How to: Understand and Use Standard Libraries F

By its very nature, the library provided with a programming language is a mixed bag.
P.J. Plauger
The Standard C Library, p. x

F.1 Functions

C++ inherits many free (non-class) functions from C. A **function library** is a collection of cohesive functions that have a common domain. For example, the header file `<cmath>` imports mathematical functions, the file `<cctype>` imports character functions, and the library `<cstdlib>` imports "standard" algorithms like the C-based functions `atoi` and `atof`. In addition to the function libraries inherited from C, C++ includes several standard class libraries. In particular, the *Standard Template Library*, or STL, provides implementations of functions, algorithms, and container classes like vector. We use some of the ideas from STL, for example in the class `tvector` and in the sorting functions of *sortall.h*, but a complete discussion of STL is beyond the scope of this book. Complete though terse information on STL is available in [Str97]; a description of why the library works as it does and a wonderful book on generic programming is [Aus98].

The function libraries imported using header files of the form `<cXXX>` are in *std* namespace. For a brief introduction to namespaces see Section A.2.3 in How to A. Functions in the global namespace are imported using `<XXX.h>`. For example, use `<cmath>` for functions in the *std* namespace, but `<math.h>` for functions in the global namespace. Older libraries/environments typically support only the `.h` versions of the function libraries.

F.1.1 The Library `<cmath>`

Functions in the standard math library, `<cmath>`, are given in Table F.1. On older systems this library is called `<math.h>`. All trigonometric functions use radian measure. See the functions in `mathutils.h`, Program G.9 for functions to convert between degrees and radians.

Most of the functions in `<cmath>` are described sufficiently in Table F.1. The arguments to `atan2` are presumed to be x- and y-coordinates, so that, `atan2(1,1)` is the same as `atan2(3,3)` or `atan(`π`/4)`.

Table F.1 Some functions in <cmath>

function name	prototype	returns
double fabs	(double x)	absolute value of x
double abs	(double x)	absolute value of x (C++only)
double log	(double x)	natural log of x
double log10	(double x)	base-ten log of x
double sin	(double x)	sine of x (x in radians)
double cos	(double x)	cosine of x (x in radians)
double tan	(double x)	tangent of x (x in radians)
double asin	(double x)	arc sine of x $[-\pi/2, \pi/2]$
double acos	(double x)	arc cosine of x $[0, \pi]$
double atan	(double x)	arc tangent of x $[-\pi/2, \pi/2]$
double atan2	(double x, double y)	atan(x/y)
double sinh	(double x)	hyperbolic sine of x
double cosh	(double x)	hyperbolic cosine of x
double tanh	(double x)	hyperbolic tangent of x
double pow	(double x, double y)	x^y
double sqrt	(double x)	$\sqrt{x}$, square root of x
double fmod	(double d, double m)	floating-point remainder d/m
double ldexp	(double d, int i)	d*pow(2,i)
double floor	(double x)	largest integer value $\leq$ x
double ceil	(double x)	smallest integer value $\geq$ x

F.1.2 The Library <cctype>

The functions in <cctype> operate on char values, they're summarized in Table F.2. On older systems this library is called <ctype.h>. You would expect functions with the prefix is, such as islower and isalnum, to have return type bool. However, to ensure compatibility with both C and C++ code, many libraries use integer values for the return type of these predicates in <cctype>. These boolean-valued functions return some nonzero value for true, but this value is not necessarily one. All the functions use int parameters, but arguments are expected to be in the range of legal char values.

F.2 Constants and Limits

Several header files import constants and functions that encapsulate platform-specific limits on the maximum and minimal values of different built-in types. Unfortunately, the C++ standard does not require an int to be represented by 32 bits, nor a double

Table F.2 Some functions in <cctype>

function prototype	returns true when
`int isalpha(int c)`	c is alphabetic (upper or lower case)
`int isalnum(int c)`	c is alphabetic or a digit
`int islower(int c)`	c is a lowercase letter
`int isdigit(int c)`	c is a digit character '0'-'9'
`int iscntrl(int c)`	c is a control character
`int isprint(int c)`	c is printable character including space
`int ispunct(int c)`	c is a punctuation (printable, not space, not alnum)
`int isspace(int c)`	c is any white-space character, `' ','\t','\n','\v', '\r','\f'`
`int isupper(int c)`	c is an uppercase letter
	returns
`int tolower(int c)`	lowercase equivalent of c
`int toupper(int c)`	uppercase equivalent of c

to be represented by 64 bits, although these are the standard sizes on 32-bit computers and are the standard sizes used in languages like Java.

