

Contents

1	General.....	1
1.1	Scope.....	1
1.2	Normative references	1
1.3	Terms and definitions	1
1.3.1	argument	1
1.3.2	diagnostic message	2
1.3.3	dynamic type.....	2
1.3.4	ill-formed program.....	2
1.3.5	implementation-defined behavior	2
1.3.6	implementation limits.....	2
1.3.7	locale-specific behavior	2
1.3.8	multibyte character	2
1.3.9	parameter	2
1.3.10	signature.....	2
1.3.11	static type.....	2
1.3.12	undefined behavior	2
1.3.13	unspecified behavior.....	3
1.3.14	well-formed program	3
1.4	Implementation compliance.....	3
1.5	Structure of this International Standard.....	4
1.6	Syntax notation	4
1.7	The C++ memory model.....	4
1.8	The C++ object model	4
1.9	Program execution	5

1.10	Acknowledgments	8
2	Lexical conventions	9
2.1	Phases of translation	9
2.2	Character sets	10
2.3	Trigraph sequences	11
2.4	Preprocessing tokens	11
2.5	Alternative tokens	12
2.6	Tokens	12
2.7	Comments	12
2.8	Header names	13
2.9	Preprocessing numbers	13
2.10	Identifiers	13
2.11	Keywords	14
2.12	Operators and punctuators	15
2.13	Literals	15
2.13.1	Integer literals	15
2.13.2	Character literals	16
2.13.3	Floating literals	18
2.13.4	String literals	19
2.13.5	Boolean literals	19
3	Basic concepts	21
3.1	Declarations and definitions	21
3.2	One definition rule	22
3.3	Declarative regions and scopes	24
3.3.1	Point of declaration	25
3.3.2	Local scope	26
3.3.3	Function prototype scope	26
3.3.4	Function scope	27
3.3.5	Namespace scope	27
3.3.6	Class scope	27
3.3.7	Name hiding	28
3.4	Name lookup	29
3.4.1	Unqualified name lookup	29
3.4.2	Argument-dependent name lookup	32
3.4.3	Qualified name lookup	34

3.4.3.1	Class members	35
3.4.3.2	Namespace members	35
3.4.4	Elaborated type specifiers	39
3.4.5	Class member access	40
3.4.6	Using-directives and namespace aliases	41
3.5	Program and linkage	41
3.6	Start and termination.....	43
3.6.1	Main function.....	43
3.6.2	Initialization of non-local objects	44
3.6.3	Termination.....	45
3.7	Storage duration.....	46
3.7.1	Static storage duration	46
3.7.2	Automatic storage duration.....	46
3.7.3	Dynamic storage duration.....	47
3.7.3.1	Allocation functions.....	47
3.7.3.2	Deallocation functions	48
3.7.4	Duration of sub-objects.....	48
3.8	Object Lifetime	49
3.9	Types.....	52
3.9.1	Fundamental types	53
3.9.2	Compound types	55
3.9.3	CV-qualifiers	55
3.10	Lvalues and rvalues	56
4	Standard conversions	59
4.1	Lvalue-to-rvalue conversion	59
4.2	Array-to-pointer conversion	60
4.3	Function-to-pointer conversion	60
4.4	Qualification conversions	60
4.5	Integral promotions.....	61
4.6	Floating point promotion	61
4.7	Integral conversions.....	62
4.8	Floating point conversions.....	62
4.9	Floating-integral conversions	62
4.10	Pointer conversions.....	62
4.11	Pointer to member conversions	63

4.12	Boolean conversions	63
5	Expressions	65
5.1	Primary expressions	66
5.2	Postfix expressions	68
5.2.1	Subscripting	68
5.2.2	Function call	68
5.2.3	Explicit type conversion (functional notation)	70
5.2.4	Pseudo destructor call	70
5.2.5	Class member access	70
5.2.6	Increment and decrement	71
5.2.7	Dynamic cast	72
5.2.8	Type identification	73
5.2.9	Static cast	74
5.2.10	Reinterpret cast	75
5.2.11	Const cast	76
5.3	Unary expressions	78
5.3.1	Unary operators	78
5.3.2	Increment and decrement	79
5.3.3	Sizeof	79
5.3.4	New	80
5.3.5	Delete	83
5.4	Explicit type conversion (cast notation)	84
5.5	Pointer-to-member operators	85
5.6	Multiplicative operators	85
5.7	Additive operators	86
5.8	Shift operators	87
5.9	Relational operators	87
5.10	Equality operators	88
5.11	Bitwise AND operator	89
5.12	Bitwise exclusive OR operator	89
5.13	Bitwise inclusive OR operator	89
5.14	Logical AND operator	89
5.15	Logical OR operator	90
5.16	Conditional operator	90
5.17	Assignment operators	91

