.2.

expression - combination of operators and names producing a value. TC++PL 6.2.

extended type information - any scheme that provides additional information base on the standard run time type information. TC++PL 15.4.4.1, D&E 14.2.5.2.

extension - see language extension

extern - a keyword used to indicate that the definition of an entity being declared is defined elsewhere. Because "extern: is only necessary for global variables it is largely redundant.

extracter - an iostream >> (put to) function. TC++PL 21.2,21.3, D&E 8.3.1.

facet - a class representing a primitive aspect of a locale, such as a way of writing an integer or a character encoding. TC++PL D.3.

false - bool value; converts to 0. TC++PL 4.2, D&E 11.7.2.

fat interface - an interface with more member functions and friends than are logically necessary. TC++PL 24.4.3.

field - see bitfield.

file - a sequence of bytes or words holding information in a computer. The term "file" is usually reserved to information placed on disk or elsewhere outside the main memory. The iostream part of the C++ standard library provides ifstream, ofstream, and fstream as abstraction for accessing files. TC++PL 21.5.

file stream - stream attached to a file. See also, fstream, ifstream, ofstream. TC++PL 21.5.1.

finally - a language construct supporting ad hoc cleanup in some languages. Similar, but not identical to C++'s catch(...). Use the "resource acquisition is initialization" technique instead.

find() - standard library linear search algorithm for a value in a sequence. TC++PL 18.5.2.

find_if() - standard library linear search algorithm for an element meeting a search criterion in a sequence. TC++PL 18.5.2.

float - single-precision floating-point number. TC++PL 4.5.

floating-point literal - the source text representation of a floating point value. For example, 0.314e1. TC++PL 4.5.1.

floating-point type - a float, double, or long double. A floating-point number is typically represented as a mantissa and an exponent. TC++PL 4.5.

for-statement - iteration statement specifying an initializer, an iteration condition, a "next-iteration" operation, and a controlled statement. TC++PL 6.3.3.

free store - memory allocated by new; also called dynamic memory. Often standard library facilities, such as vector, can be used to avoid explicit use of free store. TC++PL 6.2.6, 10.4.3, D&E 2.11.2, 11.4.2.

free() - C standard deallocation function. Use delete instead.

free-standing function - a function that is not a member function. Useful for decreasing coupling between representation and algorithm. TC++PL 7, 18.

friend - a function or class explicitly granted access to members of a class by that class. TC++PL 11.5, C.11.4, D&E 2.10, 3.6.1-2.

friend function - a function declared as friend in a class so that it has the same access as the class' members without having to be within the scope of the class. And, no, friends do not "violate encapsulation". TC++PL 11.5, 11.2.3, C.11.4, D&E 2.10, 3.6.1.

front-end - the parts of a compiler that perform lexical and syntax checking, type checking, and initial semantic checking of a translation unit. Typically all compiler error messages comes from the front-end. See also: back-end. D&E 3.3.

front_inserter() - returns an iterator that can be used to add elements at the front of the container . TC++PL 19.2.4.

fstream - a file stream for input and output.

function - a named sequence of statements that can be invoked/called given arguments and that might return a value. The type of the function includes the number and types of argument and the type of the value returned, if any. See also: function declaration, function body. TC++PL 2.3, 7, D&E 2.6.

function argument - an argument to a function.

function body - the outermost block of a function. See also: try-block, function definition. TC++PL 2.7, 13.

function declaration - declaration of a function, including its name, argument types, and return type.

function definition - function declaration including a function body.

function member - see member function.

function object - object with the application operator, operator()(), defined so that it can be called like a function. A function object is more general than a function because it can hold data and provide additional operations. Sometimes called a functor. Given current compiler technology, simple function objects inline better than pointers to functions, so that parameterization with function objects can be far more efficient than use of pointers to functions or virtual functions. See also: binder, adapter, inlining. Example. TC++PL 18.4.

function parameter - a parameter of a function.

function prototype - C term for a function declaration that isn't also a function definition. D&E 2.6.

function template - see template function.

function try-block - try-block associated with the outmost block of a function, the function body. TC++PL 3.7.2.

functor - see function object.

G++ - see GNU C++.

garbage collection - techniques for reclaiming unused memory without relying on user-supplied delete or free() commands. A permitted but not required technique for C++. Commercial and free garbage collectors exist for C++: See my C++ page. Use of classes that control their own storage, such as the standard library vector, string, and map, reduces the need for garbage collection. See also: resource acquisition is initialization, destructor. TC++PL C.9.1. D&E 10.7.

general-purpose programming language - (1) a programming language intended for use in a wide range of application areas without restrictions that make it totally unsuitable for traditional major uses of computers, such as mathematical computations, data processing, text processing, graphics, and communications. (2) a language that can do what at least as much as other languages called "general purpose" can do. See also: C++.

generic programming - programming using templates to express algorithms and data structures parameterized by data types, operations, and polices. See also: polymorphism, multi-paradigm programming. TC++PL 2.7, 24.4.1, D&E 15.11.2.

get function - see >>.

global data - data defined in the global scope. This is usually best avoided because a programmer can't easily know what code manipulates it and how. It is therefore a common source of errors. Global constants are usually ok.

global scope - the scope containing all names defined outside any function, class, or namespace. Names in the global scope can be prefixed by ::. For example, ::main(). TC++PL 2.9.4.

glossary - "collection of glosses; lists and explanations of special words." - The Advanced Learners Dictionary of Current English. A pain to compile.

GNU C++ - GNU's implementation of C++.

goto - the infamous goto. Primarily useful in machine generated C++ code.

TC++PL 6.3.4.

grammar - a systematic description of the syntax of a language. The C++ grammar is large and rather messy. Some of the syntactic complexity was inherited from C. TC++PL A, D&E 2.8.

