

C++ Keywords

The following lists the keywords and reserved words of the C++ language:

asm	To declare that a block of code is to be passed to the assembler
auto	A storage class specifier that is used to define objects in a block
bool	Boolean false-true type that can hold either the false or true literals
break	Terminates a switch statement or a loop
case	Used specifically within a switch statement to specify a match for the statement's expression
catch	Specifies actions taken when an exception occurs
char	Fundamental data type that defines character objects
class	To declare a user-defined type that encapsulates data members and operations or member functions
const	To define objects whose value will not alter throughout the lifetime of program execution
continue	Transfers control to the start of a loop
default	Handles expression values in a switch statement that are not handled by case
delete	Memory deallocation operator
do	Indicates the start of a do-while statement in which the sub-statement is executed repeatedly until the value of the expression is logical-false
double	Fundamental data type used to define a floating-point number
else	Used specifically in if-else statement
enum	To declare a user-defined enumeration data type
explicit	To declare an explicit constructor
export	Allows a template definition to be <i>accessible from another translation unit</i> .
extern	An identifier specified as extern has external linkage to the block
false	Boolean literal of value zero
float	Fundamental data type used to define a floating-point number
for	Indicates the start of a for statement to achieve repetitive control
friend	A class or operation whose implementation can access the private data members of a class
goto	Transfer control to a specified label
if	Indicate start of an if statement to achieve selective control
inline	A function specifier that indicates to the compiler that inline substitution of the function body is to be preferred to the usual function call implementation

int	Fundamental data type used to define integer objects
long	A data type modifier that defines a 32-bit int or an extended double
mutable	Allows an object member to override constness
namespace	Defines a scope
new	Memory allocation operator
operator	Overloads a C++ operator with a new declaration
private	Declares class members which are not visible outside the class
protected	Declares class members which are private except to derived classes
public	Declares class members which are visible outside the class
register	A storage class specifier that is an auto specifier, but which also indicates to the compiler that an object will be frequently used and should therefore be kept in a register
return	Returns an object to a function's caller
short	A data type modifier that defines a 16-bit int number
signed	A data type modifier that indicates an object's sign is to be stored in the high-order bit
sizeof	Returns the size of an object in bytes
static	The lifetime of an object defined static exists throughout the lifetime of program execution
struct	To declare new types that encapsulate both data and member functions
switch	Switch statement
template	Parametrised or generic type
this	A class pointer which points to an object or instance of the class
throw	Generate an exception
true	Boolean literal of value one
try	Indicates start of a block of exception handlers
typedef	Synonym for another integral or user-defined type
typeid	The typeid () operator returns the type of its operand
typename	Within a template typename indicates that a qualified name denotes a type
union	Similar to a structure, struct , in that it can hold different types of data, but a union can hold only one of its members at a given time.
unsigned	A data type modifier that indicates the high-order bit is to be used for an object
using	using declaration and using directive
virtual	A function specifier that declares a member function of a class which will be redefined by a derived class
void	Absent of a type or function parameter list
volatile	Define an object which may vary in value in a way that is undetectable to the compiler
wchar_t	Wide character type
while	Start of a while statement and end of a do-while statement

ASCII Character Set

It appears customary to include the ASCII character set in an Appendix of a book on computer programming, so here it is¹:

<i>Decimal</i>	<i>Hexadecimal</i>	<i>Octal</i>	<i>Binary</i>	<i>Key</i>	<i>ASCII character</i>
0	00	0	00000000	CTRL+2	null
1	01	1	00000001	CTRL+A	☺
2	02	2	00000010	CTRL+B	●
3	03	3	00000011	CTRL+C	♡
4	04	4	00000100	CTRL+D	◇
5	05	5	00000101	CTRL+E	☼
6	06	6	00000110	CTRL+F	⚡
7	07	7	00000111	beep	•
8	08	10	00001000	backspace	■
9	09	11	00001001	tab	
10	0a	12	00001010	newline	■
11	0b	13	00001011	CTRL+K	♂
12	0c	14	00001100	CTRL+L	♀
13	0d	15	00001101	enter	♪
14	0e	16	00001110	CTRL+N	♪
15	0f	17	00001111	CTRL+O	⌘
16	10	20	00010000	CTRL+P	.
17	11	21	00010001	CTRL+Q	,
18	12	22	00010010	CTRL+R	↑
19	13	23	00010011	CTRL+S	!!
20	14	24	00010100	CTRL+T	¶
21	15	25	00010101	CTRL+U	§
22	16	26	00010110	CTRL+V	■
23	17	27	00010111	CTRL+W	±
24	18	30	00011000	CTRL+X	↑
25	19	31	00011001	CTRL+Y	↓
26	1a	32	00011010	CTRL+Z	→
27	1b	33	00011011	esc	←
28	1c	34	00011100	CTRL+\	@
29	1d	35	00011101	CTRL+]	↔

¹ Note that characters 0–31 (inclusive) and 127 are control characters, 32–126 represent keys on the keyboard and 127–255 are the IBM extended ASCII character set, which can be displayed by pressing the ALT key and typing the decimal code of the character on the numeric keypad. The IBM extended character set is not part of the ASCII character set and as a result may not be fully portable.

<i>Decimal</i>	<i>Hexadecimal</i>	<i>Octal</i>	<i>Binary</i>	<i>Key</i>	<i>ASCII character</i>
30	1e	36	00011110	CTRL+6	▲
31	1f	37	00011111	CTRL+-	▼
32	20	40	00100000	spacebar	
33	21	41	00100001	!	!
34	22	42	00100010	"	"
35	23	43	00100011	#	#
36	24	44	00100100	\$	\$
37	25	45	00100101	%	%
38	26	46	00100110	&	&
39	27	47	00100111	'	'
40	28	50	00101000	((
41	29	51	00101001))
42	2a	52	00101010	*	*
43	2b	53	00101011	+	+
44	2c	54	00101100	,	,
45	2d	55	00101101	-	-
46	2e	56	00101110	.	.
47	2f	57	00101111	/	/
48	30	60	00110000	0	0
49	31	61	00110001	1	1
50	32	62	00110010	2	2
51	33	63	00110011	3	3
52	34	64	00110100	4	4
53	35	65	00110101	5	5
54	36	66	00110110	6	6
55	37	67	00110111	7	7
56	38	70	00111000	8	8
57	39	71	00111001	9	9
58	3a	72	00111010	:	:
59	3b	73	00111011	;	;
60	3c	74	00111100	<	<
61	3d	75	00111101	=	=
62	3e	76	00111110	>	>
63	3f	77	00111111	?	?
64	40	100	01000000	@	@
65	41	101	01000001	A	A
66	42	102	01000010	B	B
67	43	103	01000011	C	C
68	44	104	01000100	D	D
69	45	105	01000101	E	E
70	46	106	01000110	F	F
71	47	107	01000111	G	G
72	48	110	01001000	H	H
73	49	111	01001001	I	I
74	4a	112	01001010	J	J
75	4b	113	01001011	K	K
76	4c	114	01001100	L	L
77	4d	115	01001101	M	M
78	4e	116	01001110	N	N
79	4f	117	01001111	O	O
80	50	120	01010000	P	P
81	51	121	01010001	Q	Q
82	52	122	01010010	R	R
83	53	123	01010011	S	S
84	54	124	01010100	T	T
85	55	125	01010101	U	U
86	56	126	01010110	V	V