5.18	Comma operator	92
5.19	Constant expressions	92
6	Statements	95
6.1	Labeled statement	95
6.2	Expression statement	95
6.3	Compound statement or block	95
6.4	Selection statements	96
6.4.1	The if statement	97
6.4.2	The switch statement	97
6.5	Iteration statements	97
6.5.1	The while statement	98
6.5.2	The do statement	98
6.5.3	The for statement	99
6.6	Jump statements	99
6.6.1	The break statement	99
6.6.2	The continue statement	100
6.6.3	The return statement	100
6.6.4	The goto statement	100
6.7	Declaration statement	100
6.8	Ambiguity resolution	101
7	Declarations	103
7.1	Specifiers	104
7.1.1	Storage class specifiers	105
7.1.2	Function specifiers	106
7.1.3	The typedef specifier	107
7.1.4	The friend specifier	108
7.1.5	Type specifiers	108
7.1.5.1	The <i>cv-qualifiers</i>	109
7.1.5.2	Simple type specifiers	110
7.1.5.3	Elaborated type specifiers	111
7.2	Enumeration declarations	112
7.3	Namespaces	114
7.3.1	Namespace definition	114
7.3.1.1	Unnamed namespaces	115
7.3.1.2	Namespace member definitions	115
7.3.2	Namespace alias	117
7.3.3	The using declaration	117
7.3.4	Using directive	123
7.4	The asm declaration	126

7.5	Linkage specifications	126
8	Declarators	131
8.1	Type names	132
8.2	Ambiguity resolution	132
8.3	Meaning of declarators	134
8.3.1	Pointers	135
8.3.2	References.....	135
8.3.3	Pointers to members	136
8.3.4	Arrays	137
8.3.5	Functions.....	138
8.3.6	Default arguments.....	141
8.4	Function definitions	144
8.5	Initializers	145
8.5.1	Aggregates	147
8.5.2	Character arrays	150
8.5.3	References.....	150
9	Classes	153
9.1	Class names	153
9.2	Class members	155
9.3	Member functions	157
9.3.1	Nonstatic member functions	158
9.3.2	The <code>this</code> pointer	160
9.4	Static members.....	160
9.4.1	Static member functions	161
9.4.2	Static data members.....	161
9.5	Unions.....	162
9.6	Bit-fields	163
9.7	Nested class declarations	164
9.8	Local class declarations	165
9.9	Nested type names	166
10	Derived classes	167
10.1	Multiple base classes	168
10.2	Member name lookup	169
10.3	Virtual functions	172

10.4	Abstract classes.....	176
11	Member access control	179
11.1	Access specifiers.....	180
11.2	Accessibility of base classes and base class members.....	181
11.3	Access declarations.....	182
11.4	Friends	183
11.5	Protected member access	186
11.6	Access to virtual functions.....	187
11.7	Multiple access	188
11.8	Nested classes	188
12	Special member functions.....	189
12.1	Constructors	189
12.2	Temporary objects	191
12.3	Conversions	192
12.3.1	Conversion by constructor.....	193
12.3.2	Conversion functions	194
12.4	Destructors.....	195
12.5	Free store	198
12.6	Initialization.....	199
12.6.1	Explicit initialization	200
12.6.2	Initializing bases and members.....	201
12.7	Construction and destruction	204
12.8	Copying class objects	207
13	Overloading	213
13.1	Overloadable declarations.....	213
13.2	Declaration matching.....	215
13.3	Overload resolution	216
13.3.1	Candidate functions and argument lists	217
13.3.1.1	Function call syntax.....	218
13.3.1.1.1	Call to named function.....	218
13.3.1.1.2	Call to object of class type.....	219
13.3.1.2	Operators in expressions.....	220

13.3.1.3	Initialization by constructor	222
13.3.1.4	Copy-initialization of class by user-defined conversion	222
13.3.1.5	Initialization by conversion function	222
13.3.1.6	Initialization by conversion function for direct reference binding	223
13.3.2	Viable functions	223
13.3.3	Best Viable Function	223
13.3.3.1	Implicit conversion sequences	225
13.3.3.1.1	Standard conversion sequences	227
13.3.3.1.2	User-defined conversion sequences	227
13.3.3.1.3	Ellipsis conversion sequences	228
13.3.3.1.4	Reference binding	228
13.3.3.2	Ranking implicit conversion sequences	228
13.4	Address of overloaded function	230
13.5	Overloaded operators	232
13.5.1	Unary operators	233
13.5.2	Binary operators	233
13.5.3	Assignment	233
13.5.4	Function call	234
13.5.5	Subscripting	234
13.5.6	Class member access	234
13.5.7	Increment and decrement	234
13.6	Built-in operators	235
14	Templates	239
14.1	Template parameters	240
14.2	Names of template specializations	242
14.3	Template arguments	244
14.3.1	Template type arguments	245
14.3.2	Template non-type arguments	246
14.3.3	Template template arguments	248
14.4	Type equivalence	248
14.5	Template declarations	249
14.5.1	Class templates	249
14.5.1.1	Member functions of class templates	249
14.5.1.2	Member classes of class templates	250
14.5.1.3	Static data members of class templates	250
14.5.2	Member templates	251
14.5.3	Friends	252
14.5.4	Class template partial specializations	254
14.5.4.1	Matching of class template partial specializations	256
14.5.4.2	Partial ordering of class template specializations	257
14.5.4.3	Members of class template specializations	257
14.5.5	Function templates	258
14.5.5.1	Function template overloading	259
14.5.5.2	Partial ordering of function templates	260