GUI - Graphical User Interface. There are many C++ libraries and tools for building GUI-based applications, but no standard C++ GUI.

handle - an object that controls access to another. Often, a handle also controls the acquisition and release of resources. A common use is for a handle to control access to a variably-sized data structure. See also: resource acquisition is initialization, vector, string, smart pointer. TC++PL 25.7, D&E 11.5.2.

handle class - a small class that provides interface to an object of another class. A handle is the standard way of providing variable sized data structures in C++. Examples are string and vector. TC++PL 25.7.

handler - see exception handler

hash_map - hashed contained based on the standard library framework. Not (yet) part of the standard but very common in libraries based on the standard library. See also: map, vector, list. TC++PL 17.6.

header - see header file

header file - file holding declarations used in more than one translation unit. Thus, a header file acts as an interface between separately compiled parts of a program. A header file often contains inline function definitions, const definitions, enumerations, and template definitions, but it cannot be #included from for than one source file if it contain non-inline function definitions or variable definitions. TC++PL 2.4.1, 9.2.1. D&E 2.5, 11.3.3.

hiding - see information hiding

hierarchy - see class hierarchy.

higher-order function - functions that produce other functions. C++ does not have general higher-order functions, but by returning function objects a function can efficiently emulate some techniques traditionally relying of higher-order functions. See also: binder. TC++PL 18.4.4.

history of C++ - The work on what became C++ started by Bjarne Stroustrup in AT&T Bell Labs in 1979. The first commercial release was in 1985. Standards work stared in 1990 leading to ratification of the ISO standard in 1998. TC++PL 1.4. D&E Part 1.

Hungarian notation - a coding convention that encodes type information in

variable names. Its main use is to compensate for lack of type checking in weakly-typed or untyped languages. It is totally unsutable for C++ where it complicates maintenance and gets in the way of abstraction.

hybrid language - derogative term for a programming language that supports more programming styles (paradigms) rather than just object-oriented programming.

I/O - see iostream

IDE - Integrated (or Interactive) Development Enviornment. A software development environment (SDE) emphasizing a GUI interface centered around a source code editor. There are many IDEs for C++, but no standard SDE.

identifier - see name.

if-statement - statement selecting between two alternatives based on a condition. TC++PL 6.3.2.

ifstream - an file stream for input.

implementation defined - an aspect of C++'s semantics that is defined for each implementation rather than specified in the standard for every implementation. An example is the size of an int (which must be at least 16 bits but can be longer). Avoid implementation defined behavior whenever possible. See also: undefined. TC++PL C.2.

implementation inheritance - see private base.

implicit type conversion - conversion applied implicitly based on an expected type and the type of a value. See also, explicit type conversion, user-defined type conversion. TC++PL 11.3.3, 11.3.5, 11.4, C.6, D&E 2.6.2, 3.6.1, 3.6.3, 11.2.

in-class - lexically within the declaration of a class. TC++PL 10.2.9, 10.4.6.2.

include - see #include.

incomplete type - type that allows an object to be copied, but not otherwise used. A pointer to an undeclared type is the typical example of an incomplete type.

inequality operator - see !=.

infix operator - a binary operator where the operator appears between the operands. For example, a+b.

information hiding - placing information where it can be accessed only

through a well-defined interface. See also: access control, abstract class, separate compilation. TC++PL 2.4.

inheritance - a derived class is said to inherit the members of its base classes. TC++PL 2.6.2, 12.2, 23.4.3.1, D&E 3.5, 7.2, 12.

initialization - giving an object an initial value. Initialization differs from assignment in that there is no previous value involved. Initialization is done by constructors.

initializer list - comma-separated list of expressions enclosed in curly braces, e.g. { 1, 2, 3 } used to initialize a struct or an array. TC++PL 5.2.1, 5.7, 11.3.3.

inline function - function declared inline using the inline keyword or by being a member function defined in-class. Compilers are encouraged to generate inline code rather than function calls for inline functions. Most benefits from inlining comes with very short functions. TC++PL 7.1.1, 9.2, 10.2.9, D&E 2.4.1 .

inlining - see inline function.

input - see iostream.

inserter - (1) an iostream << (put to) function. (2) an STL operation yielding an iterator to be used for adding elements to a containter. TC++PL 19.2.4, 21.2, D&E 8.3.1. See also: extracter, back_inserter, front_inserter.

instantiation - see template instantiation.

int - basic signed integer type; its precision is implementation-defined, but an int has at least 32 bits. TC++PL 2.3.1, 4.4.

integer type - a short, int, or long. Standard C++ doesn't support long long. TC++PL 4.4.

integral type - a bool, character type, or integer type. Supports arithmetic and logical operations. TC++PL 4.1.1.

interface - a set of declarations that defines how a part of a program can be accessed. The public members and the friends of a class defines that class' interface for other code to use. A class without data members defines a pure interface. The protected members provide an additional interface for use by members of derived classes. See also: abstract class.

interface function - A function that can access the representation of a class. See also: friend, member function, derived class, protected.

interface inheritance - see abstract class, public base.

invariant - a condition of the representation of an object (the object's state) that should hold each time an interface function is called; usually established by a constructor TC++PL 24.3.7, E.3.5.

iostream - (1) standard library flexible, extensible, type-safe input and output framework. (1) stream that can be used for both input and output. See also: file stream, string stream. TC++PL 3.4, 3.6, 21, D&E 3.11.4.1, 8.3.1.

ISO - the international standards organization. It defines and maintains the standards of the major non-proprietary programming languages, notably C++.

ISO C - see C.

ISO C++ - C++.

istream - input stream type. TC++PL 3.6, 21.3.

istringstream - a string stream for input.

iteration - traversal of data structure, directly or indirectly using an iteration-statement. See also: recursion. The standard library offer algorithms, such as copy() and find(), that can be effective alternatives to explicit iteration. TC++PL 6.3.3. 18.

iteration-statement - for-statement, while-statement, or do-statement.

iterator - a standard library abstraction for objects referring to elements of a sequence. TC++PL 3.8.1, 19.2-3.

K&R C - C as defined by Kernighan and Ritchie.

Kernighan - Brian Kernighan is a co-author of Kernighan & Ritchie: "The C programming Language".