<i>Decimal</i>	<i>Hexadecimal</i>	<i>Octal</i>	<i>Binary</i>	<i>Key</i>	<i>ASCII character</i>
87	57	127	01010111	W	W
88	58	130	01011000	X	X
89	59	131	01011001	Y	Y
90	5a	132	01011010	Z	Z
91	5b	133	01011011	[[
92	5c	134	01011100	\	\
93	5d	135	01011101]]
94	5e	136	01011110	^	^
95	5f	137	01011111	~	~
96	60	140	01100000	,	,
97	61	141	01100001	a	a
98	62	142	01100010	b	b
99	63	143	01100011	c	c
100	64	144	01100100	d	d
101	65	145	01100101	e	e
102	66	146	01100110	f	f
103	67	147	01100111	g	g
104	68	150	01101000	h	h
105	69	151	01101001	i	i
106	6a	152	01101010	j	j
107	6b	153	01101011	k	k
108	6c	154	01101100	l	l
109	6d	155	01101101	m	m
110	6e	156	01101110	n	n
111	6f	157	01101111	o	o
112	70	160	01110000	p	p
113	71	161	01110001	q	q
114	72	162	01110010	r	r
115	73	163	01110011	s	s
116	74	164	01110100	t	t
117	75	165	01110101	u	u
118	76	166	01110110	v	v
119	77	167	01110111	w	w
120	78	170	01111000	x	x
121	79	171	01111001	y	y
122	7a	172	01111010	z	z
123	7b	173	01111011	{	{
124	7c	174	01111100		
125	7d	175	01111101	}	}
126	7e	176	01111110	~	~
127	7f	177	01111111	CTRL+↵	Δ
128	80	200	10000000	ALT+128	Ç
129	81	201	10000001	ALT+129	ü
130	82	202	10000010	ALT+130	é
131	83	203	10000011	ALT+131	â
132	84	204	10000100	ALT+132	ä
133	85	205	10000101	ALT+133	à
134	86	206	10000110	ALT+134	ã
135	87	207	10000111	ALT+135	ç
136	88	210	10001000	ALT+136	ê
137	89	211	10001001	ALT+137	ë
138	8a	212	10001010	ALT+138	è
139	8b	213	10001011	ALT+139	ï
140	8c	214	10001100	ALT+140	î
141	8d	215	10001101	ALT+141	ì
142	8e	216	10001110	ALT+142	Ä
143	8f	217	10001111	ALT+143	Å

<i>Decimal</i>	<i>Hexadecimal</i>	<i>Octal</i>	<i>Binary</i>	<i>Key</i>	<i>ASCII character</i>
144	90	220	10010000	ALT+144	É
145	91	221	10010001	ALT+145	æ
146	92	222	10010010	ALT+146	Æ
147	93	223	10010011	ALT+147	ô
148	94	224	10010100	ALT+148	ö
149	95	225	10010101	ALT+149	ò
150	96	226	10010110	ALT+150	û
151	97	227	10010111	ALT+151	ù
152	98	230	10011000	ALT+152	
153	99	231	10011001	ALT+153	Ö
154	9a	232	10011010	ALT+154	Û
155	9b	233	10011011	ALT+155	ç
156	9c	234	10011100	ALT+156	£
157	9d	235	10011101	ALT+157	¥
158	9e	236	10011110	ALT+158	ℳ
159	9f	237	10011111	ALT+159	ƒ
160	a0	240	10100000	ALT+160	á
161	a1	241	10100001	ALT+161	í
162	a2	242	10100010	ALT+162	ó
163	a3	243	10100011	ALT+163	ú
164	a4	244	10100100	ALT+164	ñ
165	a5	245	10100101	ALT+165	Ñ
166	a6	246	10100110	ALT+166	ä
167	a7	247	10100111	ALT+167	o
168	a8	250	10101000	ALT+168	ì
169	a9	251	10101001	ALT+169	┌
170	aa	252	10101010	ALT+170	┐
171	ab	253	10101011	ALT+171	½
172	ac	254	10101100	ALT+172	¼
173	ad	255	10101101	ALT+173	ì
174	ae	256	10101110	ALT+174	«
175	af	257	10101111	ALT+175	»
176	b0	260	10110000	ALT+176	▒
177	b1	261	10110001	ALT+177	▒
178	b2	262	10110010	ALT+178	▒
179	b3	263	10110011	ALT+179	
180	b4	264	10110100	ALT+180	└
181	b5	265	10110101	ALT+181	┘
182	b6	266	10110110	ALT+182	┐
183	b7	267	10110111	ALT+183	┘
184	b8	270	10111000	ALT+184	┘
185	b9	271	10111001	ALT+185	┘
186	ba	272	10111010	ALT+186	
187	bb	273	10111011	ALT+187	┘
188	bc	274	10111100	ALT+188	┘
189	bd	275	10111101	ALT+189	┘
190	be	276	10111110	ALT+190	┘
191	bf	277	10111111	ALT+191	┘
192	c0	300	11000000	ALT+192	└
193	c1	301	11000001	ALT+193	└
194	c2	302	11000010	ALT+194	└
195	c3	303	11000011	ALT+195	└
196	c4	304	11000100	ALT+196	└
197	c5	305	11000101	ALT+197	└
198	c6	306	11000110	ALT+198	└
199	c7	307	11000111	ALT+199	└
200	c8	310	11001000	ALT+200	└

Decimal	Hexadecimal	Octal	Binary	Key	ASCII character
201	c9	311	11001001	ALT+201	ƒ
202	ca	312	11001010	ALT+202	⌌
203	cb	313	11001011	ALT+203	ƒ
204	cc	314	11001100	ALT+204	ƒ
205	cd	315	11001101	ALT+205	=
206	ce	316	11001110	ALT+206	ƒ
207	cf	317	11001111	ALT+207	⌌
208	d0	320	11010000	ALT+208	⌌
209	d1	321	11010001	ALT+209	ƒ
210	d2	322	11010010	ALT+210	ƒ
211	d3	323	11010011	ALT+211	⌌
212	d4	324	11010100	ALT+212	⌌
213	d5	325	11010101	ALT+213	ƒ
214	d6	326	11010110	ALT+214	ƒ
215	d7	327	11010111	ALT+215	ƒ
216	d8	330	11011000	ALT+216	ƒ
217	d9	331	11011001	ALT+217	⌌
218	da	332	11011010	ALT+218	ƒ
219	db	333	11011011	ALT+219	■
220	dc	334	11011100	ALT+220	■
221	dd	335	11011101	ALT+221	■
222	de	336	11011110	ALT+222	■
223	df	337	11011111	ALT+223	■
224	e0	340	11100000	ALT+224	α
225	e1	341	11100001	ALT+225	β
226	e2	342	11100010	ALT+226	Γ
227	e3	343	11100011	ALT+227	π
228	e4	344	11100100	ALT+228	Σ
229	e5	345	11100101	ALT+229	σ
230	e6	346	11100110	ALT+230	μ
231	e7	347	11100111	ALT+231	τ
232	e8	350	11101000	ALT+232	Φ
233	e9	351	11101001	ALT+233	Θ
234	e9	351	11101010	ALT+234	Ω
235	eb	353	11101011	ALT+235	δ
236	ec	354	11101100	ALT+236	∞
237	ed	355	11101101	ALT+237	ø
238	ee	356	11101110	ALT+238	ε
239	ef	357	11101111	ALT+239	∩
240	f0	360	11110000	ALT+240	≡
241	f1	361	11110001	ALT+241	±
242	f2	362	11110010	ALT+242	≥
243	f3	363	11110011	ALT+243	≤
244	f4	364	11110100	ALT+244	[
245	f5	365	11110101	ALT+245]
246	f6	366	11110110	ALT+246	÷
247	f7	367	11110111	ALT+247	≈
248	f8	370	11111000	ALT+248	°
249	f9	371	11111001	ALT+249	•
250	fa	372	11111010	ALT+250	·
251	fb	373	11111011	ALT+251	√
252	fc	374	11111100	ALT+252	n
253	fd	375	11111101	ALT+253	²
254	fe	376	11111110	ALT+254	■
255	ff	377	11111111	ALT+255	blank

Operators: Precedence, Associativity and Arity

The following list shows all of the operators in C++. The precedence, associativity and arity of each operator is given. Note that **sizeof**, **new** and **delete** are operators. All of the operators listed can be overloaded except for `::`, `..`, **sizeof**, `.*` and `?:`.