14.6	Name resolution	261
14.6.1	Locally declared names	264
14.6.2	Dependent names	267
14.6.2.1	Dependent types	268
14.6.2.2	Type-dependent expressions	268
14.6.2.3	Value-dependent expressions	269
14.6.2.4	Dependent template arguments	269
14.6.3	Non-dependent names	270
14.6.4	Dependent name resolution	270
14.6.4.1	Point of instantiation	270
14.6.4.2	Candidate functions	271
14.6.5	Friend names declared within a class template	271
14.7	Template instantiation and specialization	272
14.7.1	Implicit instantiation	273
14.7.2	Explicit instantiation	276
14.7.3	Explicit specialization	277
14.8	Function template specializations	282
14.8.1	Explicit template argument specification	283
14.8.2	Template argument deduction	285
14.8.2.1	Deducing template arguments from a function call	287
14.8.2.2	Deducing template arguments taking the address of a function template	288
14.8.2.3	Deducing conversion function template arguments	288
14.8.2.4	Deducing template arguments from a type	288
14.8.3	Overload resolution	293
15	Exception handling	297
15.1	Throwing an exception	298
15.2	Constructors and destructors	300
15.3	Handling an exception	300
15.4	Exception specifications	302
15.5	Special functions	304
15.5.1	The <code>terminate()</code> function	304
15.5.2	The <code>unexpected()</code> function	305
15.5.3	The <code>uncaught_exception()</code> function	305
15.6	Exceptions and access	305
16	Preprocessing directives	307
16.1	Conditional inclusion	308
16.2	Source file inclusion	309
16.3	Macro replacement	310
16.3.1	Argument substitution	311
16.3.2	The <code>#</code> operator	311
16.3.3	The <code>##</code> operator	312

16.3.4	Rescanning and further replacement.....	312
16.3.5	Scope of macro definitions	312
16.4	Line control.....	314
16.5	Error directive	314
16.6	Pragma directive	314
16.7	Null directive	314
16.8	Predefined macro names.....	315
17	Library introduction.....	317
17.1	Definitions	317
17.1.1	arbitrary-positional stream.....	317
17.1.2	character.....	317
17.1.3	character container type.....	317
17.1.4	comparison function	317
17.1.5	component.....	318
17.1.6	default behavior	318
17.1.7	handler function.....	318
17.1.8	iostream class templates	318
17.1.9	modifier function	318
17.1.10	object state	318
17.1.11	narrow-oriented iostream classes.....	318
17.1.12	NTCTS.....	318
17.1.13	observer function	318
17.1.14	replacement function.....	318
17.1.15	required behavior	318
17.1.16	repositional stream.....	319
17.1.17	reserved function.....	319
17.1.18	traits class.....	319
17.1.19	wide-oriented iostream classes	319
17.2	Additional definitions	319
17.3	Method of description (Informative)	319
17.3.1	Structure of each subclause.....	319
17.3.1.1	Summary.....	320
17.3.1.2	Requirements	320
17.3.1.3	Specifications.....	320
17.3.1.4	C Library.....	321
17.3.2	Other conventions.....	321
17.3.2.1	Type descriptions.....	321
17.3.2.1.1	Enumerated types.....	322
17.3.2.1.2	Bitmask types.....	322
17.3.2.1.3	Character sequences.....	323
17.3.2.1.3.1	Byte strings	323
17.3.2.1.3.2	Multibyte strings.....	324
17.3.2.1.3.3	Wide-character sequences.....	324
17.3.2.2	Functions within classes	324
17.3.2.3	Private members	324

17.4	Library-wide requirements	324
17.4.1	Library contents and organization	325
17.4.1.1	Library contents	325
17.4.1.2	Headers	325
17.4.1.3	Freestanding implementations	326
17.4.2	Using the library	326
17.4.2.1	Headers	326
17.4.2.2	Linkage	327
17.4.3	Constraints on programs	327
17.4.3.1	Reserved names	327
17.4.3.1.1	Macro names	327
17.4.3.1.2	Global names	327
17.4.3.1.3	External linkage	328
17.4.3.1.4	Types	328
17.4.3.2	Headers	328
17.4.3.3	Derived classes	328
17.4.3.4	Replacement functions	328
17.4.3.5	Handler functions	329
17.4.3.6	Other functions	329
17.4.3.7	Function arguments	330
17.4.3.8	Required paragraph	330
17.4.4	Conforming implementations	330
17.4.4.1	Headers	330
17.4.4.2	Restrictions on macro definitions	330
17.4.4.3	Global or non-member functions	330
17.4.4.4	Member functions	331
17.4.4.5	Reentrancy	331
17.4.4.6	Protection within classes	331
17.4.4.7	Derived classes	331
17.4.4.8	Restrictions on exception handling	331
18	Language support library	333
18.1	Types	333
18.2	Implementation properties	334
18.2.1	Numeric limits	334
18.2.1.1	Class template <code>numeric_limits</code>	334
18.2.1.2	<code>numeric_limits</code> members	335
18.2.1.3	Type <code>float_round_style</code>	339
18.2.1.4	Type <code>float_denorm_style</code>	340
18.2.1.5	<code>numeric_limits</code> specializations	340
18.2.2	C Library	341
18.3	Start and termination	342
18.4	Dynamic memory management	343
18.4.1	Storage allocation and deallocation	343
18.4.1.1	Single-object forms	343
18.4.1.2	Array forms	345
18.4.1.3	Placement forms	345
18.4.2	Storage allocation errors	346
18.4.2.1	Class <code>bad_alloc</code>	346
18.4.2.2	Type <code>new_handler</code>	347