Koenig lookup - see argument-based lookup.

language extension - (1) relatively new feature that people haven't yet gotten used to. (2) proposed new feature. (3) feature provided by one or more implementations, but not adopted by the standard; the use of some such features implies lock-in to a particular compiler supplier.

learning C++ - focus on concepts and techniques. You don't need to learn C first. See also "Learning Standard C++ as a New Language", available from my papers page. How do I start?. TC++PL 1.2, 1.7, D&E 7.2.

Library TR - technical report from the ISO C++ standards committee defining a set of new standard library components, including regular expression matching (regexp), hashedcontainers (ordered_map), and smart pointers. See my C++ page.

line comment - comment started by // and terminated by end-of-line. TC++PL 6.4, D&E 3.11.1.

linkage - the process of merging code from separately compiled translation units into a program or part of a program. TC++PL 9.

linker - the part of a C++ implementation that merge the code generated from separately compiled translation units into a program or part of a program. TC++PL 9.1, D&E 4.5, 11.3.

Liskov Substitution Principle - design classes so that any derived class will be acceptable where its base class is. C++ public bases enforce that as far as the interface provided by the base class. TC++PL 24.3.4, D&E 2.10.

list - standard library linked container. See also: vector, map. TC++PL 3.7.3, 17.2.2.

literal - notation for values of bool, character types, integer types, or floating-point types. See also: enumerators. TC++PL 4.2, 4.3.1, 4.4.1, 4.5.1, 5.2.2, D&E 11.2.1.

local class - class defined within a function. Most often, the use of a local class is a sign that a function is too large. Beware that a local class cannot be a valid template argument.

local function - function defined within a function. Not supported by C++. Most often, the use of a local function is a sign that a function is too large.

locale - standard library class for representing culture dependencies relating to input and output, such as floating-point output formats, character sets, and collating rules. A locale is a container of facets. TC++PL 21.1, D.

long double - extended-precision floating-point number. TC++PL 4.5.

long int - integer of a size greater than or equal to the size of an int. TC++PL 4.4.

loop - a statement that expresses the notion of doing something zero or more times, such as a for-statement and a while-statement.

LSP - see Liskov Substitution Principle.

lvalue - an expression that may appear on the left-hand side of an assignment; for example, v[7] if v is an array or a vector. An lvalue is modifiable unless it is const. TC++PL 4.9.6, D&E 3.7.1.

macro - facility for character substitution; doesn't obey C++ scope or type rules. C++ provides alternatives to most uses of macros; see template, inline, const, and namespace. Don't use macros unless you absolutely have to. TC++PL 7.8, D&E 2.9.2, 4.4, 18.

main() - the function called by the system to start a C++ program. TC++PL 3.2, 6.1.7, 9.4 .

maintenance - work on a program after its initial release. Typical maintenance activities include bug fixing, minor feature enhancements, porting to new systems, improvements of error handling, modification to use different natural languages, improvements to documentation, and performance tuning. Maintenance typically consumes more than 80% of the total effort and cost expended on a program.

malloc() - C standard allocation function. Use new or vector instead.

map - standard library associative container, based on "less than" ordering. See also: hash_map, vector, list. TC++PL 3.7.4, 17.4.1.

Max Munch - (1) mythical participant in the C++ standards process. (2) the rule that says that while parsing C++ always chooses the lexically or syntactically longest alternative. Thus ++ is the increment operation, not two additions, and long int is a single integer type rather than the long integer followed by an int. Cross references in this glossary follow this rule.

member - type, variable, constant, or function declared in the scope of a class. TC++PL 5.7, 10.2, D&E 2.3, 2.5.2, 2.11.

member class - a class that is a member of another; also called a nested class. TC++PL 11.12, D&E 3.12, 13.5.

member constant - const or enumeration declared as a member. If initialized in-class, such a constant can be used in constant expressions within the class. TC++PL 10.4.6.2.

member data - see data member.

member function - a function declared in the scope of a class. A member function that is not a static member function must be called for an object of its class. TC++PL 10.2.1, D&E 2.3, 3.5.

member initializer - initializer for a member specified in the constructor for its class. TC++PL 10.4.6, 12.2.2, D&E 12.9.

member type - member class, member enumeration, or member typedef.

memberwise copy - copying a class object by copying each of its members in turn, using proper copy constructors or copy assignments. That's the default meaning of copy. TC++PL 10.4.4.1, 10.4.6.3, D&E 11.4.4.

memory - static memory, stack, or free store.

memory management - a way of allocating and freeing memory. In C++ memory is either static, allocated on the stack, or allocated on the free store.

When people talk about memory management, they usually think of free store or even specifically about garbage collection. Memory can often be effectively managed through standard library containers, such as vector or string, or through general resource management techniques. See also: auto_ptr, constructor, destructor, resource acquisition is initialization. TC++PL C.9, D&E 3.9, 10.

mem_fun() - an adapter that allows a member function to be used as an argument to a standard algorithm requiring a free-standing function. TC++PL 18.4.4.2.

method - see virtual member function.

Microsoft C++ - see Visual C++

modifiable lvalue - lvalue that is not const. TC++PL 4.9.6.

most-derived type - the type used to create an object (before any conversions). See also: dynamic type, static type.

multi-method - a virtual function that selects the function to be called based on more than one operand. See also: multiple dispatch. D&E 13.8.

multi-paradigm design - design focussed on applying the various paradigms to their best advantage. See also: multi-paradigm programming.

multi-paradigm programming - programming applying different styles of programming, such as object-oriented programming and generic programming where they are most appropriate. In particular, programming using combinations of different programming styles (paradigms) to express code more clearly than is possible using only one style. See also: C++.

multimap - map that allows multiple values for a key. TC++PL 17.4.2.

multiple dispatch - the generalization of double dispatch to more operands. See also: single dispatch.

multiple inheritance - the use of more than one immediate base class for a derived class. One typical use is to have one base define an interface and another providing help for the implementation. TC++PL 12.2.4, 12.4, 15.2.5, D&E 12.

mutable - an attribute of a member that makes it possible to change its value even if its object is declared to be const TC++PL 10.2.7.2, D&E 13.3.3.

name - sequence of letters and digits started by a letter, used to identify ("name") user-defined entities in program text. An underscore is considered a letter. Names are case sensitive. The standard imposes no upper limit on the length of names. TC++PL 4.9.3.

namespace - a named scope. TC++PL 2.5.1, 8.1, C.10. D&E 17.

namespace alias - alternative name for a namespace; often a shorter name. TC++PL 8.2.7, D&E 17.4.3.