Key: L=Left, R=Right, U=Unary, B=Binary, T=Ternary, N/A=Not applicable

Operator	Precedence	Associativity	Arity	Description
<code>::</code>	17	R	U	Global scope
<code>::</code>		L	B	Class scope
<code>()</code>	16	L	N/A	Function call
<code>()</code>		L	N/A	Type construction
<code>[]</code>		L	B	Array index
<code>-></code>		L	B	Indirect member access
<code>.</code>		L	B	Direct member access
sizeof	15	R	U	Object size in bytes
new		R	U	Memory allocator
delete		R	U	Memory deallocator
<code>()</code>		R	B	cast
<code>-</code>		R	U	Minus sign
<code>+</code>		R	U	Plus sign
<code>*</code>		R	U	Pointer dereference
<code>&</code>		R	U	reference
<code>!</code>		R	U	Logical NOT
<code>~</code>		R	U	Bitwise NOT
<code>++</code>		R	U	increment
<code>--</code>		R	U	decrement
<code>.*</code>	14	L	B	Direct member pointer access
<code>->*</code>		L	B	Indirect member pointer access
<code>/</code>	13	L	B	division
<code>*</code>		L	B	multiplication
<code>%</code>		L	B	modulus
<code>+</code>	12	L	B	addition
<code>-</code>		L	B	subtraction
<code><<</code>	11	L	B	Bit-shift left

<i>Operator</i>	<i>Precedence</i>	<i>Associativity</i>	<i>Arity</i>	<i>Description</i>
>>	10	L	B	Bit-shift right
<		L	B	Less than
>		L	B	Greater than
<=		L	B	Less than or equal to
>=		L	B	Greater than or equal to
==	9	L	B	equality
!=		L	B	inequality
&	8	L	B	Bitwise AND
^	7	L	B	Bitwise XOR
	6	L	B	Bitwise OR
&&	5	L	B	Logical AND
	4	L	B	Logical OR
?:	3	L	T	conditional
=	2	R	B	assignment
+=		R	B	Addition and assignment
-=		R	B	Subtraction and assignment
/=		R	B	Division and assignment
*=		R	B	Multiplication and assignment
%=		R	B	Modulus and assignment
&=		R	B	Bitwise AND and assignment
=		R	B	Bitwise OR and assignment
^=		R	B	Bitwise XOR and assignment
<<=		R	B	Bit-shift left and assignment
>>=		R	B	Bit-shift right and assignment
,		L	B	Comma (sequencing)

Glossary

<i>abstraction</i>	The essential characteristics of an object which distinguish it from other objects.
<i>array</i>	Group of identical data types or elements.
<i>base class</i>	A class from which another class can inherit. Also known as a superclass.
<i>character</i>	Such as /, *, ?, a, A, b, Refer to Appendix B for a complete listing of the ASCII character set.
class	A class encapsulates data and functions which operate on the class 's data, defining a given structure and behaviour. A class is a type and a scope with an exact specification specified in the class declaration.
<i>comment</i>	A user-defined string of information placed within program code that does not influence compilation.
<i>compile</i>	Convert a program source code file in to an object file which contains machine language instructions.
<i>concrete class</i>	A class that possesses the functionality of C++'s built-in data types, such as int and double .
<i>concurrency</i>	The action of different objects operating simultaneously.
<i>constant</i>	Something that does not change.
<i>declaration</i>	Specifies an identifier's name and type.
<i>definition</i>	A declaration that allocates memory.
<i>derived</i>	A class that has inherited from one or more alternative classes. Also known as a subclass.
<i>encapsulation</i>	The formulation of separating the structure and behaviour of an abstraction into cells.
enum	An enumeration data type which provides mnemonic identifiers to a set of integer constants.

<i>exception</i>	A situation in which a program encounters an abnormal situation for which it was not designed. In C++ you can transfer control (throw an exception) to another part of a program that is designed to deal explicitly with exceptions.
<i>execution</i>	The process of running a program.
<i>friend</i>	A class or function that has access to the private sections of another class .
<i>function</i>	A standalone module that can have input or output or both.
<i>identifier</i>	A name given to either a constant or a variable object.
<i>inheritance</i>	A hierarchy among classes.
<i>instance</i>	An instance does things, and we can do things to an instance by sending messages. Messages are sent via member functions. An instance is also referred to as an <i>object</i> .
<i>interface</i>	A set of functions within a module.
<i>keyword</i>	Such as int , double or new . A word that the compiler already has a definition for. A keyword (or reserved word) cannot be used as the name of a data object. Keywords in C++ are in lowercase. For a list of keywords refer to Appendix A.
<i>link</i>	<i>Linking</i> is the process of combining object files into a single executable program.
<i>make</i>	Combined process of compiling and linking to generate an executable program.
<i>member function</i>	An operation on an object. Member functions are called <i>methods</i> or <i>operations</i> in other object-oriented programming languages.
<i>module</i>	A piece of program code which forms part of a larger software system. A module has a clearly defined interface. A source file is an example of a module.
<i>object</i>	An object does things, and we can do things to an object by sending messages. Messages are sent via member functions. An object is also referred to as an <i>instance</i> .
<i>persistence</i>	An object is referred to as persistent if it outlives its creator or program execution and/or an object survives a move from its original memory address.
<i>polymorphism</i>	A name that is attached to a variable which denotes objects of several different classes that are related through a common base class . Polymorphous – assuming many forms.
<i>pragmatics</i>	Rules for the good and proper use of a language.
<i>preprocessor</i>	The compiler's preprocessor. Preprocessor directives are prefixed with the number (or hash) sign, #.
<i>primitives</i>	Primary subsystems of a complex system such as base classes.

private	If a member of class is private it can only be accessed by member functions and friends of the class in which it is declared.
protected	If a member of a class is protected it can only be accessed by member functions and friends of the class in which it is declared or a class that is <i>derived</i> from this class .
public	If a member of a class is public it can be accessed by any function, member or not.
<i>run-time type information</i>	A mechanism which allows information to be obtained about an object at run-time.
<i>scope</i>	The visibility, availability and life of an object within a program. There are generally six different types of scope: global, local, class , function, namespace and file.
<i>semantics</i>	The meaning of syntactically correct constructs of a language.
<i>source file</i>	A file containing declarations, functions and objects. Frequently referred to as a translation unit.
<i>STL</i>	The C++ Standard Template Library.
structure	A user-defined declaration that encapsulates data members and member functions or operations.
<i>syntax</i>	The set of rules for the correct use of a language.
template	A generic class . Often referred to as a <i>parametrised type</i> .
<i>translation unit</i>	A module.
<i>type</i>	A class implements a type. A type determines what values and operations can be performed on an object. All names or objects must have a type in C++.
<i>type checking</i>	All objects are created with a specific type, such as int , double and Complex* . Type checking involves checking that the use of an object is consistent with its type. C++ is often referred to as <i>strongly typed</i> . If violations of type are detected at compilation, a language is referred to as <i>strongly statically typed</i> . Alternatively, if violations of type are allowed to pass compilation and type checking is postponed until run-time then a language is referred to as <i>dynamically typed</i> .
union	Similar to a <i>structure</i> , with the restriction that a union can hold only one of its members at any given time.
<i>variable</i>	As the name suggests, something that changes. The name of a storage location that depends on the type and size of the data that it contains.
<i>virtual function</i>	A base class member function that can be redefined by each derived class .