18.4.2.3	set_new_handler	347
18.5	Type identification.....	347
18.5.1	Class type_info	347
18.5.2	Class bad_cast	348
18.5.3	Class bad_typeid.....	349
18.6	Exception handling	349
18.6.1	Class exception.....	349
18.6.2	Violating <i>exception-specifications</i>	350
18.6.2.1	Class bad_exception	350
18.6.2.2	Type unexpected_handler.....	351
18.6.2.3	set_unexpected.....	351
18.6.2.4	unexpected.....	351
18.6.3	Abnormal termination.....	351
18.6.3.1	Type terminate_handler	351
18.6.3.2	set_terminate.....	352
18.6.3.3	terminate.....	352
18.6.4	uncaught_exception	352
18.7	Other runtime support.....	352
19	Diagnostics library	355
19.1	Exception classes	355
19.1.1	Class logic_error	355
19.1.2	Class domain_error.....	356
19.1.3	Class invalid_argument	356
19.1.4	Class length_error.....	356
19.1.5	Class out_of_range.....	357
19.1.6	Class runtime_error	357
19.1.7	Class range_error	357
19.1.8	Class overflow_error	357
19.1.9	Class underflow_error.....	358
19.2	Assertions	358
19.3	Error numbers	358
20	General utilities library	359
20.1	Requirements	359
20.1.1	Equality comparison	359
20.1.2	Less than comparison	359
20.1.3	Copy construction.....	360
20.1.4	Default construction.....	360
20.1.5	Allocator requirements	360
20.2	Utility components.....	363
20.2.1	Operators.....	364
20.2.2	Pairs	364
20.3	Function objects.....	365
20.3.1	Base.....	367

20.3.2	Arithmetic operations	367
20.3.3	Comparisons	368
20.3.4	Logical operations	369
20.3.5	Negators	369
20.3.6	Binders	370
20.3.6.1	Class template binder1st	370
20.3.6.2	bind1st	370
20.3.6.3	Class template binder2nd	370
20.3.6.4	bind2nd	371
20.3.7	Adaptors for pointers to functions	371
20.3.8	Adaptors for pointers to members	372
20.4	Memory	374
20.4.1	The default allocator	374
20.4.1.1	allocator members	375
20.4.1.2	allocator globals	376
20.4.2	Raw storage iterator	376
20.4.3	Temporary buffers	377
20.4.4	Specialized algorithms	377
20.4.4.1	uninitialized_copy	377
20.4.4.2	uninitialized_fill	378
20.4.4.3	uninitialized_fill_n	378
20.4.5	Class template auto_ptr	378
20.4.5.1	auto_ptr constructors	379
20.4.5.2	auto_ptr members	379
20.4.5.3	auto_ptr conversions	380
20.4.6	C Library	380
20.5	Date and time	381
21	Strings library	383
21.1	Character traits	383
21.1.1	Character traits requirements	383
21.1.2	traits typedefs	385
21.1.3	char_traits specializations	385
21.1.3.1	struct char_traits<char>	385
21.1.3.2	struct char_traits<wchar_t>	386
21.2	String classes	387
21.3	Class template basic_string	389
21.3.1	basic_string constructors	393
21.3.2	basic_string iterator support	396
21.3.3	basic_string capacity	396
21.3.4	basic_string element access	398
21.3.5	basic_string modifiers	398
21.3.5.1	basic_string::operator+=	398
21.3.5.2	basic_string::append	398
21.3.5.3	basic_string::assign	399
21.3.5.4	basic_string::insert	400
21.3.5.5	basic_string::erase	401
21.3.5.6	basic_string::replace	401
21.3.5.7	basic_string::copy	402

21.3.5.8	<code>basic_string::swap</code>	403
21.3.6	<code>basic_string</code> string operations	403
21.3.6.1	<code>basic_string::find</code>	403
21.3.6.2	<code>basic_string::rfind</code>	404
21.3.6.3	<code>basic_string::find_first_of</code>	404
21.3.6.4	<code>basic_string::find_last_of</code>	405
21.3.6.5	<code>basic_string::find_first_not_of</code>	405
21.3.6.6	<code>basic_string::find_last_not_of</code>	406
21.3.6.7	<code>basic_string::substr</code>	406
21.3.6.8	<code>basic_string::compare</code>	406
21.3.7	<code>basic_string</code> non-member functions	407
21.3.7.1	<code>operator+</code>	407
21.3.7.2	<code>operator==</code>	408
21.3.7.3	<code>operator!=</code>	408
21.3.7.4	<code>operator<</code>	409
21.3.7.5	<code>operator></code>	409
21.3.7.6	<code>operator<=</code>	409
21.3.7.7	<code>operator>=</code>	410
21.3.7.8	<code>swap</code>	410
21.3.7.9	Inserters and extractors	410
21.4	Null-terminated sequence utilities	411
22	Localization library	415
22.1	Locales	415
22.1.1	Class <code>locale</code>	416
22.1.1.1	locale types	418
22.1.1.1.1	Type <code>locale::category</code>	418
22.1.1.1.2	Class <code>locale::facet</code>	420
22.1.1.1.3	Class <code>locale::id</code>	420
22.1.1.2	locale constructors and destructor	421
22.1.1.3	locale members	422
22.1.1.4	locale operators	422
22.1.1.5	locale static members	423
22.1.2	locale globals	423
22.1.3	Convenience interfaces	423
22.1.3.1	Character classification	423
22.1.3.2	Character conversions	424
22.2	Standard locale categories	424
22.2.1	The <code>ctype</code> category	424
22.2.1.1	Class template <code>ctype</code>	424
22.2.1.1.1	<code>ctype</code> members	425
22.2.1.1.2	<code>ctype</code> virtual functions	426
22.2.1.2	Class template <code>ctype_byname</code>	427
22.2.1.3	<code>ctype</code> specializations	428
22.2.1.3.1	<code>ctype<char></code> destructor	429
22.2.1.3.2	<code>ctype<char></code> members	429
22.2.1.3.3	<code>ctype<char></code> static members	430
22.2.1.3.4	<code>ctype<char></code> virtual functions	430
22.2.1.4	Class <code>ctype_byname<char></code>	431
22.2.1.5	Class template <code>codecvt</code>	431
22.2.1.5.1	<code>codecvt</code> members	432