NCITS - National Committee for Information Technology Standards. The part of ANSI that deals with programming language standards, notably C++, and sells copies of the C++ standard. Formerly known as X3.

nested class - see member class.

nested function - see local function.

new - object creation operator. See also: constructor, placement new, operator new(), resource management, memory management, garbage collection. TC++PL 6.2.6, 19.4.5, D&E 2.3, 10.2.

new-style cast - dynamic_cast, static_cast, const_cast, or reinterpret_cast. D&E 14.3.

new_handler - a (possibly user-defined) function called by new if operator new() fails to allocate sufficient memory. See also: std::bad_alloc exception. TC++PL 6.2.6.2, 14.4.5., 19.4.5.

non-static member - member of a class that is not declared to be a static member. An object of a class has its own space for each non-static data member.

not - synonym for !, the logical negation operator TC++PL C.3.1.

nothrow guarantee - the guarantee that an operation will not throw an exception. See also exception safety, basic guarantee, and strong guarantee. TC++PL E.2.

NULL - zero. 0. 0 is an integer. 0 can be implicitly converted to every pointer type. See also: nullptr. TC++PL 5.1.1, D&E 11.2.3.

nullptr - C++0x keyword for the null pointer. It is not an integer. It can be assigned only to pointers.

object - (1) a contiguous region of memory holding a value of some type. (2) a named or unnamed variable of some type; an object of a type with a constructor is not considered an object before the constructor has completed and is no longer considered an object once a destructor has started executing for it. Objects can be allocated in static memory, on the stack, on on the free store. TC++PL 4.9.6, 10.4, 10.4.3, D&E 2.3, 3.9.

object code - see object file.

object file - the result of compiling a source file. See also: compiler.

object-oriented design - design focussed on objects and object-oriented programming. TC++PL 23.2, D&E 7.2.

object-oriented programming - programming using class hierarchies and virtual functions to allow manipulation of objects of a variety of types through well-defined interfaces and allow a program to be extended incrementally through derivation. See also: polymorphism, data abstraction. TC++PL 2.6, 12, D&E 3.5, 7.2.

object-oriented programming language - a programming language designed to support or enforce some notion of object-oriented programming. C++ supports OOP and other effective forms of programming, but does not try to enforce a single style of programming. See also: generic programming, multi-paradigm programming, hybrid language.

ODR - see one definition rule

ofstream - an file stream for output.

old-style cast - see C-style cast.

one definition rule - there must be exactly one definition of each entity in a program. If more than one definition appears, say because of replication through header files, the meaning of all such duplicates must be identical. TC++PL 9.2.3, D&E 2.5, 15.10.2.

OOD - see object-oriented design.

OOP - see object-oriented programming.

OOPL - see object-oriented programming language.

operator - conventional notation for built-in operation, such as +, *, and &. A programmer can define meanings for operators for user-defined types. See also: operator overloading, unary operator, binary operator, ternary operator, prefix operator, postfix operator. TC++PL 6.2.

operator delete() - deallocation function used by delete#. Possibly defined by user. TC++PL 6.2.6.2, 19.4.5. See also: operator new().

operator delete[]() - deallocation function used by delete#. Possibly defined by user. TC++PL 6.2.6.2, 19.4.5. See also: operator new[]().

operator function - function defining one of the standard operators; e.g. operator+(). See also: operator, operator overloading, conversion operator.

operator new() - allocation function used by new. Possibly defined by user. TC++PL 6.2.6.2, 19.4.5. See also: operator delete().

operator new[]() - allocation function used by new. Possibly defined by user.

TC++PL 6.2.6.2, 19.4.5. See also: operator delete[]().

operator overloading - having more than one operator with the same name in the same scope. Built-in operators, such as + and *, are overloaded for types such as int and float. Users can define their own additional meanings for user-defined types. It is not possible to define new operators or to give new meanings to operators for built-in types. The compiler picks the operator to be used based on argument types based overload resolution rules. See also: overload resolution. TC++PL 6.2, D&E 3.6, 11.7.1.

optimizer - a part of a compiler that eliminates redundant operations from code and adjusts code to perform better on a given computer. See also, front-end, back-end, code generator. D&E 3.3.3.

or - synonym for ||, the logical or operator TC++PL C.3.1.

order of construction - a class object is constructed from the bottom up: first bases in declaration order, then members in declaration order, and finally the body of the constructor itself. TC++PL 10.4.6, 12.2.2, 15.2.4.1, 15.4.3. D&E 2.11.1, 13.2.4.2.

order of destruction - a class object is destroyed in the reverse order of construction. See also: destructor.

ostream - output stream type. TC++PL 3.4, 21.2.

ostringstream - a string stream for output.

output - see iostream.

out_of_range - standard exception thrown by vector if an argument to at() is out of range. TC++PL 16.3.3.

overload - see overloading.

overload resolution - a set of rules for selecting the best version of an operator based on the types of its operands. A set of rules for selecting the best version of an overloaded function based on the types of its arguments. The intent of the overload resolution rules is to reject ambiguous uses and to select the simplest function or operator for each use. TC++PL 6.2, D&E 11.2.

overloaded function - see overloading.

overloaded operator - see operator overloading

overloading - having more than one function with the same name in the same scope or having more than one operator with the same name in the same scope. It is not possible to overload across different scopes. See also: using-declaration. TC++PL 6.2, D&E 3.6, 11.2.

override - see overriding.

overriding - declaring a function in a derived class with the same name and a matching type as a virtual function in a base class. The argument types must match exactly. The return types must match exactly or be co-variant. The overriding function will be invoked when the virtual function is called. TC++PL 15.6.2, 6.2, D&E 3.5.2-3, 13.7.

paradigm - pretentious and overused term for a way of thinking. Often used with the erroneous assumption that "paradigms" are mutually exclusive, and often assuming that one paradigm is inherently superior to all others. Derived from Kuhn's theory of science. TC++PL 2.2.

parameter - a variable declared in a function or templates for representing an argument. Also called a formal argument. Similarly, for templates.

partial specialization - a template used (only) for the subset of its template parameters that matches a specialization pattern. TC++PL 13.5.