References

- Ada Reference Manual (1983) *Reference Manual for the Ada Programming Language*, February 1983, Washington DC: Department of Defence, Ada Joint Program Office.
- Adams, J., Leestma, S. and Nyhoff, L. (1995) *C++: An Introduction to Computing*, Prentice Hall, Englewood Cliffs NJ.
- Ammerau, L. (1997) *STL for C++ Programmers*, John Wiley & Sons, Chichester, England.
- Anton, H. (1991) *Elementary Linear Algebra*, 6th edn, John Wiley & Sons, New York.
- Anuff, E. (1996) *The Java Sourcebook*, John Wiley & Sons, New York.
- Austen, M. H. (1999) *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*, Addison-Wesley, Reading, MA.
- Barton, J. J. and Nackman, L. R. (1994) *Scientific and Engineering C++: An Introduction with Advanced Techniques and Examples*, Addison-Wesley, Reading MA.
- Bashein, G. and Detmer, P. R. (1994) Centroid of a polygon, in *Graphics Gems IV* (ed. Heckbert, P. S.), Academic Press, London.
- Bauby, J.-D. (1998) *The Diving-Bell and the Butterfly*, Fourth Estate, London.
- Bobrow, D., DeMichiel, L., Gabriel, R., Keene, S., Kiczales, G. and Moon, D. (1988) *Common Lisp Object System Specification*, X3J13 Document 88-002R, September 1988, *SIGPLAN Notices* 23.
- Booch, G. (1994) *Object-Oriented Analysis and Design with Applications*, 2nd edn, Benjamin/Cummings, Redwood City CA.
- Booch, G., Jacobson, I. and Rumbaugh, J. (1999) *The Unified Modelling Language User Guide*, Addison-Wesley, Reading MA.
- Bowyer, A. (1995) *SVLIS Set-Theoretic Kernel Modeller: Introduction and User Manual*, 2nd edn, Information Geometers, Winchester. See also The SVLIS Web page: http://www.bath.ac.uk/~ensab/G_mod/Svlis/svlis.html.
- Bowyer, A. and Woodwark, J. (1983) *A Programmer's Geometry*, Butterworths, London.
- Bowyer, A. and Woodwark, J. (1993) *Introduction to Computing with Geometry*, Information Geometers, Winchester.
- Budd, T. A. (1987) *A Little Smalltalk*, Addison-Wesley, Reading MA.
- Budd, T. A. (1994) *Classic Data Structures in C++*, Addison-Wesley, Reading MA.
- Calvert, C. (1997) *C++Builder*, Sams Publishing, Indianapolis.
- Coad, P. and Nicola, J. (1993) *Object-Oriented Programming*, Yourdon Press, New Jersey.
- Coplien, J. O. (1992) *Advanced C++ Programming Styles and Idioms*, Addison-Wesley, Reading MA.
- Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990) *Introduction to Algorithms*, MIT Press and McGraw-Hill, New York.

- Corney, J. (1997) *3D Modelling with the ACIS Kernel and Toolkit*, John Wiley & Sons, Chichester.
- Dahl, O.-J., Myrhaug, B. and Nygaard, K. (1970) *SIMULA Common Base Language*, Norwegian Computing Centre S-22, Oslo.
- Devlin, K. (2000) *The Maths Gene: Why Everyone has it, but most People don't use it*, Weidenfeld & Nicolson, London.
- Ellis, M. A. and Stroustrup, B. (1990) *The Annotated C++ Reference Manual*, Addison-Wesley, Reading MA.
- Ford, W. and Topp, W. (1996) *Data Structures in C++*, Prentice Hall, Englewood Cliffs NJ.
- Gifford, B. (1990) *Wild at Heart*, Grafton, London.
- Glaeser, G. (1994) *Fast Algorithms for 3D-Graphics*, Springer-Verlag, London.
- Goldberg, A. and Robson, D. (1983) *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading MA.
- Golub, G. H. and Van Loan, C. F. (1989) *Matrix Computations*, 2nd edn, Johns Hopkins University Press, Baltimore MD.
- Graham, R. L. (1972) *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set*, *Info. Proc. Lett.*, 1, 132–3.
- Hearn, D. and Baker, M. P. (1994) *Computer Graphics*, 2nd edn, Prentice Hall, Englewood Cliffs NJ.
- Horowitz, E., Sahni, S. and Mehta, D. (1995) *Fundamentals of Data Structures in C++*, Computer Science Press, New York.
- Jacobson, I., Booch, G. and Rumbaugh, J. (1999) *The Unified Software Development Process*, Addison-Wesley, Reading MA.
- Jennings, A. and McKeown, J. J. (1992) *Matrix Computation*, 2nd edn, John Wiley & Sons, New York.
- KAI (2000) *The KAI Web page*, <http://www.kai.com/>.
- Kay, A. C. (1993) *The Early History of Smalltalk*, ACM SIGPLAN Second History of Programming Languages Conference, 20–23 April 1993, ACM, pp. 69–95.
- Kernighan B. and Ritchie D. (1978) *The C Programming Language*, 1st edn, Prentice Hall, Englewood Cliffs NJ.
- Kernighan B. and Ritchie D. (1988) *The C Programming Language*, 2nd edn, Prentice Hall, Englewood Cliffs NJ.
- Kirkerud, B. (1989) *Object-Oriented Programming with Simula*, Addison-Wesley, Reading MA.
- Lafore, R. (1991) *Object-Oriented Programming in Turbo C++*, Waite Group Press, Mill Valley CA.
- Lafore, R. (1993) *Lafore's Windows Programming Made Easy*, Waite Group Press, Mill Valley CA.
- Laszlo, M. J. (1996) *Computational Geometry and Computer Graphics in C++*, Prentice Hall, Englewood Cliffs NJ.
- Lippman, S. B. (1991) *C++ Primer*, 2nd edn, Addison-Wesley, Reading MA.
- Luse, M. (1993) *Bitmapped Graphics Programming in C++*, Addison-Wesley, Reading MA.
- Masters, T. (1993) *Practical Neural Network Recipes in C++*, Academic Press, London.
- Meyer, B. (1991) *Eiffel: The Language*, Prentice Hall, Englewood Cliffs NJ.
- Meyer, B. (1995) *Object Success: A Manager's Guide to Object Orientation, its Impact on the Corporation, and its Use for Reengineering the Software Process*, Prentice Hall, Englewood Cliffs NJ.
- Mortenson, M. E. (1989) *Computer Graphics: An Introduction to the Mathematics and Geometry*, Industrial Press, New York.
- Mössenböck, H. (1993) *Object-Oriented Programming in Oberon-2*, Springer-Verlag, London.

- Musser, D. and Saini, A. (1996) *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*, Addison-Wesley, Reading, MA.
- Negroponte, N. (1995) *being digital*, Hodder & Stoughton, London.
- O'Rourke, J. (1994) *Computational Geometry in C*, Cambridge University Press, Cambridge.
- OpenGL (2000) *The OpenGL Web page*, <http://www.opengl.org/>.
- Peitgen, H.-O., Jurgens, H. and Saupe, D. (1992) *Chaos and Fractals, New Fractals of Science*, Springer-Verlag, London.
- Petzold, C. (1992) *Programming Windows 3.1*, 3rd edn, Microsoft Press, Redmond WA.
- Phong, B. T. (1975) *Illumination for Computer-Generated Pictures*, *Commun. ACM*, 18(6), 311–17.
- Pinson, L. J. and Wiener, R. S. (1988) *An Introduction to Object-Oriented Programming and Smalltalk*, Addison-Wesley, Reading MA.
- Plastock, R. A. and Kalley, G. (1986) *Theory and Problems of Computer Graphics*, Schaum's Outline Series, McGraw-Hill, New York.
- Plauger, P. J. (1995) *The Draft Standard C++ Library*, Prentice Hall, Englewood Cliffs NJ.
- Porter, A. (1993) *C++ Programming for Windows*, Osborne McGraw-Hill, Maidenhead.
- Preparata, F. P. and Shamos, M. I. (1985) *Computational Geometry: An Introduction*, Springer-Verlag, London.
- Pugh, W. (1989) *Skip lists: a probabilistic alternative to balanced trees*, in *Proc. Workshop Algorithms and Data Structures*, Ottawa Canada, August 1989.
- Python (2000) *The Python web page*, <http://www.python.org/>.
- Rao, V. B. and Rao, H. V. (1993) *C++ Neural Networks and Fuzzy Logic*, MIS Press, New York.
- Reisdorph, K. and Henderson, K. (1997) *Teach Yourself Borland C++ Builder in 21 Days*, Sams Publishing, Indianapolis.
- Reiser, M. and Wirth, N. (1992) *Programming in Oberon – Steps Beyond Pascal and Modula-2*, Addison-Wesley, Reading MA.
- Richards, M. and Whitney-Stevens, C. (1979) *BCPL: The Language and its Compiler*, Cambridge University Press, Cambridge.
- Ritchie, D. M. (1993) *The Development of the C Language*, ACM SIGPLAN Second History of Programming Languages Conference, 20–23 April 1993, ACM, pp. 201–8.
- Robson, R. (1997) *Using the STL: the C++ Standard Template Library*, Springer-Verlag, New York.
- Rumbaugh, J., Jacobson, I. and Booch, G. (1999) *The Unified Modelling Language Reference Manual*, Addison-Wesley, Reading MA.
- Saunders, J. (1989) *A Survey of Object-Oriented Programming Languages*, *Journal of Object-Oriented Programming*, 1 (6).
- Schildt, H. (1994) *C++ from the Ground Up*, Osborne McGraw-Hill, Maidenhead.
- Schildt, H. (1995) *C++: The Complete Reference*, 2nd edn, Osborne McGraw-Hill, Maidenhead.
- Sedgewick, R. (1988) *Algorithms*, 2nd edn, Addison-Wesley, Reading MA.
- Sedgewick, R. (1992) *Algorithms in C++*, Addison-Wesley, Reading MA.
- Snyder, J. M. and Barr, A. H. (1987) *Raytracing Complex Models Containing Surface Tessellations*, *SIGGRAPH* 87, pp. 119–28.
- Standard (1998) *Programming Languages-C++* (ISO/IEC 14882:1998). The C++ standard can be obtained on-line from the American National Standards Institute (ANSI) at <http://www.ansi.org/>.
- Stevens, T. (1992) *Fractal Programming and Ray Tracing with C++*, Prentice Hall, Englewood Cliffs NJ.
- Stevens, R. T. (1994) *Object-Oriented Graphics Programming in C++*, Academic Press, London.
- Stroustrup, B. (1982) *Classes: An Abstract Data Type Facility for the C Language*, ACM SIGPLAN Notices.
- Stroustrup, B. (1986) *The C++ Programming Language*, Addison-Wesley, Reading MA.