22.2.1.5.2	codecvt virtual functions	433
22.2.1.6	Class template codecvt_byname	435
22.2.2	The numeric category	435
22.2.2.1	Class template num_get	435
22.2.2.1.1	num_get members	437
22.2.2.1.2	num_get virtual functions	437
22.2.2.2	Class template num_put	439
22.2.2.2.1	num_put members	440
22.2.2.2.2	num_put virtual functions	440
22.2.3	The numeric punctuation facet	443
22.2.3.1	Class template numpunct	443
22.2.3.1.1	numpunct members	444
22.2.3.1.2	numpunct virtual functions	445
22.2.3.2	Class template numpunct_byname	445
22.2.4	The collate category	445
22.2.4.1	Class template collate	445
22.2.4.1.1	collate members	446
22.2.4.1.2	collate virtual functions	446
22.2.4.2	Class template collate_byname	447
22.2.5	The time category	447
22.2.5.1	Class template time_get	447
22.2.5.1.1	time_get members	448
22.2.5.1.2	time_get virtual functions	449
22.2.5.2	Class template time_get_byname	450
22.2.5.3	Class template time_put	450
22.2.5.3.1	time_put members	451
22.2.5.3.2	time_put virtual functions	451
22.2.5.4	Class template time_put_byname	451
22.2.6	The monetary category	452
22.2.6.1	Class template money_get	452
22.2.6.1.1	money_get members	452
22.2.6.1.2	money_get virtual functions	452
22.2.6.2	Class template money_put	454
22.2.6.2.1	money_put members	454
22.2.6.2.2	money_put virtual functions	454
22.2.6.3	Class template moneypunct	455
22.2.6.3.1	moneypunct members	456
22.2.6.3.2	moneypunct virtual functions	456
22.2.6.4	Class template moneypunct_byname	457
22.2.7	The message retrieval category	458
22.2.7.1	Class template messages	458
22.2.7.1.1	messages members	458
22.2.7.1.2	messages virtual functions	459
22.2.7.2	Class template messages_byname	459
22.2.8	Program-defined facets	459
22.3	C Library Locales	463
23	Containers library	465
23.1	Container requirements	465
23.1.1	Sequences	468
23.1.2	Associative containers	471

23.2	Sequences	474
23.2.1	Class template deque	477
23.2.1.1	deque constructors, copy, and assignment	479
23.2.1.2	deque capacity	480
23.2.1.3	deque modifiers	480
23.2.1.4	deque specialized algorithms	480
23.2.2	Class template list	481
23.2.2.1	list constructors, copy, and assignment	483
23.2.2.2	list capacity	484
23.2.2.3	list modifiers	484
23.2.2.4	list operations	484
23.2.2.5	list specialized algorithms	486
23.2.3	Container adaptors	486
23.2.3.1	Class template queue	486
23.2.3.2	Class template priority_queue	487
23.2.3.2.1	priority_queue constructors	488
23.2.3.2.2	priority_queue members	488
23.2.3.3	Class template stack	488
23.2.4	Class template vector	489
23.2.4.1	vector constructors, copy, and assignment	491
23.2.4.2	vector capacity	492
23.2.4.3	vector modifiers	492
23.2.4.4	vector specialized algorithms	493
23.2.5	Class vector<bool>	493
23.3	Associative containers	495
23.3.1	Class template map	497
23.3.1.1	map constructors, copy, and assignment	499
23.3.1.2	map element access	500
23.3.1.3	map operations	500
23.3.1.4	map specialized algorithms	500
23.3.2	Class template multimap	500
23.3.2.1	multimap constructors	503
23.3.2.2	multimap operations	503
23.3.2.3	multimap specialized algorithms	503
23.3.3	Class template set	503
23.3.3.1	set constructors, copy, and assignment	505
23.3.3.2	set specialized algorithms	506
23.3.4	Class template multiset	506
23.3.4.1	multiset constructors	508
23.3.4.2	multiset specialized algorithms	508
23.3.5	Class template bitset	509
23.3.5.1	bitset constructors	510
23.3.5.2	bitset members	511
23.3.5.3	bitset operators	514
24	Iterators library	515
24.1	Iterator requirements	515
24.1.1	Input iterators	516
24.1.2	Output iterators	517
24.1.3	Forward iterators	518
24.1.4	Bidirectional iterators	519
24.1.5	Random access iterators	519

