

<stdlib.h>, or <cstdlib>

String conversion

double <b>atof</b> (char const * <b>str</b> )	Parse str as a double; return 0.0 on failure
int <b>atoi</b> (char const * <b>str</b> )	Parse str as an int; return 0 on failure
long int <b>atol</b> (char const * <b>str</b> )	Parse str as a long int; return 0 on failure
double <b>strtod</b> (char const * <b>str</b> , char ** <b>endptr</b> )	Parse str as a double; return 0 on failure; set endptr to next char
long int <b>strtol</b> (char const * <b>str</b> , char ** <b>endptr</b> , int <b>base</b> )	As strtod; parses as int in given base. Use strtoul for unsigned.

Binary search

void * <b>bsearch</b> (void const * <b>key</b> , void const * <b>base</b> , size_t <b>num</b> , size_t <b>size</b> , int (* <b>comparator</b> )(void const *, void const *))	Search key (needle). Base of array (haystack). Elements in base. Size of one base element. Function taking two pointers to elements, and comparing them, <i>à la</i> strcmp ( <i>op. cit.</i> ).
--	--

Sorting

void <b>qsort</b> (void * <b>base</b> , size_t <b>num</b> , size_t <b>size</b> , int (* <b>comparator</b> )(const void *, const void *))	First element of array to sort. Number of elements in array. Size of one array element. Function taking two pointers to elements, and comparing them, <i>à la</i> strcmp ( <i>op. cit.</i> ).
--	--

Memory management

void * <b>malloc</b> (size_t <b>size</b> )	Get pointer to new heap-allocated block of specified size
void * <b>calloc</b> (size_t <b>num</b> , size_t <b>size</b> )	As with malloc; specified size num * size
void * <b>realloc</b> (void * <b>ptr</b> , size_t <b>size</b> )	Make ptr specified size, in-place if possible; return pointer to it
void <b>free</b> (void * <b>ptr</b> )	Deallocate block specified by ptr

Environment

void <b>abort</b> ()	Abort process with SIGABRT signal
void <b>exit</b> (int <b>status</b> )	Terminate; return status as exit code
int <b>atexit</b> ( void (* <b>function</b> )(void))	Register function to run at exit (in reverse order of registration)
char * <b>getenv</b> (char const * <b>name</b> )	Get environment variable name, or NULL if not present
int <b>system</b> (char const * <b>command</b> )	Synchronously run new process command; return is system-dependent

Pseudo-random sequence generation

int <b>rand</b> ()	Pseudo-random integral number, [0, RAND_MAX].
void <b>srand</b> (unsigned int <b>seed</b> )	Initialize generator using seed.

```
#include <stdlib.h>
#include <time.h>
int main () {
    srand ( time(NULL) );
    /* use rand() */
}
```

<string.h>, or <cstring>

void * <b>memset</b> (void * <b>p</b> , int <b>c</b> , size_t <b>n</b> )	writes n copies of c starting at p
char * <b>strcat</b> (char * <b>dest</b> , char const * <b>src</b> )	appends the string src to dest
char * <b>strcpy</b> (char * <b>dest</b> , char const * <b>src</b> )	copy src to dest; return dest
char * <b>strchr</b> (char const * <b>str</b> , int <b>char</b> )	First position of char in str; NULL if not found
char * <b>strrchr</b> (char const * <b>str</b> , int <b>char</b> )	As strchr, searching from the end
int <b>strcmp</b> (char const * <b>s1</b> , char const * <b>s2</b> )	compare s1 and s2; 0 → equal; >0 → s1>s2; <0 → s1<s2
size_t <b>strspn</b> (char const * <b>s</b> , char const * <b>accept</b> )	length of prefix of s consisting of characters in accept
size_t <b>strcspn</b> (char const * <b>s</b> , char const * <b>reject</b> )	length of prefix of s consisting of characters not in reject
char * <b>strpbrk</b> (char const * <b>s</b> , char const * <b>accept</b> )	first character in s that occurs in accept; NULL if none found
char * <b>strstr</b> (char const * <b>h</b> , char const * <b>n</b> )	first occurrence of n in h; or NULL if none found
char * <b>strdup</b> (char const * <b>src</b> )	returns a pointer to malloc'd copy of src
size_t <b>strlen</b> (char const * <b>s</b> )	number of bytes from s until occurrence of NULL