Performance TR - technical report from the ISO C++ standards committee discussing issues related to perfoemance, especially as concerns embedded systems programming and hardware access. See my C++ page.

placement delete - See explicit call of destructor.

placement new - a version of the new operator where the user can add arguments to guide allocation. The simplest form, where the object is placed in a specific location, is supported by the standard library. Example. For example, placement new is used in the implementation of standard library containers. See also: explicit call of destructor. TC++PL 10.4.11, E.3.1, D&E 10.4.

POD - "Plain Old Data" - (roughly) a class that doesn't contain data members that would be illegal in C. A POD can therefore be used for data that needs to be share with C functions. A POD can have non-virtual member functions.

pointer - an object holding an address or 0. TC++PL 2.3.3, 5.1, D&E 9.2.2.1, 11.4.4.

policy object - an object used to specify guide decisions (e.g. the meaning of "less than") or implementation details (e.g. how to access memory) for an object or an algorithm. See also trait, facet. TC++PL 13.4, 24.4.1.

polymorphism - providing a single interface to entities of different types. virtual functions provide dynamic (run-time) polymorphism through an interface provided by a base class. Overloaded functions and templates provide static (compile-time) polymorphism. TC++PL 12.2.6, 13.6.1, D&E 2.9.

postfix operator - a unary operator that appears after its operand. For example var++.

prefix operator - a unary operato that appears before its operand. For example, &var.

preprocessor - the part of a C++ implementation that removes comments, performs macro substitution and #includes. Avoid using the preprocessor whenever possible. See also: macro, #include, inline, const, template, namespace. TC++PL 7.8, 9.2.1, D&E 18.

priority_queue - standard library queue where a priority determines the order in which an element reaches the head of the queue. TC++PL 17.3.3.

private - access control keyword. See private member, private base.

private base - a base class declared private in a derived class, so that the base's public members are accessible only from that derived class. TC++PL 15.3.2, D&E 2.10.

private member - a member accessible only from its own class. TC++PL 2.5.2, 10.2.2, 15.3, D&E 2.10.

procedural programming - programming using procedures (functions) and data structures (structs). See also: data abstraction, object-oriented programming, generic programming, multi-paradigm programming. TC++PL 2.3.

procedure - see function.

program - a set of translation units complete enough to be made executable by a linker. TC++PL 9.4.

programming language - artificial language for expressing concepts and general algorithms in a way that lends itself to solving problems using computers. There do not appear to be a general consensus on what a programming language is or should be. TC++PL 1.3.2, 2.1-2, D&E page 7.

prohibiting operations - operations can be rendered inaccessible by declaring them private; in this way default operations, such as construction, destruction, and copying can be disallowed for a class. TC++PL 11.2.2, D&E 11.4.

proprietary language - language owned by an organization that is not an official standards organization, such as ISO; usually manipulated by its owner for commercial advantage.

protected - access control keyword. See protected member, protected base.

protected base - a base class declared protected in a derived class, so that

the base's public and protected members are accessible only in that derived class and classes derived from that. TC++PL 15.3.2, D&E 13.9.

protected member - a member accessible only from classes derived from its class. TC++PL 15.3.1, D&E 13.9.

protection - see encapsulation.

protection model - the mechanisms for access control. See public, private, protected, friend. TC++PL 15.3, D&E 2.10.

public - access control keyword. See public member, public base.

public base - a base class declared public in a derived class, so that the base's public members are accessible to the users of that derived class. TC++PL 15.3.2, D&E 2.3.

public member - a member accessible to all users of a class. TC++PL 2.5.2, 10.2.2, 15.3, D&E 2.10.

pure object-oriented language - programming language claiming to support only object-oriented programming. C++ is designed to support several programming paradigms, including traditional C-style programming, data abstraction, object-oriented programming, and generic programming. For a longer explanation, read Why C++ isn't just an object-oriented programming language. See also: hybrid language.

pure virtual function - virtual function that must be overridden in a derived class. Indicated by the curious =0 syntax. A pure virtual function can be defined in the class where it is declared pure, but needn't be and usually isn't. A class with at least one pure virtual function is an abstract class. TC++PL 12.3. D&E 13.2.1.

push_back() - member function that adds an element at the end of a standard container, such as vector, thereby increasing the container's size by one. Example. TC++PL 3.7.3, 16.3.5, E.3.4.

put function - see <<.

qualified name - name qualified by the name of its enclosing class or namespace using the scope resolution operator ::. For example, std::vector or ::main. TC++PL 4.9.3, 8.2.1, 10.2.4, 15.2.1, 15.2.2, D&E 3.11.3.

queue - standard library first-in-first-out sequence. TC++PL 17.3.2.