24.2	Header <iterator> synopsis	520
24.3	Iterator primitives	522
24.3.1	Iterator traits.....	522
24.3.2	Basic iterator.....	523
24.3.3	Standard iterator tags	524
24.3.4	Iterator operations.....	525
24.4	Predefined iterators	525
24.4.1	Reverse iterators	525
24.4.1.1	Class template reverse_iterator	526
24.4.1.2	reverse_iterator requirements.....	527
24.4.1.3	reverse_iterator operations	527
24.4.1.3.1	reverse_iterator constructor.....	527
24.4.1.3.2	Conversion.....	527
24.4.1.3.3	operator*.....	527
24.4.1.3.4	operator->	528
24.4.1.3.5	operator++	528
24.4.1.3.6	operator--	528
24.4.1.3.7	operator+.....	528
24.4.1.3.8	operator+=	528
24.4.1.3.9	operator-.....	529
24.4.1.3.10	operator-=	529
24.4.1.3.11	operator[]	529
24.4.1.3.12	operator==	529
24.4.1.3.13	operator<.....	529
24.4.1.3.14	operator!=	529
24.4.1.3.15	operator>.....	529
24.4.1.3.16	operator>=	530
24.4.1.3.17	operator<=	530
24.4.1.3.18	operator-.....	530
24.4.1.3.19	operator+.....	530
24.4.2	Insert iterators	530
24.4.2.1	Class template back_insert_iterator.....	531
24.4.2.2	back_insert_iterator operations.....	531
24.4.2.2.1	back_insert_iterator constructor	531
24.4.2.2.2	back_insert_iterator::operator=	531
24.4.2.2.3	back_insert_iterator::operator*.....	531
24.4.2.2.4	back_insert_iterator::operator++.....	531
24.4.2.2.5	back_inserter.....	532
24.4.2.3	Class template front_insert_iterator	532
24.4.2.4	front_insert_iterator operations	532
24.4.2.4.1	front_insert_iterator constructor.....	532
24.4.2.4.2	front_insert_iterator::operator=.....	532
24.4.2.4.3	front_insert_iterator::operator*.....	532
24.4.2.4.4	front_insert_iterator::operator++	533
24.4.2.4.5	front_inserter.....	533
24.4.2.5	Class template insert_iterator.....	533
24.4.2.6	insert_iterator operations.....	533
24.4.2.6.1	insert_iterator constructor	533
24.4.2.6.2	insert_iterator::operator=	533
24.4.2.6.3	insert_iterator::operator*.....	534
24.4.2.6.4	insert_iterator::operator++.....	534
24.4.2.6.5	inserter	534

24.5	Stream iterators.....	534
24.5.1	Class template <code>istream_iterator</code>	534
24.5.1.1	<code>istream_iterator</code> constructors and destructor.....	535
24.5.1.2	<code>istream_iterator</code> operations	535
24.5.2	Class template <code>ostream_iterator</code>	536
24.5.2.1	<code>ostream_iterator</code> constructors and destructor.....	537
24.5.2.2	<code>ostream_iterator</code> operations	537
24.5.3	Class template <code>istreambuf_iterator</code>	537
24.5.3.1	Class template <code>istreambuf_iterator::proxy</code>	538
24.5.3.2	<code>istreambuf_iterator</code> constructors	539
24.5.3.3	<code>istreambuf_iterator::operator*</code>	539
24.5.3.4	<code>istreambuf_iterator::operator++</code>	539
24.5.3.5	<code>istreambuf_iterator::equal</code>	539
24.5.3.6	<code>operator==</code>	539
24.5.3.7	<code>operator!=</code>	539
24.5.4	Class template <code>ostreambuf_iterator</code>	540
24.5.4.1	<code>ostreambuf_iterator</code> constructors	540
24.5.4.2	<code>ostreambuf_iterator</code> operations	540
25	Algorithms library	543
25.1	Non-modifying sequence operations	551
25.1.1	For each	551
25.1.2	Find	552
25.1.3	Find End.....	552
25.1.4	Find First.....	552
25.1.5	Adjacent find	553
25.1.6	Count.....	553
25.1.7	Mismatch	553
25.1.8	Equal	554
25.1.9	Search	554
25.2	Mutating sequence operations	555
25.2.1	Copy.....	555
25.2.2	Swap	555
25.2.3	Transform	556
25.2.4	Replace	556
25.2.5	Fill.....	557
25.2.6	Generate	557
25.2.7	Remove	557
25.2.8	Unique.....	558
25.2.9	Reverse	558
25.2.10	Rotate.....	559
25.2.11	Random shuffle.....	559
25.2.12	Partitions	560
25.3	Sorting and related operations	560
25.3.1	Sorting.....	561
25.3.1.1	<code>sort</code>	561
25.3.1.2	<code>stable_sort</code>	561
25.3.1.3	<code>partial_sort</code>	561
25.3.1.4	<code>partial_sort_copy</code>	562
25.3.2	Nth element.....	562
25.3.3	Binary search	562