<stdarg.h>, or <stdarg.h>

```
#include <stdio.h>
#include <stdarg.h>
void printints(FILE * f, ...) {
    int i;
    va_list ap;
    va_start(ap, f);
    while(i = va_arg(ap, int)) printf("%d ", i);
    va_end(ap);
}
void main() {
    printints(stdout, 1, 2, 3, 4); /* Prints: 1 2 3 4 */
    printints(stdout, 1, 2, 0, 4); /* Prints: 1 2 */
}
```

<stdint.h>, or <cstdint>

Provides type synonyms: int8\_t, uint8\_t, int16\_t, uint16\_t, int32\_t, uint32\_t, int64\_t, uint64\_t, and for each, constants X\_MAX and X\_MIN, where X is the capitalized form of the type name.

<math.h>, or <cmath>

Constants			
M_E	<i>e</i>	M_PI_4	$\pi/4$
M_LOG2E	$\log_2 e$	M_1_PI	$1/\pi$
M_LOG10E	$\log_{10} e$	M_2_PI	$2 \times 1/\pi$
M_LN2	$\ln(2)$	M_2_SQRTPI	$2 \times 1/\sqrt{\pi}$
M_LN10	$\ln(10)$	M_SQRT2	$\sqrt{2}$
M_PI	$\pi$	M_SQRT1_2	$\sqrt{1/2}$
M_PI_2	$\pi/2$		

Trig functions

Use radians. Defined for float, double, and long double.			
cos	sin	tan	
acos	asin	atan	atan2(y, x)
cosh	sinh	tanh	

Exponential and logarithmic functions

The typename T may be float, double, or long double.	
T exp(T x)	$e^x$
T frexp(T x, int *exp)	Breaks x into sig × 2 <sup>exp</sup> ; returns sig
T ldexp(T sig, int exp)	sig × 2 <sup>exp</sup>
T log(T x)	$\ln x$
T log10(T x)	$\log_{10} x$
T modf(T x, T * i)	floor(x) in i; returns remainder

Power functions

T pow(T base, T exp)	base <sup>exp</sup>
T sqrt(T x)	$\sqrt{x}$

Rounding functions

T ceil(T x)	The next whole number above x
T fabs(T x)	The absolute value of x
T floor(T x)	The previous whole number below x
T fmod(T num, denom)	Remainder of num/denom

<ctype.h>, or <cctype>

This defines tolower and toupper, in the form int toX(int c). Returns isalpha(c) ? the converted character : c. The following are in the form int isX(int). Note all parameters are ints – cast all chars to uint_8.	
isalnum	alphanumeric character
isalpha	alphabetic character
isupper	uppercase character
islower	lowercase character
ispunct	punctuation character

isdigit	digit
isxdigit	hexadecimal digit
isprint	printable character, including space
isgraph	graphic character, excluding space
isspace	whitespace character
isblank	blank character
iscntrl	control character

<istream>

int <b>get</b> ()	Get the next character cast as an int
istream& <b>get</b> (char & <b>c</b> )	Put the next character into c
istream& <b>getline</b> (char * <b>s</b> , streamsize <b>n</b> , char <b>delim</b> ='\\n')	Get chars into s until n-1 chars, or delim, or EOF. Discard delim.
int <b>peek</b> ()	Get the next character without extracting it
istream& <b>putback</b> (char <b>c</b> )	Put the character c back into the stream as the next to be read
ostream& <b>operator&lt;&lt;</b> (T& <b>val</b> )	Attempt to parse incoming chars as a value of type T

bool <b>eof</b> () const	Is the eof bit set?
--------------------------	---------------------

<ostream>

ostream& <b>put</b> (char <b>c</b> )	Write c to the output stream
ostream& <b>operator&lt;&lt;</b> (T <b>val</b> )	For a type T, serialize val and stream it out
ostream& <b>write</b> (char const * <b>s</b> , streamsize <b>n</b> )	Write n characters out (does not terminate at NULL!)