- Stroustrup, B. (1991) *The C++ Programming Language*, 2nd edn, Addison-Wesley, Reading MA.
- Stroustrup, B. (1993a) *A History of C++: 1979–1991*, ACM SIGPLAN Second History of Programming Languages Conference, 20–23 April 1993, ACM, pp. 271–7.
- Stroustrup, B. (1993b) Why consider language extensions?, in *Proc. European C++ User Group Technical Conference*, 7 March–9 July 1993, pp. 50–63.
- Stroustrup, B. (1994) *The Design and Evolution of C++*, Addison-Wesley, Reading MA.
- Stroustrup, B. (1997) *The C++ Programming Language*, 3rd edn, Addison-Wesley, Reading MA.
- Sun (2000) *The Sun Java web page*, <http://www.java.sun.com/>.
- Swan, T. (1991) *Tom Swan's GNU C++ for Linux*, MacMillan, New York.
- Sykes, C. (1994) *No Ordinary Genius, The Illustrated Richard Feynman*, Weidenfield & Nicolson, London.
- Szyperski, C. (1998) *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, Reading MA.
- Thomas, P. and Weedon, R. (1995) *Object-Oriented Programming in Eiffel*, Addison-Wesley, Reading MA.
- USENIX (1987) Sante Fe NM, C++ Workshop, November.
- USENIX (1988) Denver CO, C++ Conference, October.
- USENIX (1990) San Francisco CA, C++ Conference, April.
- USENIX (1991) Washington DC, C++ Conference, April.
- USENIX (1992) Portland OR, C++ Technical Conference, August.
- USENIX (1993) Munich, European C++ User Group Technical Conference, July.
- Watt, A. and Watt, M. (1992) *Advanced Animation and Rendering Techniques: Theory and Practice*, ACM Press, New York.
- Weiss, M. A. (1996) *Algorithms, Data Structures and Problem Solving with C++*, Addison-Wesley, Reading MA.
- Wells, D. and Young, C. (1993) *Raytracing Creations: Generate 3D Photo-Realistic Images on the PC* (with a chapter by Farmer, D.), Waite Group Press, Mill Valley CA.
- Wilt, N. (1994) *Object-Oriented Ray Tracing in C++*, John Wiley & Sons, New York.
- Yao, P. (1994) *Borland C++ 4.0 Programming for Windows*, Borland Press, Scotts Valley CA.

The order of the operators coincides with the precedence levels of Appendix C.

- ::
 - class scope 266
 - global scope 142
- ()
 - function call 38, 132
 - type conversion 332
- [] (array index) 195
 - overloading 328
- > (indirect member access) 412
- . (direct member access) 225, 412
- () (cast) 83, 706
 - overloading 330
- * (pointer dereference) 376
- & (reference) 145
- ! (logical NOT) 104
- ~ (one's complement) 125
- ++ (increment) 75
 - overloading 321
- (decrement) 75
 - overloading 321
- . * (direct member pointer access) 413
- > * (indirect member pointer access) 413
- / (division) 74
 - overloading 319
- * (multiplication) 74
 - overloading 319
- % (modulus) 74
 - overloading 344
- + (addition) 74
 - overloading 316
- (subtraction) 74
 - overloading 316
- << (bit-shift left) 39, 126
 - overloading 338, 364, 723
- >> (bit-shift right) 126
 - overloading 338, 364, 723
- < (less than) 100
 - overloading 324
- > (greater than) 100
 - overloading 324
- <= (less than or equal to) 100
 - overloading 324
- >= (greater than or equal to) 100
 - overloading 324
- == (equal to) 100
 - overloading 324
- != (not equal to) 100
 - overloading 324
- & (bitwise AND) 121
- ^ (bitwise exclusive OR) 124
- | (bitwise OR) 123
- && (logical AND) 102
- || (logical OR) 103
- ? : (conditional) 108
- = (assignment) 74
 - overloading 325, 596
- += (addition and assignment) 75
 - overloading 319
- = (subtraction and assignment) 75
 - overloading 319
- /= (division and assignment) 75
 - overloading 319
- *= (multiplication and assignment) 75
 - overloading 319
- %= (modulus and assignment) 75
- &= (bitwise AND and assignment) 127
- |= (bitwise OR and assignment) 127
- ^= (bitwise exclusive OR and assignment) 127
- <<= (bit-shift left and assignment) 127
- >>= (bit-shift right and assignment) 127
- , (comma) 54
- /* . . . */ (C-style comments) 32
- // (C++-style comments) 32
- abort()* 118, 419, 549
- Abs()* 471
- abstract **class** *see* **class**, abstract
- abstraction 14
- access declaration *see* inheritance, access declaration
- access modifier
 - const** 65
 - volatile** 66
- access permission *see* file, access permission
- access specifier
 - inheritance 581
 - private** 257, 581
 - protected** 585
 - public** 257, 581
- accumulate()* 908
- adaptors 924
 - container 928
 - iterator 924
- address-of operator 371
- adjacent_difference()* 163
- adjacent_find()* 889
- adjustfield 733
- Algol 9
- algorithms
 - sequence 886