F.2.1 Limits in <climits>

The header file <climits> (or <limits.h>) imports the constants shown in *oldlimits.cpp*, Program F.1. However, the value INT_MIN, for example, is almost certainly a preprocessor #define rather than a C++ constant. Although these constants are simple to use, consider using the constants and classes defined in <limits>, whose use is shown in Program F.2 below.

Program F.1 oldlimits.cpp

```
#include <iostream>
#include <iomanip>          // for setw
#include <climits>
#include <string>
using namespace std;

// illustrates range of values for integral types

const int FIELD_SIZE = 13;            // size of field for output chunk

void Print(const string& type, long low, unsigned long high);

int main()
```

```
{
    cout << setw(FIELD_SIZE) << "type"
         << setw(FIELD_SIZE) << "low"
         << setw(FIELD_SIZE) << "high" << endl << endl;

    Print("char",  CHAR_MIN, CHAR_MAX);
    Print("uchar", 0,        UCHAR_MAX);
    Print("short", SHRT_MIN, SHRT_MAX);
    Print("ushort",0,        USHRT_MAX);
    Print("int",   INT_MIN,  INT_MAX);
    Print("uint",  0,        UINT_MAX);
    Print("long",  LONG_MIN, LONG_MAX);
    Print("ulong", 0,        ULONG_MAX);
    return 0;
}

void Print(const string& type, long int low, unsigned long int high)
// postcondition: values printed in field width FIELD_SIZE
{
    cout << setw(FIELD_SIZE) << type
         << setw(FIELD_SIZE) << low
         << setw(FIELD_SIZE) << high << endl;
}
```

oldlimits.cpp

OUTPUT

```
prompt> oldlimits
          type           low          high

          char          -128           127
         uchar             0           255
         short        -32768         32767
        ushort             0         65535
           int   -2147483648    2147483647
          uint             0    4294967295
          long   -2147483648    2147483647
         ulong             0    4294967295
```

F.2.2 Double Limits in <cfloat>

The header file <cfloat> (or <float.h>) imports several constants including DBL_MIN and DBL_MAX which specify the minimal and maximal double values, respectively.

F.2.3 Limits in <limits>

The header file <limits> imports a templated class numeric_limits that provides values related to all the built-in types. Clients can create versions of numeric_limits for programmer-defined classes. For example, we could create a version for the class BigInt. All the methods and constants in numeric_limits are static, so no variables of type numeric_limits are created.

We use only four of the methods available in the class numeric_limits. There are many more, for example, in the class numeric_limits<double> specifically for floating point values. In the function printLimits we use the standard C++ operator typeid, imported from <typeinfo>. Basically, typeid allows types to be compared for equality, and provides access to a string form of a type's name. For more information on numeric_limits and typeid see [Str97].

Program F.2 limits.cpp

```
#include <iostream>
#include <limits>
#include <typeinfo>
#include <iomanip>
using namespace std;

// print class-specific limits using numeric_limits from <limits>

template <class Type>
void printLimits(const Type& t)
// post: print max,min values and # bits used by t
{
    cout << "\ninformation for " << typeid(t).name() << endl;
    cout << "min =\t"  << numeric_limits<Type>::min()  << endl;
    cout << "max =\t"  << numeric_limits<Type>::max()  << endl;
    cout << "#bits=\t" << numeric_limits<Type>::digits << endl;
    cout << "is integral? "
         << boolalpha << numeric_limits<Type>::is_integer << endl;
}

int main()
{

    printLimits(0);
    printLimits(0u);
    printLimits(0L);
    printLimits('a');
    printLimits(static_cast<unsigned char>('a'));
    printLimits(0.0);
    printLimits(static_cast<float>(0.0));
    return 0;
}
```

limits.cpp

OUTPUT

```
prompt> limits
information for int
min =   -2147483648
max =   2147483647
#bits=  31
is integral? true

information for unsigned int
min =   0
max =   4294967295
#bits=  32
is integral? true

information for long
min =   -2147483648
max =   2147483647
#bits=  31
is integral? true

information for char
min =   -128            actually prints a char, not an int
max =   128             actually prints a char, not an int
#bits=  7
is integral? true

information for unsigned char
min =   0
max =   255
#bits=  8
is integral? true

information for double
min =   2.22507e-308
max =   1.79769e+308
#bits=  53              # bits in mantissa
is integral? false

information for float
min =   1.17549e-38
max =   3.40282e+38
#bits=  24              # bits in mantissa
is integral? false
```

F.2.4 ASCII Values

Since most C++ environments use ASCII coding for characters, Table F.3 provides ASCII values for all the standard characters.