25.3.3.1	lower_bound.....	562
25.3.3.2	upper_bound.....	563
25.3.3.3	equal_range.....	563
25.3.3.4	binary_search.....	563
25.3.4	Merge.....	564
25.3.5	Set operations on sorted structures	564
25.3.5.1	includes.....	565
25.3.5.2	set_union.....	565
25.3.5.3	set_intersection.....	565
25.3.5.4	set_difference.....	566
25.3.5.5	set_symmetric_difference.....	566
25.3.6	Heap operations	566
25.3.6.1	push_heap.....	567
25.3.6.2	pop_heap.....	567
25.3.6.3	make_heap.....	567
25.3.6.4	sort_heap.....	567
25.3.7	Minimum and maximum	568
25.3.8	Lexicographical comparison.....	568
25.3.9	Permutation generators	569
25.4	C library algorithms.....	569
26	Numerics library	571
26.1	Numeric type requirements.....	571
26.2	Complex numbers.....	572
26.2.1	Header <complex> synopsis	572
26.2.2	Class template complex.....	573
26.2.3	complex specializations.....	574
26.2.4	complex member functions.....	575
26.2.5	complex member operators.....	575
26.2.6	complex non-member operations	576
26.2.7	complex value operations	578
26.2.8	complex transcendentals.....	578
26.3	Numeric arrays.....	579
26.3.1	Header <valarray> synopsis	579
26.3.2	Class template valarray.....	582
26.3.2.1	valarray constructors.....	584
26.3.2.2	valarray assignment.....	584
26.3.2.3	valarray element access	585
26.3.2.4	valarray subset operations	585
26.3.2.5	valarray unary operators	586
26.3.2.6	valarray computed assignment	586
26.3.2.7	valarray member functions	587
26.3.3	valarray non-member operations	588
26.3.3.1	valarray binary operators	588
26.3.3.2	valarray logical operators	589
26.3.3.3	valarray transcendentals.....	590
26.3.4	Class slice.....	590
26.3.4.1	slice constructors.....	591
26.3.4.2	slice access functions	591
26.3.5	Class template slice_array	591

26.3.5.1	slice_array constructors	592
26.3.5.2	slice_array assignment	592
26.3.5.3	slice_array computed assignment	592
26.3.5.4	slice_array fill function	593
26.3.6	The gslice class	593
26.3.6.1	gslice constructors	594
26.3.6.2	gslice access functions	594
26.3.7	Class template gslice_array	594
26.3.7.1	gslice_array constructors	595
26.3.7.2	gslice_array assignment	595
26.3.7.3	gslice_array computed assignment	595
26.3.7.4	gslice_array fill function	596
26.3.8	Class template mask_array	596
26.3.8.1	mask_array constructors	596
26.3.8.2	mask_array assignment	596
26.3.8.3	mask_array computed assignment	597
26.3.8.4	mask_array fill function	597
26.3.9	Class template indirect_array	597
26.3.9.1	indirect_array constructors	598
26.3.9.2	indirect_array assignment	598
26.3.9.3	indirect_array computed assignment	598
26.3.9.4	indirect_array fill function	599
26.4	Generalized numeric operations	599
26.4.1	Accumulate	599
26.4.2	Inner product	600
26.4.3	Partial sum	600
26.4.4	Adjacent difference	601
26.5	C Library	601
27	Input/output library	605
27.1	Iostreams requirements	605
27.1.1	Imbue Limitations	605
27.1.2	Positioning Type Limitations	605
27.2	Forward declarations	605
27.3	Standard istream objects	608
27.3.1	Narrow stream objects	608
27.3.2	Wide stream objects	609
27.4	Iostreams base classes	610
27.4.1	Types	610
27.4.2	Class ios_base	611
27.4.2.1	Types	613
27.4.2.1.1	Class ios_base::failure	613
27.4.2.1.2	Type ios_base::fmtflags	613
27.4.2.1.3	Type ios_base::iostate	614
27.4.2.1.4	Type ios_base::openmode	615
27.4.2.1.5	Type ios_base::seekdir	615
27.4.2.1.6	Class ios_base::Init	615
27.4.2.2	ios_base fmtflags state functions	616

27.4.2.3	ios_base locale functions	616
27.4.2.4	ios_base static members	617
27.4.2.5	ios_base storage functions	617
27.4.2.6	ios_base callbacks	618
27.4.2.7	ios_base constructors/destructors	618
27.4.3	Class template fpos	618
27.4.3.1	fpos Members	618
27.4.3.2	fpos requirements	618
27.4.4	Class template basic_ios	619
27.4.4.1	basic_ios constructors	620
27.4.4.2	Member functions	621
27.4.4.3	basic_ios iostate flags functions	622
27.4.5	ios_base manipulators	623
27.4.5.1	fmtflags manipulators	623
27.4.5.2	adjustfield manipulators	624
27.4.5.3	basefield manipulators	625
27.4.5.4	floatfield manipulators	625
27.5	Stream buffers	625
27.5.1	Stream buffer requirements	626
27.5.2	Class template basic_streambuf<charT, traits>	626
27.5.2.1	basic_streambuf constructors	628
27.5.2.2	basic_streambuf public member functions	629
27.5.2.2.1	Locales	629
27.5.2.2.2	Buffer management and positioning	629
27.5.2.2.3	Get area	629
27.5.2.2.4	Putback	630
27.5.2.2.5	Put area	630
27.5.2.3	basic_streambuf protected member functions	630
27.5.2.3.1	Get area access	630
27.5.2.3.2	Put area access	631
27.5.2.4	basic_streambuf virtual functions	631
27.5.2.4.1	Locales	631
27.5.2.4.2	Buffer management and positioning	631
27.5.2.4.3	Get area	632
27.5.2.4.4	Putback	633
27.5.2.4.5	Put area	634
27.6	Formatting and manipulators	635
27.6.1	Input streams	636
27.6.1.1	Class template basic_istream	636
27.6.1.1.1	basic_istream constructors	638
27.6.1.1.2	Class basic_istream::sentry	638
27.6.1.2	Formatted input functions	639
27.6.1.2.1	Common requirements	639
27.6.1.2.2	Arithmetic Extractors	639
27.6.1.2.3	basic_istream::operator>>	640
27.6.1.3	Unformatted input functions	641
27.6.1.4	Standard basic_istream manipulators	645
27.6.1.5	Class template basic_iostream	646
27.6.1.5.1	basic_iostream constructors	646
27.6.1.5.2	basic_iostream destructor	646
27.6.2	Output streams	646
27.6.2.1	Class template basic_ostream	646