- mutating 892
- non-mutating 886
- allocator 939
- allocators 939
- ANSI
 - standard library 169
 - C header files 169
 - C++ header files 173
- append()* 936
- argc* 790
- argv* 790
- arithmetic assignment
 - overloading *see* operator overloading, arithmetic assignment
- array 191
 - data member 302
 - function arguments 208
 - one-dimensional 208
 - two-dimensional 210
 - higher order 202
 - indexing 195
 - of objects 301
 - one-dimensional 197
 - initialisation 197
 - of pointers 397
 - pointers and 390
 - size 193
 - of structures 229
 - two-dimensional 198
 - initialisation 200
- array subscript operator
 - overloading *see* operator overloading, array subscript operator
- ASCII character set 52
- asm** 88
- assert()* 419, 852
- Assert()* 561
- assignment 72
 - operator overloading *see* operator overloading, assignment operator
 - structure objects 227
- assignment suppression character 799
- associative arrays 500
- associative containers 913
- atof()* 791
- atoi()* 791
- atol()* 791
- auto** 84
 - class** *see* **class**, data member, **auto**
- automatic memory management 17
- back_insert_iterator* 925
- bad()* 745
- bad_alloc* 554, 571
- bad_cast* 703
- bad_exception* 553
- bad_typeid* 703
- basefield* 733
- basic_filebuf* 716
- basic_fstream* 718
- basic_ifstream* 717, 737
- basic_ofstream* 718, 741
- basic_ios* 716
- basic_istream* 716
- basic_iostream* 717
- basic_ostream* 717
- basic_streambuf* 716
- basic_string* 932
- bidirectional_iterator* 877
- binary_function* 882
- binary number* 121
- binary operator
 - overloading *see* operator overloading, binary operators
- binary_search()* 901
- bind1st()* 884
- bind2nd()* 884
- bit fields
 - class** *see* **class**, bit fields
 - structure *see* structure, bit fields
- bitmap 818
- BITMAPFILEHEADER 819
- BITMAPINFO 820
- BITMAPINFOHEADER 819
- bitwise operators 121
 - &, AND 121
 - ^, XOR 124
 - |, OR 123
 - ~, one's complement 125
- bool** 62
- Boolean 246
- Borland
 - IDE 26
 - ObjectWindows 254, 301
- bounding box 179
- break** 116
 - loop 116
 - switch** 105
- C++
 - applications 20
 - future 4
 - history 3
 - c_str()* 766
 - calloc()* 410
 - cascading 39
 - case** 105
 - casting 83, 706
 - pointer 407
 - catch** 537
 - all exceptions 543
 - multiple statements 542
 - ceil()* 755
 - cerr* *see* stream, predefined (C++)
 - char** 52
 - char_traits* 714, 931
 - character
 - constant 52
 - string constant 55
 - character traits 714, 931
 - cin* *see* stream, predefined (C++)
 - Circle* 347, 656
 - class** 253
 - abstract 644
 - base 578
 - constructor 588
 - destructor 588
 - bit fields 272
 - constructor 272
 - calling 274
 - const** 285
 - default 275, 277

- default arguments 293
- derived **class** *see* **class**, derived, constructor
- inline** 290
- multiple inheritance 632
- overloaded 276
- volatile** 285
- container 29
- copy constructor 280
 - function calls 283
 - overloading 283
 - inheritance 594
- data member 257
 - auto** 270
 - const** 270
 - extern** 270
 - initialisation 278
 - mutable** 270
 - register** 270
 - static** 268
 - volatile** 271
- declaration 257, 305
- derived 578
 - constructor 589
 - default 589
 - destructor 589
 - default 589
- destructor 285, 588
 - const** 287
 - derived **class** *see* **class**, derived, destructor
 - multiple inheritance 632
 - volatile** 287
- local 302
- member function 260
 - automatic **inline** 289
 - calling 264
 - const** 290
 - default arguments 293
 - inline** 288
 - naming 264
 - object arguments 295
 - out-of-line definition 264
 - returning
 - a local object 436
 - object 295
 - by reference 298
 - static** 268
 - volatile** 292
- method 261
- nested 303
- polymorphic 637, 701
- private**
 - data member 257
- protected**
 - data member 585
- public**
 - data member 257
 - member function 260
- template** *see* **template**, **class**
- virtual** base 635
- virtual** member function 637
- clear()* 745
- clearerr()* 817
- clockwise-anticlockwise 181, 306
- clog* *see* stream, predefined (C++)
- close()* 747
- clreol()* 786
- clrscr()* 786
- code reusability 607
- colour table 819
- command line arguments 165, 789
- comments 32
 - /*...*/* *see* */*...*/*
 - //* *see* *//*
 - nested 33
- compile 41
- compilers 20
- complex 917
- Complex 359
- complex number 358
- composed type 500
- composite operators 330
- compound statement 94
- concurrency 18
- conditional compilation statement 153
- conditional operator 108
- console streams *see* stream, console
- const** 65
 - data member *see* **class**, data member, **const**
 - pointer 382
 - reference arguments 146
 - return value 146
- const_cast<>()* 709
- constant 68
 - character 69
 - decimal 68
 - enumeration 70
 - floating-point 69
 - hexadecimal 69
 - integer 68
 - octal 69
 - string 39, 69
- constream 785
- constructor *see* **class**, constructor
- containers 910
 - associative 913
 - numeric 917
 - sequence 911
- containment 616, 629
 - hasa relationship 630
- continue** 119
- conversion
 - characters *see* *printf()*, conversion characters
 - function 332, 361
- convex hull 524
- copy()* 892
- copy_backward()* 892
- copy constructor, *see* **class**, copy constructor
- count()* 886
- count_if()* 887
- cout* *see* stream, predefined (C++)
- data abstraction 8
- data-hiding 12
- data member *see* **class**, data member
- Date 231
- dec 726
- declaration 71
 - class** *see* **class**, declaration
- decrement operator *see* operator, decrement
- default** 107

- default arguments
 - member function *see* **class**, member function, default arguments
- definition 71
- delete** 414
 - array 421
 - global 430
 - object 414
 - overloading 427
 - inheritance 601
- delline()* 786
- deque 912
- dereferencing 376
- destructor *see* **class**, destructor
- device independent bitmap 818
 - file format 819
- DIBitmap 821
- digraph 56
- direction angles 616
- direction cosines 616
- direct member access operator 225
- disk file *see* file, disk file
- divides 883
- do-while** 114
- domain_error* 554
- double** 59
- double-dispatch 697
- down-cast 697
- dynamic_cast*<>()
 - pointer 700
 - reference 701
- dynamic memory allocation 410
- Eiffel 5
- else** *see* **if-else**
- EmployedPerson* 231
- encapsulation 15
- end of file *see* file, end of
- endl* 40, 726
- ends 726
- enum** 61, 244
- enumerated type 244
- eof()* 745
- epsilon error 60
- equal()* 887
- equal_range()* 901
- equal_to* 883
- escape sequence 55
- evaluation order
 - expressions 73
 - function arguments 142
- exception 553
- exception handling 537
 - constructor 554
 - exception classes 553, 560
 - exception specification 546, 687
 - functions 544
 - inheritance *see* inheritance, exception handling
 - new** operator 570
 - re-throwing an exception 547
- exception specification 687
- exit()* 118, 549
- explicit** 335
- export** 483
- expression 73
- extern** 85, 166
 - class** *see* **class**, data member, **extern**
- external linkage 166
- extraction operator 723
- fabs()* 153
- fail()* 745
- false** 62
- fclose()* 806
- feof()* 817
- ferror()* 817
- fflush()* 818
- fgetc()* 807
- fgetpos()* 815
- fgets()* 809
- FILE 793, 805
- file
 - access permission 744
 - disk file 737
 - end of 745
 - header 150
 - implementation 150
 - objects and 756
 - opening mode 744
 - pointer 752, 814
 - random access
 - (C) 814
 - (C++) 752
 - read
 - (C) 807
 - (C++) 81, 750
 - read and write 812
 - stream classes 714
 - write
 - (C) 810
 - (C++) 81, 750
- file pointer *see* file, pointer
- fill()* 735, 892
- fill_n()* 892
- find()* 888
- find_end()* 888
- find_first_of()* 888
- find_if()* 888
- fixed 733
- flags()* 734
- float** 58
- floatfield 733
- floor()* 755
- flush 726
- flushall()* 818
- fmt_flags* 733
- fopen()* 806
- for** 109
 - nested 116
- for_each()* 890
- format flags *see* ios, format flags
- format string 794
- formatting
 - input 77, 726
 - output 77, 726
- fprintf()* 814
- fputc()* 810
- fputs()* 811
- fread()* 812
- free()* 410
- freeze()* 784
- freopen()* 818

- friend** 347
 - class** 352
 - function 347,484
 - inheritance 605
 - operator overloading *see* operator overloading, **friend**
 - virtual** functions 641
- front_insert_iterator 925
- fscanf() 814
- fseek() 815
- fsetpos() 815
- fstream 718
- ftell() 815
- function 132
 - arguments 38,142
 - default 161
 - pointer 383
 - reference 144
 - const** 146
 - value 143
 - array 208
 - one-dimensional 208
 - string 212
 - two-dimensional 210
 - body 38,133
 - calling 136
 - declaration 138
 - definition 137
 - inline** 163
 - libraries 168
 - member *see* **class**, member function
 - name 38,133
 - objects 881
 - overloading 156
 - pointer to 400
 - prototype *see* function, declaration
 - return** 135
 - const** 146
 - multiple values 147
 - reference 148
 - return type 38,134
 - default 135
 - void** 134
 - reusability 150
 - scope 139
 - global 140
 - local 139
 - precedence 141
 - signature *see* function, declaration
 - template** *see* **template**, function
 - use of 149
- fundamental data type 51
- FuzzyShape 10,646
- fwrite() 812