RAII - see resource acquisition is initialization.

random number generator - function or function object producing a series of pseudorandom numbers according to some distribution. TC++PL 22.7.

raw memory - see uninitialized memory.

realloc() - C standard allocation function. Use vector and push_back() instead.

recursion - a function calling itself, hopefully with different arguments so that the recursion eventually ends with a call for which the function doesn't call itself. See also: iteration. TC++PL 7.1.1.

reference - an alternative name for an object or a function. See also: operator overloading, call-by-reference. TC++PL 5.4.1, D&E 3.7.

regression testing - systematically checking that a new version of a program doesn't break correct uses of a previous version of the program.

reinterpret_cast - a type conversion operation that reinterprets the raw memory of an object as a value of another type. The result of a reinterpret_cast can only be portably used after being converted back into its original type. Use only as a last resort. See also: cast. TC++PL 6.2.7, D&E 14.3.3.

resource - any entity that a program acquires and releases. Typical examples are free store, file handles, threads, sockets. See also: resource acquisition is initialization, exception safety, basic guarantee, resource management. TC++PL 14.4, E.2-3 D&E 16.5.

resource acquisition is initialization - A simple technique for handling resources in programs using exceptions. One of the keys to exception safety. Example. TC++PL 14.4, E.3 D&E 16.5.

resource leak - programming error causing a resource not to be released. See also: resource acquisition is initialization, basic guarantee. TC++PL 14.4, E.2-3 D&E 16.5.

resource management - a way of acquiring and releasing a resource, such as memory, thread, or file. See also: resource acquisition is initialization, auto_ptr, vector. TC++PL 14.4, D&E 10.4.

resumption semantics - In some languages, but not C++, an exception handler can respond by telling the thrower to resume (``just carry on as if the problem hadn't happened"). This looks like a good idea in some cases, but in general leads to contorted code because of unfortunate dependencies between separate levels of abstraction. See also: termination semantics. TC++PL 14.4.5, D&E 16.6.

return type relaxation - Allowing a virtual function returning a B* or a B& to be overridden by a function with a return type D* or D&, provided B is a public base of D. See also: overriding. TC++PL 15.6.2, D&E 13.7.

reverse iterator - iterator for iterating through a sequence in reverse order. TC++PL 19.2.5.

Ritchie - Dennis Ritchie is the designer and original implementer of C. Co-author of Kernighan & Ritchie: "The C programming Language".

RTFM - "Read The Manual" (The 'F' is silent). Usually a very good idea.

RTTI - see Run Time Type Information.

run time type information - information about a type available at run time through operations on an object of that type. See also: dynamic_cast, typeid(), and type_info. TC++PL 15.4, D&E 14.2.

rvalue - an expression that may appear on the right-hand side of an assignment, but not of the left-hand side; for example, 7. D&E 3.7.1.

scope - a region of source text delimited by curly braces: { ... }, a list of function or template parameters, or all of a translation unit outside other scopes. See also: block, namespace, global scope. TC++PL 2.9.4.

SDE - Software Development Environment. An environment of editors, compilers, tools, libraries, etc. used by a programmer to produce software. There are many SDEs for C++, but no standard SDE.

selection-statement - if-statement or switch-statement. TC++PL 6.3.2.

self - see this.

semantics - the rules specifying the meaning of a syntactically correct construct of a program. For example, specifying the actions taken to perform a for-statement or an object definition.

separate compilation - the practice of compiling parts of a program, called translation units, separately and then later linking the results together using a linker. This is essential for larger programs. See also: linkage, header file, one definition rule. TC++PL 2.4.1, 9.1. D&E 2.5.

separately compiled - see separate compilation.

sequence adapter - a class that provides a modified interface to another. For example, a standard library stack is an adapter for a more flexible data structure such as a vector. See also: adapter, stack, queue, priority_queue. TC++PL 17.3.

set - standard library associative container

short - integer of a size less than or equal to the size of an int. TC++PL 4.4.

sibling class - two classes are siblings if a class is (directly or indirectly)

derived from them both and one is not derived from the other. Note that this is a rather inclusive definition of "sibling class" in that is does not require that the siblings have the same immediate derived class (I didn't want to introduce a notion of "cousin classes"). See also: dynamic_cast, crosscast.

signature - the set of parameter types for a function; that is, the function's type ignoring its return type. This is a confusingly specialized definition compared to other programming languages where "signature" means "function type".

Simula - ancestor of C++ designed by Ole-Johan Dahl and Kristen Nygaard; the source of the C++ class concept. TC++PL 1.4, 2.6.2, D&E 1.1, 3.1.

single dispatch - the technique of choosing the member function to be invoked based on the object used in the call. See also: double dispatch.

size of an object - the number of bytes required to represent an object. See also sizeof, alignment. TC++PL 4.6.

sizeof - operator yielding the size of an object.

smart pointer - user-defined type providing operators like a function, such as * and ++, and with a semantics similar to pointers. See also: iterator. Sometimes smart a pointer is called a handle. TC++PL 11.10-11, 13.6.3.1, 19.3, 25.7, D&E 11.5.1

software - a collection of programs

sort() - standard library algorithm for sorting a random access sequence, such as a vector or an array. Example comparing sort() to qsort(). TC++PL 18.7.1.

source file - .c file or header.

specialization - a class or function generated from a template by supplying a complete set of template arguments. TC++PL 13.2.2, 13.5, D&E 15.10.3.

stack - (1) memory used to hold local variables for a function. (2) standard library first-in-last-out sequence. TC++PL 10.4.3, 17.3.1, D&E 2.3, 3.9.

Standard C++ - C++ as defined by ISO.

standard header - header for standard library facility. Included using the "#include< ... >" syntax. TC++PL 9.2.2, 16.1.2.

standard library - The library defined in the C++ standard. Contains strings, stream I/O, a framework of containers and algorithms, support for numerical computation, support for internationalization, the C standard library, and some language support facilities. See also: complex, valarray, locale. TC++PL 16-22, D, E.

standards committee - see C++ standards committees.

statement - the basic unit controlling the execution flow in a function, such as if-statement, while-statement, do-statement, switch-statement, expression statement, and declaration. TC++PL 6.3.

static - (1) keyword used to declare a class member static; meaning allocated in static memory. For a member function, this implies that there is no this pointer. (2) keyword used to specify that a local variable should be allocated in static memory. (3) deprecated: keyword used to specify that a global name should not be visible from other translation units. TC++PL 7.1.2, 10.2.4, 10.4.8-9.

static member - member of a class for which there is only one copy for the whole program rather than one per object. TC++PL 10.2.4, D&E 13.4.

static member function - a member function that need not be called for an object of the class. TC++PL 10.2.4, D&E 13.4.

static memory - memory allocated by the linker. TC++PL 10.4.3, D&E 2.3, 2.11.1, 3.9, 11.4.2.

static type - the type of an object as known to the compiler based on its declaration. See also: dynamic type.

static type safety - type safety enforced before a program starts executing (at compile time or at static link time).

static variable - variable allocated in static memory. TC++PL 7.1.2, 10.2.4, 10.4.3, D&E 3.9.

static_cast - a type conversion operation that converts between related types, such as pointer types within a class hierarchy and between enumerations and integral types. See also: cast, dynamic_cast. TC++PL 6.2.7, 15.4.2.1, D&E 14.3.2.