Overloading operator<< for your classes

```
class X {
    char * name;
public:
    friend ostream & operator<<(ostream&, X&);
};
ostream & X::operator<<(ostream& out, X& x) {
    out << x.name;
};
```

I/O manipulators

dec, hex, oct	integrals are inserted in this base
endl	insert newline; flush buffer
fixed, scientific	floats are inserted in this notation
flush	flush the buffer
left, right	justify fixed-width output left or right
[no]showbase	numbers prefixed with base format prefix
[no]showpoint	decimal point even for whole numbers
[no]boolalpha	bools are inserted as their names
[no]showpos	'+' precedes every non-negative value
[no]unitbuf	buffer is flushed after every insertion
[no]uppercase	letters in numerals are uppercased
setfill(char)	custom character for padding
setprecision(int)	max digits to express floating-point values
setw(int)	minimum width of next insertion

## <assert.h>, or <cassert>

This provides the macro `assert(expr)`. Use liberally at every point that something should be expected. To remove effects of runtime assertion, `#define NDEBUG`.

## STL templates: vector<T> and map<T, T>

```
<vector>

vector<int> v;
v.push_back(1);
v.push_back(2);
for(vector<int>::iterator i = v.begin(); i!=v.end(); ++i) {
    cout << *i << ' '; // Output: 1 2
}
while(!v.empty()) {
    cout << v.back() << ' '; // Output: 2 1
    v.pop_back();
}
```

```
<map>

typedef map<int, char> IMap;
IMap m;
m[1000] = 'a';
m[2000] = 'b';
for(IMap::iterator it = m.begin(); it != m.end(); ++it)
    cout << it->first << ": " << it->second << endl;
```

## Sample makefile

```
ccc = g++ -Wall -g -c
objects = binary.o object1.o object2.o
binary: $(objects)
    g++ -o binary $(objects)

binary.o: binary.cpp [included headers]
    $(ccc) binary.cpp

object1.o: object1.cpp [included headers]
    $(ccc) object1.cpp

object2.o: object2.cpp [included headers]
    $(ccc) object2.cpp
```

## Sample header file contents

```
#ifndef <PROJECT>_<PATH>_<FILE>_H_
#define <PROJECT>_<PATH>_<FILE>_H_

#include <string.h> // Headers for types
using namespace std;

enum { PROTO, TYPE }; // Type definitions

/** Description of prototype(). */
int prototype(int, int); // Exported function prototypes

#endif // <PROJECT>_<PATH>_<FILE>_H_
```

## Sample implementation file contents

```
#include <string.h> // Library headers
using namespace std;
```

```
#include "../other.h" // Local headers
#include "../header.h" // Header file for this implementation
int prototype(int, int) {return 0;}; // Implement prototypes
```

## ASCII codes

Character	Decimal	Hex	Binary
<b>0</b>	48	30	0011 0000
<b>9</b>	57	39	0011 1001
<b>A</b>	65	41	0100 0001
<b>Z</b>	90	5A	0101 1010
<b>a</b>	97	61	0110 0001
<b>z</b>	122	7A	0111 1010

## Function pointers

```
typedef int (*i2i)(int); // int→int; ALWAYS typedef!
int inc(int i) { return i+1; } // an i2i
class X { public: static int dbl(int i) { return i*2; }};
// X::dbl is an i2i
typedef i2i (*b2_i2i)(bool); // bool→(int→int)
i2i baz(bool b) { return b ? &(X::dbl) : &inc; } // a b2_i2i
[...]
```

```
b2_i2i foo = &baz; // foo is &(bool→(int→int))
i2i bar = (*foo)(true); // bar == &(X::dbl)
cout << (*bar)(-5) << endl; // Out: -10
bar = (*foo)(false); // bar == &inc
cout << (*bar)(-5) << endl; // Out: -4
```

## Operators

Unary prefix operators

+	-	--	++	!	~	*	&
(type)	new	new[]	delete	delete[]			

Unary postfix operators

--	++	[]	(...)
----	----	----	-------

Binary infix operators

Unoverloadable operators are given a border.