- Gaussian elimination 461
- gcount() 750
- generate() 893
- generate_n() 893
- get() 205,739
- getche() 114
- getline() 739
- gets() 808
- GlobalMemory 432,495,673
- global variable 269
- good() 745

- goto** 120
- greater 883
- greater_equal 883
- gslice 923

- header file 35,169
- hello, object 31
- hello, world 794
- hex 726
- Hexahedron 245
- highvideo() 786

- if** 92
 - nested 98
 - scope 99
- if-else** 93
- ifstream 718,737
- implementation file 267
- include 35
 - " " 35
 - <> 35
- includes() 903
- increment operator *see* operator, increment
- Index 565
- indirect_array 924
- indirection 370
- indirect member access operator 225,412
- inheritance 15,577,628
 - access declaration 582
 - access specifier 581
 - default 582
 - private** 581
 - protected** 585
 - public** 581
 - declaration 580
 - exception handling 684
 - isa relationship 630
 - member function
 - overloading 602
 - overriding 602
 - multiple 630
 - operator overloading 605
 - template** 664
- initialisation 72
- inline** 163
 - constructor *see* **class**, constructor, **inline**
 - function *see* function, **inline**
 - member function *see* **class**, member function, **inline**
- inner_product() 908
- inplane_merge 902
- insert_iterator 925
- insertion operator 725
- inline() 786
- instance 258
- int** 57
- internal 733
- invalid_argument 554
- ios_base 716
 - format flags
 - boolalpha 733
 - dec 733
 - fixed 733
 - hex 733
 - internal 733
 - left 733

- oct 733
- right 733
- scientific 733
- showbase 733
- showpoint 733
- showpos 733
- skipws 733
- unitbuf 733
- uppercase 733
- adjustfield 733
- basefield 733
- floatfield 733
- ios 716
- iostream 717
- is_open() 749
- isalnum() 722
- isalpha() 722
- isCon() 786
- isdigit() 722
- isspace() 722
- istream 717
- istream_iterator 877
- istrstream 783
- iter_swap() 896
- iterator 502
 - class** 506
 - function 502
- iterators, STL 875
- iterator 877
- Iterator 681
- itoa() 791
- Java 6, 28
- keyword 36, 71
- left 733
- length_error 554
- less 883
- less_equal 883
- lexicographical_compare() 907
- Line 237, 308, 521
- line intersection 175, 306
- link 41
- LinkedList 508, 679
- LinkedListIterator 517, 681
- list 912
- literal 63
- local **class** *see* **class**, local
- locale 736
- LocalMemory 672
- logic_error 554
- logical
 - false 100
 - true 100
- logical_and 883
- logical_not 883
- logical_or 883
- logical operators 101
 - !, NOT 104
 - &&, AND 102
 - ||, OR 103
- long** 65
- loop 109
 - do-while** 114
 - for** 109
 - forever** 115
 - nested** 116
 - scope** 121
 - while** 113
- lower_bound() 901
- lowvideo() 786
- ltoa() 791
- macro 842
 - predefined**
 - __cplusplus 851
 - __DATE__ 851
 - __FILE__ 851
 - __LINE__ 851
 - __STDC__ 851
 - __TIME__ 851
- main() 37, 165, 789
- make_heap() 905
- makefile 43
- malloc() 409
- manipulators 77
 - console 786
 - dec 80
 - endl 79
 - flush 79
 - hex 80
 - non-parametrised 726
 - oct 80
 - parametrised 726
 - resetiosflags() 80
 - setiosflags() 80
 - setprecision() 80
 - setw() 79
 - user-defined**
 - non-parametrised 731
 - parametrised 731
- map 914
- mask_array 923
- Matrix 448, 497, 676
- Max() 471
- mem_fun_t 884
- mem_fun_ref_t 885
- mem_fun1_t 885
- mem_fun1_ref_t 885
- member function *see* **class**, member function
- memcpy() 407
- memory 217
- minus 883
- Memory 672
- memset() 424
- merge() 902
- Mesh 622
- message 261
- method 261
- Min() 150
- mismatch() 888
- modifiers 64
- modularity 15
- modulus 883
- multimap 914
- multiple inheritance *see* inheritance, multiple
- multiplies 883
- multiset 914
- mutable** 270
 - data member *see* **class**, data member,
 - mutable**

- namespace** 855
 - alias 858
 - nameless 863
 - std 40, 874
- negate 883
- nested **class** *see* **class**, nested
- new** 414
 - array 421
 - exception handling *see* exception handling, **new**
 - operator
 - global 430
 - object 414
 - overloading 427
 - inheritance 601
 - placement syntax 435
- Newton–Raphson method 401
- next_permutation()* 906
- normvideo()* 786
- not_equal_to* 883
- not1 884
- not2 884
- nothrow 572
- nothrow_t 572
- nth_element()* 900
- NULL 377
- null termination character 204
- NumberedPoint 608, 756
- numeric_limits 67, 917
- Oberon-2 5
- object 19, 258
 - arrays of 301
 - const** 291
 - default initialisation 275
 - definition 258
 - temporary 299
 - volatile** 293
- object-oriented programming 9
- ObjectWindows *see* Borland, ObjectWindows
- oct 726
- ofstream 718, 741
- omanip2 786
- open()* 747
- opening mode *see* file, opening mode
- operation 261
- operator 74
 - arithmetic 74
 - arity 75
 - associativity 75
 - chaining 74
 - decrement 75
 - increment 75
 - precedence 76
- operator** 317
- operator overloading 18, 315
 - << 338
 - >> 338
 - arithmetic assignment 319
 - inheritance 596
 - array subscript operator 328
 - assignment operator 325
 - binary operators 315
 - composite operators 330
 - extraction operator *see* >>, overloading
 - delete** 414
 - friend** 57
 - inheritance *see* inheritance, operator
 - overloading
 - insertion operator *see* <<, overloading
 - new** 414
 - non-member functions 327
 - operators that cannot be overloaded 343
 - physical meaning 323
 - relational operators 324
 - unary operators 319
- ostream 717
- ostream_iterator 879
- ostrstream 784
- out_of_range 554
- output_iterator 925
- overflow_error 554
- overloading
 - function *see* function, overloading
 - operator *see* operator overloading
- overriding member functions *see* inheritance, member
 - function, overriding
- OwlMain() 166
- pair 915
- palette 819
- partial_sort()* 899
- partial_sort_copy()* 899
- partial_sum()* 909
- partition()* 893
- PathAndFile 760
- pcount()* 784
- peek()* 741
- persistence 18
- Person 222
- PlanarPolygon 662
- plane 650
- Plane 649
- plus 883
- Point 233, 237, 258, 477, 521
- POINT 254
- pointer 369
 - addition 385
 - arithmetic 385
 - to an array 397
 - array of 397
 - casting 407
 - const** 382
 - to data member 413
 - decrement 387
 - to function 400
 - function argument 383
 - increment 387
 - initialisation 377
 - to member function 413
 - to member operators 412
 - NULL 377
 - to objects 410
 - to pointer 395
 - relational operators 389
 - returning 406
 - smart 493
 - to a string constant 400
 - subtraction 386
 - this** 442
 - void** 379
- pointer_to_binary_function 886
- pointer_to_unary_function 886