Stepanov - Alex Stepanov is the original designer and implementer of the STL. D&E 11.15.2.

STL - the "Standard Template Library" by Alex Stepanov, which became the basis for the containers, algorithms, and iterators part of the ISO C++ standard library. TC++PL 15-19.

strcmp() - a C-style standard library function for comparing C-style strings.

stream - see iostream.

stream I/O - see iostream.

string - standard-library type representing a sequence of characters, support

by convenient operators, such as == and +=. The general form of of strings, basic_string, supports strings of different kinds of characters. TC++PL 3.5, 20.

string stream - stream attached to a string. See also, stringstream, istringstream, ostringstream. TC++PL 21.5.3.

stringstream - a string stream for input and output.

strong guarantee - the guarantee that an exception thrown by an operation leaves every object in the state in which it was before the start of the operation. Builds on the basic guarantee. See also exception safety, nothrow guarantee, and basic guarantee. TC++PL E.2.

Stroustrup - see Bjarne Stroustrup.

strstream - deprecated ancestor of stringstream.

struct - class with members public by default. Most often used for data structures without member functions or class invariants, as in C-style programming. TC++PL 5.7, 10.2.8, D&E 3.5.1.

subclass - a derived class.

subtype - see derived class. See also: public base.

suffix operator - a postfix operator.

superclass - a base class.

switch-statement - statement selecting among many alternatives based on an integer value. TC++PL 6.3.2.

syntax - the set of gramatical rules specifying how the text of a program must be composed. For example, specifying the form of a declaration or the form of a for-statement.

TC++PL - Bjarne Stroustrup: The C++ Programming Language (Special Edition). Addison Wesley. 2000.

template - class or function parameterized by a set of types, values, or templates. See also template instantiation, specialization, template class, template function. TC++PL 2.7, 13, D&E 15.

template argument - an argument to a template.

template argument constraint - see constraint.

template class - class parameterized by types, values, or templates. The template arguments necessary to identify the class to be generated for the

class template must be provided where a template class is used. For example "vector<int> v;" generates a vector of ints from the vector template. See also template. TC++PL 13.2, D&E 15.3.

template definition - declaration of a template class or of a template function including a function body.

template function - function parameterized by types, values, or templates. The function to be generated from a template function can usually be deduced from the function arguments in a call. For example, "sort(b,e)" generates "sort<vector::iterator>(b,e)" from the sort() template function if b and e are standard library vector iterators. If a template argument cannot be deduced, it must be provided through explicit qualification. See also template. TC++PL 13,3, D&E 15.6.

template instantiation - the process of creating a specialization from a template. TC++PL 13.2.2, D&E 15.10.

template parameter - a parameter of a template.

terminate() - If an exception is thrown but no handler is found, terminate() is called. By default, terminate() terminates the program. If program termination is unacceptable, a user can provide an alternative terminate() function. If you are worried about uncaught exceptions, make the body of main() a try-block. TC++PL 14.7.

termination semantics - a somewhat ominous terminology for the idea that throwing an exception "terminates" an operation and returns through the function call chain to a handler. The handler can initiate any error handling it likes, including calling the function that caused the exception again (presumably after fixing the problem that caused the problem). What a handler can't do is simply tell the thrower to just carry on; by the time the handler is invoked we have returned from the block/function that threw and all blocks/functions that led to it from the handler's try-block. See also: resumption semantics. TC++PL 14.4.5, D&E 16.6.

ternary operator - an operator taking three operands, such as ?:.

testing - systematically verifying that a program meets its specification and systematically searching for error.

this - pointer to the object for which a non-static member function is called. TC++PL 10.2.7, D&E 2.5.2.

throw - operation for interrupting the normal flow of control and returning to an appropriate exception handler identifyed by the type of the exception throw. See also: catch, exception handling. TC++PL 8.3.1, 14.3, D&E 16.3.

trait - a small policy object, typically used to describe aspects of a type. For

example, iterator_trait specifies the types resulting from operations on an iterator T. TC++PL 19.2.2.

translation unit - a part of a program that can be separately compiled. TC++PL 9.1.

trigraph - alternative representation for C++ representation characters that doesn't exist in every national character set, such as {, }, [,], and #: ??<, ??>, ??(, ??), and ??=. TC++PL C.3.1.

true - bool value; converts to 1. TC++PL 4.2, D&E 11.7.2.

try - keyword used to start a try-block.

try-block - a block, prefixed by the keyword try, specifying handlers for exceptions. See also: catch, exception handling. TC++PL 8.3.1,14.3, D&E 16.3.

two-phase lookup - a somewhat complicated mechanism used in compilation of templates. Names that do not depend on a template parameter are looked up (and bound) early, i.e., when the template template definition is first seen ("phase 1 lookup"). Names that depend on a template parameter are looked up late, i.e. during template instantiation ("phase 2 lookup") so that the lookup can find names relating to actual template arguments. TC++PL C::13.8.

type - a built-in type or a user-defined type. A type defines the proper use of a name or an expression. TC++PL 2.3.1, 4.1.

type checking - the process of checking that every expression is used according to its type. the compiler checks every expression based on the declared types of the names involved. TC++PL 7.2-3, 24.2.3, D&E 2.3, 2.6, 3.10, 3.15, 9.2.2.1.