Table F.3 ASCII values

ASCII character set

decimal	char	decimal	char	decimal	char	decimal	char
0	^@	32	space	64	@	96	‘
1	^A	33	!	65	A	97	a
2	^B	34	"	66	B	98	b
3	^C	35	#	67	C	99	c
4	^D	36	$	68	D	100	d
5	^E	37	%	69	E	101	e
6	^F	38	&	70	F	102	f
7	^G	39	’	71	G	103	g
8	^H	40	(	72	H	104	h
9	^I	41	)	73	I	105	i
10	^J	42	*	74	J	106	j
11	^K	43	+	75	K	107	k
12	^L	44	,	76	L	108	l
13	^M	45	-	77	M	109	m
14	^N	46	.	78	N	110	n
15	^O	47	/	79	O	111	o
16	^P	48	0	80	P	112	p
17	^Q	49	1	81	Q	113	q
18	^R	50	2	82	R	114	r
19	^S	51	3	83	S	115	s
20	^T	52	4	84	T	116	t
21	^U	53	5	85	U	117	u
22	^V	54	6	86	V	118	v
23	^W	55	7	87	W	119	w
24	^X	56	8	88	X	120	x
25	^Y	57	9	89	Y	121	y
26	^Z	58	:	90	Z	122	z
27	escape	59	;	91	[	123	{
28	fs	60	<	92	\	124	|
29	gs	61	=	93	]	125	}
30	rs	62	>	94	^	126	~
31	us	63	?	95	_	127	del

Bibliography

[AA85] Donald J. Albers and G.L. Alexanderson. *Mathematical People*. Birkhäuser, 1985.

[ACM87] ACM. *Turing Award Lectures: The First Twenty Years 1966–1985*. ACM Press, 1987.

[AS96] Harold Abelson and Gerald Jay Sussman. *Structure and Interpretation of Computer Programs*. 2nd ed. MIT Press and McGraw-Hill, 1996.

[Asp90] William Aspray. *Computing Before Computers*. Iowa State University Press, 1990.

[Aus98] Matthew H. Austern *Generic Programming and the STL*. Addison-Wesley, 1998.

[Ben86] Jon Bentley. *Programming Pearls*. Addison-Wesley, 1986.

[Ben88] Jon Bentley. *More Programming Pearls*. Addison-Wesley, 1988.

[Ble90] Guy E. Blelloch. *Vector Models for Data-Parallel Computing*. MIT Press, 1990.

[Boo91] Grady Booch. *Object Oriented Design with Applications*. Benjamin Cummings, 1991.

[Boo94] Grady Booch. *Object Oriented Design and Analysis with Applications*. 2nd ed. Benjamin Cummings, 1994.

[BRE71] I. Barrodale, F.D. Roberts, and B.L. Ehle. *Elementary Computer Applications in Science Engineering and Business*. John Wiley & Sons Inc., 1971.

[Coo87] Doug Cooper. *Condensed Pascal*. W.W. Norton, 1987.

[Dij82] Edsger W. Dijkstra. *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1982.

[DR90] Nachum Dershowitz and Edward M. Reingold. Calendrical calculations. *Software-Practice and Experience*, 20(9):899–928, September 1990.

[(ed91] Allen B. Tucker (ed.). *Computing Curricula 1991 Report of the ACM/IEEE-CS Joint Curriculum Task Force*. ACM Press, 1991.

[EL94] Susan Epstein and Joanne Luciano, editors. *Grace Hopper Celebration of Women in Computing*. Computing Research Association, 1994. HopperBook@cra.org.

[Emm93] Michele Emmer, editor. *The Visual Mind: Art and Mathematics*. MIT Press, 1993.

[GÖ95] Denise W. Gürer. Pioneering women in computer science. *Communications of the ACM*, 38(1):45–54, January 1995.

[Gar95] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1995.