=	+	-	*	/	%	==	!=
>	<	>=	<=	&&		&	
^	<<	>>	+=	-=	*=	/=	%=
&=	=	^=	<<=	>>=	->	.	->*
.*	::	,					

## Operator overloading

In the following, ‘##’ is any operator.

Standard infix operators

In class:        T T::operator ##(T const & b) const;  
Outside class:   T operator ##(T const & a, T const & b);

Unary prefix

In class:        T T::operator ##() const;  
Outside class:   T operator ##(T const & a);

Unary postfix ++ and --

In class:        T T::operator ##(int) const;  
Outside class:   T operator ##(int);

Array subscript

In class:        R& T::operator [](T2 const & b);  
Outside class:   *not available*

new and delete

Only in-class definition considered. new and new[] have the same prototype, as do delete and delete[].

```
void* T::operator new (     size_t x );
void T::operator delete ( void * x );
```

## C++ new and delete

Replacement for malloc/free allocation, but not interoperable therewith (*i.e.*, you may not malloc then delete, nor new then free). There are two pairs of operators: new and delete for atomic data types, and new[] and delete[] for arrays.

Atomic types	Arrays
int * i = <b>new</b> int;	int * i = <b>new</b> int[100];
// Use i	// Use i[0..100]
<b>delete</b> i;	<b>delete[]</b> i;

## Constructors and destructors

```
class X {
    char * name;
    Player(char const *); // Constructor
    ~Player();           // Destructor
};

X::X(char const * name) {
    this->name = strdup(name);
}

X::~~X() { // Free all memory pointed to
    free(this->name);
}
```

## printf and scanf formats

A format takes the form:

**%**[flags][width][.precision][length]**type**

Flags

<b>#</b>	trailing 0s, decimal point, base prefixes always remain
<b>[number]</b>	Pad with spaces to this length in chars
<b>0</b>	Pad with zeros
<b>-</b>	Left align
<b>+</b>	Explicit sign symbol
<b>(space)</b>	Leave space for ‘+’
<b>’</b>	Group thousands
<b>I</b>	Use locale digits

Width

<b>[number]</b>	Pad with spaces to this length in chars
<b>*</b>	Pad to length specified immediately prior to parameter

Precision

<b>[number]</b>	Specifies decimal places (for char*, max length)
<b>*</b>	<i>as in ‘Width’ field</i>

Length

hh	For ints, expect an int promoted from a char
h	For ints, expect a int promoted from a short
l	For ints, expect a long

ll	For ints, expect a long long
L	For doubles, expect a long double
z	For ints, expect a size_t-sized int
j	For ints, expect a intmax_t-sized int
t	For ints, expect a ptrdiff_t-sized int

Type

d, i	Decimal signed int
u	Decimal unsigned int
f, F	A double in fixed-point notation
e, E	A double in scientific notation
g, G	A double, fixed-point or scientific based on magnitude
x, X	Hexadecimal unsigned int
o	Octal unsigned int
s	A NULL-terminated char*
c	A single character
p	A void*; implementation-defined
n	Nothing. Place number of output chars in the parameter
%	A literal % character

## Useful alternative C++ tokens

<b>and</b>	<b>&amp;&amp;</b>	<b>bitand</b>	<b>&amp;</b>	<b>not</b>	<b>!</b>
<b>or</b>	<b>  </b>	<b>bitor</b>	<b> </b>	<b>compl</b>	<b>~</b>
		<b>xor</b>	<b>^</b>		