- polygon**
 - area** 654
 - concave** 519
 - convex** 519
 - point in** 522
- Polygon** 518, 652
- Polyhedra** 662
- polymorphism** 15, 637
- pop_heap()** 905
- pragmatics** 20
- precision()** 735
- predicates** 882
- preprocessor** 841
 - directive** 34, 841
 - #** 842
 - #define** 842
 - #elif** 848
 - #else** 848
 - #endif** 848
 - #error** 845
 - #if** 848
 - #ifdef** 848
 - #ifndef** 848
 - #include** 34, 846
 - #line** 846
 - #pragma** 847
 - #undef** 847
 - defined** 849
- prev_permutation()** 906
- printf()** 794
 - conversion characters** 794
- priority_queue** 930
- private**
 - data member** *see* **class, private**, **data member**
- procedural programming** 6
- programming style** 46
- programming in the large** 267
- protected** 585
- ptrdiff_t** 878
- PtrVector** 667
- public**
 - data member** *see* **class, public**, **data member**
 - member function** *see* **class, public**, **member function**
- push_heap()** 905
- put()** 743
- putback()** 741
- putchar()** 808
-
- qsort()** 381
- Quadrilateral** 656
- queue** 680
- queue** 929
- Queue** 680
-
- rand()** 118
- random_shuffle()** 893
- random file access** *see* **file**, **random access**
- Range** 565
- range_error** 554
- raw_storage_iterator** 927
- rdbuf()** 737, 786
- rdstate()** 745
- read()** 750
- ReadAndWriteFile** 759
- ReadFile** 759
-
- reading objects** 767
- realloc()** 410
- RECT** 249
- redirection** 787
- register** 86
 - class** *see* **class**, **data member**, **register**
- reinterpret_cast<>()** 708
- relational operators** 100
 - overloading** *see* **operator overloading**, **relational operators**
- remove()** 818, 894
- remove_copy()** 894
- remove_copy_if()** 894
- remove_if()** 894
- rename()** 818
- replace()** 894
- replace_copy()** 895
- replace_copy_if()** 895
- replace_if()** 894
- resetiosflags()** 726
- return** 135
- returning by reference**
 - member function** *see* **class**, **member function**, **returning, by reference**
- reverse()** 895
- reverse_bidirectional_iterator** 877
- reverse_copy()** 895
- reverse_iterator** 924
- rewind()** 817
- rfind()** 827
- RGBQUAD** 820
- right** 733
- rotate()** 895
- rotate_copy()** 895
- runtime_error** 554
- run-time type information** 693
- RVector** 665
-
- scanf()** 798
 - conversion characters** 797
 - scanf format specifier** 802
- scientific** 733
- scope** 139
 - function** *see* **function**, **scope**
 - loop** 121
- scope resolution operator** *see* **::**
 - nested class** 303
- search()** 891
- search_n()** 891
- seek_dir** 752
- seekg()** 752
- seekp()** 754
- semantics** 20
- set** 914
- set_intersection()** 903
- set_difference()** 904
- set_new_handler()** 573
- set_symmetric_difference()** 904
- set_terminate()** 549
- set_unexpected()** 551
- set_union()** 903
- setattr()** 787
- setbase()** 726
- setbk()** 787
- setclr()** 787
- setcrsrtype()** 787

- `setf()` 733
- `setfill()` 726
- `setiosflags()` 726
- `setprecision()` 726
- `setstate()` 745
- `setw()` 726
- `setxy()` 787
- Shape 648
- shift operators 121
 - <<,left 126
 - >>,right 126
- short** 65
- showbase 733
- showpoint 733
- showpos 733
- signed** 64
- Simula 9
- `size_t` 765
- sizeof** 70,194
- slice 922
- `slice_array` 922
- Smalltalk 5
- `smanip` 731
- `smanip_full` 731
- smart pointers 493
- `sort()` 898
- `sort_heap()` 906
- SortedVector 668
- SortKeyword 771
- specialisation
 - class
 - complete 487
 - partial 489
 - function 476
- `sprintf()` 805
- square root 151
- `srand()` 118
- `sscanf()` 804
- `stable_partition()` 893
- `stable_sort()` 899
- stack 679
- stack 928
- Stack 679
- standard
 - input 721
 - output 720
- standard template library (STL) 874
- statement
 - compound 94
 - null 39
 - program 38
- static** 87
 - data member *see* **class**, data member, **static**
 - local variables 162
 - member function *see* **class**, member function, **static**
- `static_cast<>()` 706
- std 868,874
- `stderr` 419
- STL *see* standard template library
- storage class specifier
 - auto** 84
 - extern** 85
 - register** 86
 - static** 87
- `str()` 784
- `strcmp()` 381
- `strcpy()` 406
- stream 35,714
 - C approach 793
 - C++ approach 714
 - console 785
 - input
 - (C) 798
 - (C++) 714,721
 - output
 - (C) 794
 - (C++) 39,714,720
 - predefined (C)
 - `stdaux` 793
 - `stderr` 793
 - `stdin` 793
 - `stdout` 793
 - `stdprn` 793
 - predefined (C++)
 - `cerr` 719
 - `cin` 719
 - `clog` 719
 - `cout` 39,719
 - `wcerr` 719
 - `wcin` 719
 - `wclog` 719
 - `wcout` 719
 - status flags 745
 - string 783
- string 203
 - constant 39,206
 - formatting 783
 - variable 203
- string 933
- String 335,417,445
- `strstream` 784
- `strstreambuf` 716,783
- `strtod()` 791
- `strtol()` 791
- `strtoul()` 791
- struct** 221
- structure 221
 - accessing 225
 - anonymous *see* structure, nameless
 - array of 229
 - bit fields 240
 - data member 223
 - public** 237
 - declaration 223
 - default assignment operator 228
 - function argument 229
 - initialiser list 225
 - member function 235
 - name 223
 - nameless 227
 - nested 231
 - object of 224
 - operator overloading 228
 - private** 237
 - public** 237
 - size of 225
 - tag 224
- sub-string 938
- SuperQuadratics 622
- `swap()` 896
- `Swap()` 473

- `swap_ranges()` 896
- switch** 105
- Switch 246
- syntax 19

- `tellg()` 752
- `tellp()` 754
- template** 471
 - class** 477
 - multiple type arguments 485
 - overloading 487
 - default argument 480
 - function 471
 - argument 474
 - multiple type arguments 474
 - overloading 476
 - inheritance *see* inheritance, **template**
 - specialisation 487
- `terminate()` 549
- Tetrahedra 623, 658
- tetrahedron 626
 - centroid 626
 - normal 626
 - perimeter 626
 - surface area 626
 - volume 626
- `textmode()` 786
- this** 442
- throw** 537
- `time()` 118
- `tolower()` 741
- `toupper()` 741
- TPoint 254
- `transform()` 897
- triangle
 - centroid 622
 - normal 622
 - perimeter 622
 - surface area 622
- Triangle 237, 619, 655
- trigraph 56
- true** 62
- try** 537
- two's complement 125
- type 16, 51
- `type_info` 703
- typedef** 247
 - Windows 248
- `typeid()` 702
- typename** 492
- type specifier 38
- typing
 - dynamic 16
 - static 16

- unary_function 882
- underflow_error 554
- `unexpected()` 551
- Unified Modelling Language (UML) 18
- union** 242
 - size of 243
- `unique()` 897
- `unique_copy()` 897

- unitbuf 733
- `unsetf()` 733
- unsigned** 65
- `upper_bound()` 901
- uppercase 733
- using** 860
 - declaration 860
 - directive 861

- valarray 919
- vector 356
 - cross product 181, 615
 - dot product 614
 - norm 182, 614
 - normalised 614
 - unit 614
- vector 910
- Vector 355, 421, 497, 556, 611
- Vector3D 611
- VectorIterator 506, 681
- `vfprintf()` 814
- `vfscanf()` 814
- virtual** 637
 - base **class** *see* **class**, virtual base
 - functions 637
 - constructor 641
 - destructor 641
 - friend** 641
 - inline** 641
 - pure 644
 - static** 641
- void** 134
- volatile** 66
 - data member *see* **class**, data member, **volatile**
- `vprintf()` 805
- `vscanf()` 805
- `vsprintf()` 805
- `vsscanf()` 805

- wchar_t** 60
- WeekDays 245
- wfstream 718
- while** 113
- white space 722
- `width()` 735
- wfstream 718
- `window()` 786
- Windows 672, 818
- `WinMain()` 166, 835
- wios 716
- wiofstream 717
- wofstream 718
- World 769
- world objects 767
- `write()` 750
- WriteFile 759
- ws 726
- wstreambuf 716
- wstring 933

- X(X&) 283
- xmsg 552