27.6.2.2	basic_ostream constructors.....	648
27.6.2.3	Class basic_ostream::sentry.....	648
27.6.2.4	basic_ostream seek members	649
27.6.2.5	Formatted output functions.....	650
27.6.2.5.1	Common requirements.....	650
27.6.2.5.2	Arithmetic Inserters	650
27.6.2.5.3	basic_ostream::operator<<	650
27.6.2.5.4	Character inserter function templates	651
27.6.2.6	Unformatted output functions.....	652
27.6.2.7	Standard basic_ostream manipulators	653
27.6.3	Standard manipulators	653
27.7	String-based streams.....	655
27.7.1	Class template basic_stringbuf.....	656
27.7.1.1	basic_stringbuf constructors.....	657
27.7.1.2	Member functions.....	657
27.7.1.3	Overridden virtual functions.....	658
27.7.2	Class template basic_istream.....	660
27.7.2.1	basic_istream constructors	660
27.7.2.2	Member functions.....	661
27.7.3	Class basic_ostringstream.....	661
27.7.3.1	basic_ostringstream constructors	662
27.7.3.2	Member functions.....	662
27.7.4	Class template basic_stringstream.....	662
27.7.5	basic_stringstream constructors.....	663
27.7.6	Member functions.....	663
27.8	File-based streams	664
27.8.1	File streams.....	664
27.8.1.1	Class template basic_filebuf.....	664
27.8.1.2	basic_filebuf constructors.....	665
27.8.1.3	Member functions.....	666
27.8.1.4	Overridden virtual functions.....	667
27.8.1.5	Class template basic_ifstream.....	669
27.8.1.6	basic_ifstream constructors	670
27.8.1.7	Member functions.....	670
27.8.1.8	Class template basic_ofstream.....	671
27.8.1.9	basic_ofstream constructors	671
27.8.1.10	Member functions.....	672
27.8.1.11	Class template basic_fstream.....	672
27.8.1.12	basic_fstream constructors.....	673
27.8.1.13	Member functions.....	673
27.8.2	C Library files.....	673
Annex A (informative)	Grammar summary	675
A.1	Keywords.....	675
A.2	Lexical conventions	675
A.3	Basic concepts.....	679
A.4	Expressions	679

A.5	Statements	682
A.6	Declarations	683
A.7	Declarators	685
A.8	Classes	687
A.9	Derived classes.....	688
A.10	Special member functions.....	688
A.11	Overloading	688
A.12	Templates	689
A.13	Exception handling	689
A.14	Preprocessing directives.....	690
Annex B (informative)	Implementation quantities	693
Annex C (informative)	Compatibility	695
C.1	C++ and ISO C.....	695
C.1.1	Clause 2: lexical conventions.....	695
C.1.2	Clause 3: basic concepts	696
C.1.3	Clause 5: expressions	698
C.1.4	Clause 6: statements.....	699
C.1.5	Clause 7: declarations	699
C.1.6	Clause 8: declarators	701
C.1.7	Clause 9: classes.....	702
C.1.8	Clause 12: special member functions.....	703
C.1.9	Clause 16: preprocessing directives.....	704
C.2	Standard C library	704
C.2.1	Modifications to headers.....	706
C.2.2	Modifications to definitions.....	706
C.2.2.1	Type <code>wchar_t</code>	706
C.2.2.2	Header <code><iso646.h></code>	707
C.2.2.3	Macro <code>NULL</code>	707
C.2.3	Modifications to declarations.....	707
C.2.4	Modifications to behavior	707
C.2.4.1	Macro <code>offsetof(type, member-designator)</code>	707
C.2.4.2	Memory allocation functions	707
Annex D (normative)	Compatibility features	709
D.1	Increment operator with <code>bool</code> operand	709
D.2	<code>static</code> keyword	709
D.3	Access declarations	709

D.4	Implicit conversion from const strings	709
D.5	Standard C library headers	709
D.6	Old iostreams members	709
D.7	char* streams	711
D.7.1	Class <code>strstreambuf</code>	711
D.7.1.1	<code>strstreambuf</code> constructors	713
D.7.1.2	Member functions	714
D.7.1.3	<code>strstreambuf</code> overridden virtual functions	714
D.7.2	Class <code>istream</code>	717
D.7.2.1	<code>istream</code> constructors	717
D.7.2.2	Member functions	717
D.7.3	Class <code>ostream</code>	718
D.7.3.1	<code>ostream</code> constructors	718
D.7.3.2	Member functions	718
D.7.4	Class <code>stringstream</code>	719
D.7.4.1	<code>stringstream</code> constructors	719
D.7.4.2	<code>stringstream</code> destructor	720
D.7.4.3	<code>stringstream</code> operations	720
Annex E (normative)	Universal-character-names	721
Index	723

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 14882 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

This second edition cancels and replaces the first edition (ISO/IEC 14882:1998), which has been technically revised.