[GHJ95] Erich Gamma and Richard Helm and Ralph Johnson and John Vlissides *Design Patterns: Elements of Reusable Object-Oriented Programming* Addison-Wesley, 1995

[Gol93] Herman H. Goldstine. *The Computer from Pascal to von Neumann*. Princeton University Press, 1993.

[Gri74] David Gries. On structured programming - a reply to smoliar. *Communications of the ACM*, 17(11):655–657, 1974.

[GS93] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag, 1993.

[Har92] David Harel. *Algorithmics, The Spirit of Computing*. Addison-Wesley, second edition, 1992.

[Hoa89] C.A.R. Hoare. *Essays in Computing Science*. Prentice-Hall, 1989. (editor) C.B. Jones.

[Hod83] Andrew Hodges. *Alan Turing: The Enigma*. Simon & Schuster, 1983.

[Hor92] John Horgan. Claude e. shannon. *IEEE Spectrum*, April 1992.

[JW89] William Strunk Jr. and E.B. White. *The Elements of Style*. MacMillan Publishing Co., third edition, 1989.

[Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 1 Fundamental Algorithms. Addison-Wesley, third edition, 1997.

[Knu98a] Donald E. Knuth. *The Art of Computer Programming*, volume 2 Seminumerical Algorithms. Addison-Wesley, third edition, 1998.

[Knu98b] Donald E. Knuth. *The Art of Computer Programming*, volume 3 Sorting and Searching. Addison-Wesley, third edition 1998.

[KR78] Brian W. Kernighan and Dennis Ritchie. *The C Programming Language*. Prentice-Hall, 1978.

[KR96] Samuel N. Kamin and Edward M. Reingold. *Programming with class: A C++ Introduction to Computer Science*. McGraw-Hill, 1996.

[Mac92] Norman Macrae. *John von Neumann*. Pantheon Books, 1992.

[McC79] Pamela McCorduck. *Machines Who Think*. W.H. Freeman and Company, 1979.

[McC93] Steve McConnell. *Code Complete*. Microsoft Press, 1993.

[MGRS91] Albert R. Meyer, John V. Gutag, Ronald L. Rivest, and Peter Szolovits, editors. *Research Directions in Computer Science: An MIT Perspective*. MIT Press, 1991.

[Neu95] Peter G. Neumann. *Computer Related Risks*. Addison Wesley, 1995.

[Pat96] Richard E. Pattis. *Get A-Life: Advice for the Beginning Object-Oriented Programmer*. Turing TarPit Press, 1999.

[Per87] Alan Perlis. The synthesis of algorithmic systems. In *ACM Turing Award Lectures: The First Twenty Years*. ACM Press, 1987.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[RDC93] Edward M. Reingold, Nachum Dershowitz, and Stewart M. Clamen. Calendrical calculations, ii: Three historical calendars. *Software-Practice and Experience*, 23(4):383–404, April 1993.

[Rie96] Arthur Riel. *Object-Oriented Design Heuristics*. Addison-Wesley, 1996.

[Rob95] Eric S. Roberts. Loop exits and structured programming: Reopening the debate. In *Papers of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education*, pages 268–272. ACM Press, March 1995. SIGCSE Bulletin V. 27 N 1.

[Rob95] Eric S. Roberts. *The Art and Science of C*. Addison-Wesley, 1995.

[Sla87] Robert Slater. *Portraits in Silicon*. MIT Press, 1987.

[Str87] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, 1987.

[Str94] Bjarne Stroustrup. *The Design and Evolution of C++*. Addison-Wesley, 1994.

[Str97] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, third edition, 1997.

[Mey92] Scott Meyers. *Effective C++*. Addison Wesley, 1992.

[Mey96] Scott Meyers. *More Effective C++*. Addison-Wesley, 1996.

[Wei94] Mark Allen Weiss. *Data Structures and Algorithm Analysis in C++*. Benjamin Cummings, 1994.

[Wil56] M.V. Wilkes. *Automatic Digital Computers*. John Wiley & Sons, Inc., 1956.

[Wil87] Maurice V. Wilkes. Computers then and now. In *ACM Turing Award Lectures: The First Twenty Years*, pages 197–205. ACM Press, 1987.

[Wil95] Maurice V. Wilkes. *Computing Perspectives*. Morgan Kaufmann, 1995.

[Wir87] Niklaus Wirth. From programming language design to compiler construction. In *ACM Turing Award Lectures: The First Twenty Years*. ACM Press, 1987.