type conversion - producing a value of one type from a value of another type. A type conversion can be an implicit conversion or an explicit conversion. See also: user-defined type conversion, cast. TC++PL 6.2.7.

type safety - the property that an object can be accessed only according to its definition. C++ approximates this ideal. A programmer can violate type safety by explicitly using a cast, by using an uninitialized variable, by using a pointer that doesn't point to an object, by accessing beyond the end of an array, and by misusing a union. For low-level systems code, it can be necessary to violate type safety (e.g. to write out the byte representation of some objects), but generally type safety must be preserved for a program to be correct and maintainable.

type system - the set of rules for how objects can be used according to their types. See also: type checking.

typedef - synonym for some type declared using the keyword **typedef**.

typeid() - operator returning basic type information. TC++PL 15.4.4, D&E 14.2.5.

typename - (1) an alternative to "class" when declaring template arguments; for example, "template<typename T> void f(T);" (2) a way of telling a compiler that a name is meant to name a type in template code; for example "template<class T> void f(T a) { typename T::diff_type x = 0; ... }". TC++PL C::13.5.

type_info - class containing basic run time type information. TC++PL 15.4.4, D&E 14.2.5.1.

unary operator - an operator taking one operand, such as ! and unary *.

uncaught exception - Exception for which no handler was found. Invokes terminate(), which by default terminates the program. TC++PL 14.7.

undefined - an aspect of C++'s semantics for which no reasonable behavior is required. An example is dereferencing a pointer with the value zero. Avoid undefined behavior. See also: implementation defined. TC++PL C.2.

uninitialized memory - memory that hasn't been initialized to hold a specific value of a type. TC++PL 19.4.4.

union - a struct with all members allocated at the same offset within an object. The language does not guarantee type safety for all uses of unions. Primarily used to save space. TC++PL C.8.2.

upcast - a cast from a derived class to one of its bases. See also: downcast, crosscast. TC++PL 15.4.

user-defined type - Class or enumeration. A programmer can define meanings for operators for user-defined types. See also: operator overloading. TC++PL 6.2, 11, D&E 3.6, 11.7.1.

user-defined type conversion - a user can define conversions either as constructors or conversion operators. These conversions are applied explicitly or implicitly just like built-in conversions. TC++PL 11.3.5, 11.4, D&E 3.6.1, 3.6.3.

using - see using-directive and using-declaration.

using-declaration - declaration of a local synonym for a name in another namespace or class. Example of using-declaration used to simplify overloading. See also: overloading, argument-based lookup. TC++PL 8.2.2. D&E 17.4.

using-directive - directive making a namespace accessible. See also:

argument-based lookup. TC++PL 8.2.3. D&E 17.4.

valarray - standard library numeric vector type supporting vector operations. TC++PL 22.4.

value - the bits of an object interpreted according to the objects type.

value return - The semantics of function return is to pass a copy of the return value. The copy operation is defined by the return type's copy constructor. TC++PL 7.4.

variable - named object in a scope. TC++PL 2.3.1, 10.4.3, D&E 2.3.

variable definition - declaration of a named object of a data type without an extern specifier.

vector - standard library template providing contiguous storage, re-sizing and the useful push_back() functions for adding elements at the end. Vector is the default container. See also: map, multimap, list, deque. TC++PL 3.7.1, 16.3.

virtual - keyword used to declare a member function virtual.

virtual base - a base that is shared by all classes in a class hierarchy that has declared it virtual. TC++PL 15.2.4, D&E 12.3, 12.4.1.

virtual constructor - a constructor cannot be virtual, because to create an object, we need complete information of its type. "virtual constructor" is the name of a technique for calling a virtual function to create an object of an appropriate type. Example. TC++PL 12.4.4, 15.6.2.

virtual destructor - a destructor declared virtual to ensure that the proper derived class destructor is called if an object of a derived class is deleted through a pointer to a base class. If a class has any virtual functions, it should have a virtual destructor. Example. TC++PL 12.4.2, D&E 10.5.

virtual member function - a member function that a derived class can override; the primary mechanism for run-time polymorphism in C++. A virtual member function is sometimes called a method. See also: overriding, pure virtual function. TC++PL 2.5.4, 2.5.5, 12.2.6, D&E 3.5, 12.4.

virtual-function pointer - a pointer to a class' virtual function table.

virtual-function table - table of all virtual functions for a class. The most common way of implementing virtual functions is to have each object of a class with virtual functions contain a virtual function pointer pointing to the class' virtual function table.

visitor pattern - a way of using double dispatch to simulate virtual calls without adding new virtual functions.

Visual C++ - Microsoft's implementation of C++ together with proprietary libraries for Windows programming in an IDE.

void - a keyword used to indicate an absence of information. TC++PL 4.1.1, 4.7.

void* - pointer to void; that is, a pointer to an object of unknown type; also called pointer to raw memory. A void* cannot be used or assigned without a cast. TC++PL 5.6, D&E 11.2.1, 11.2.3.

volatile - attribute of a declaration telling the compiler that an entity can have its value changed by extralinguistic means; for example, a real time clock: "extern volatile const long clock;". Limits optimizations. TC++PL A.7.1.

vptr - see virtual-function pointer.

vtbl - see virtual-function table.

wchar_t - wide character type. Used to hold characters of character sets that require more than a byte to represent, such as unicode. TC++PL 4.3, C.3.3. See also: large character sets, universal character name.

WG21 - a common abbreviation of the name of the ISO C++ standards committee.

while-statement - a loop statement presenting its condition "at the top". For example, while (cin>>var) vec.push_back(var);

whitespace - characters that a represented only by the space they take up on a page or screen. The most common examples are space (' '), newline ('\n'), and tab ('\t').

word - a number of bytes that on a given machine is particularly suied to holding an integers or a pointer. On many machines, an object must be aligned on a word boundary for acceptable performance. An int is typically a stored in a word. Often, a word is 4 bytes. See also: alignment. TC++PL 4.6.

xor - synonym for ^, the bitwise exclusive or operator TC++PL C.3.1.

zero - see NULL