<stdlib.h>, or <cstdlib>

String conversion

`double atof(char const * str)`
Parse `str` as a `double`; return 0.0 on failure

`int atoi(char const * str)`
Parse `str` as an `int`; return 0 on failure

`long int atol(char const * str)`
Parse `str` as a `long int`; return 0 on failure

`double strtod(char const * str, char ** endptr)`
Parse `str` as a `double`; return 0 on failure; set `endptr` to next char

`long int strtol(char const *str, char **endptr, int base)`
As `strtod`; parses as `int` in given `base`. Use `strtoul` for unsigned.

Pseudo-random sequence generation

`int rand()` — Pseudo-random integral number, [0, `RAND_MAX`].

`void srand(unsigned int seed)` — Initialize generator using seed.

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

Memory management

`void * malloc(size_t size)`
Get pointer to new heap-allocated block of specified size

`void * calloc(size_t num, size_t size)`
As with malloc; specified size `num * size`

`void * realloc(void * ptr, size_t size)`
Make `ptr` specified `size`, in-place if possible; return pointer to it

`void free(void * ptr)` — Deallocate block specified by `ptr`

Environment

`void abort()` — Abort process with `SIGABRT` signal

`void exit(int status)` — Terminate; return `status` as exit code

`int atexit( void (*function)(void))`
Register `function` to run at exit (in reverse order of registration)

`char * getenv(char const * name)`
Get environment variable `name`, or `NULL` if not present

`int system(char const * command)`
Synchronously run `command`; return value is system-dependent

Binary search

`void * bsearch(void const * key, void const * base, size_t num, size_t size, int (*comparator)( void const *, void const * ))`

`key`	Search key (needle).
`base`	Base of array (haystack).
`num`	Elements in base.
`size`	Size of one base element.
`comparator`	Function taking two pointers to elements, and comparing them, *à la* `strcmp` (*op. cit.*).

Sorting

`void qsort(void * base, size_t num, size_t size, int (*comparator)( const void *, const void * ))`

`base`	First element of array to sort.
`num`	Number of elements in array.
`size`	Size of one array element.
`comparator`	Function taking two pointers to elements, and comparing them, *à la* `strcmp` (*op. cit.*).

<string.h>, or <cstring>

`void * memset(void * p, int c, size_t n)`
writes `n` copies of `c` starting at `p`

`char * strcat(char * dest, char const * src)`
appends the string `src` to `dest`

`char * strcpy(char * dest, char const * src)`
copy `src` to `dest`; return `dest`

`char * strchr(char const * str, int char)`
First position of `char` in `str`; `NULL` if not found

`char * strrchr(char const * str, int char)`
As `strchr`, searching from the end

`int strcmp(char const * s1, char const * s2)`
compare `s1` and `s2`; 0 → equal; >0 → `s1>s2`; <0 → `s1<s2`

`size_t strspn(char const * s, char const * accept)`
length of prefix of `s` consisting of characters in `accept`

`size_t strcspn(char const * s, char const * reject)`
length of prefix of `s` consisting of characters not in `reject`

`char * strpbrk(char const * s, char const * accept)`
first character in `s` that occurs in `accept`; `NULL` if none found

`char * strstr(char const * h, char const * n)`
first occurrence of `n` in `h`; or `NULL` if none found

`char * strdup(char const * src)`
returns a pointer to `malloc`'d copy of `src`

`size_t strlen(char const * s)`
number of bytes from `s` until occurrence of `NULL`

<stdarg.h>, or <cstdarg>

```
#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`	e	`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`	π	`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^{exp}; returns sig
`T ldexp(T sig, int exp)`	sig × 2^{exp}
`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^{exp}
`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 `int`s – 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

<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`.

<istream>

`int get()` — Get the next character cast as an `int`

`istream& get(char & c)` — Put the next character into `c`

`istream& getline(char * s, streamsize n, char delim='\n')`
Get chars into `s` until `n-1` chars, or `delim`, or EOF. Discard `delim`.

`int peek()` — Get the next character without extracting it

`istream& putback(char c)`
Put the character `c` back into the stream as the next to be read

`ostream& operator>>(T& val)`
Attempt to parse incoming chars as a value of type `T`

`bool eof() const` — Is the eof bit set?

<ostream>

`ostream& put(char c)` — Write `c` to the output stream

`ostream& operator<<(T val)`
For a type `T`, serialize `val` and stream it out

`ostream& write(char const * s, streamsize n)`
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`	`bool`s 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

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
0	48	30	0011 0000
9	57	39	0011 1001
A	65	41	0100 0001
Z	90	5A	0101 1010
a	97	61	0110 0001
z	122	7A	0111 1010

Operators

Unary prefix operators

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

Unary postfix operators

--	++	[]	(...)

Binary infix operators

Unoverloadable operators are greyed out.

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

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

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. There are two pairs of operators: `new` and `delete` for atomic data types, and `new[]` and `delete[]` for arrays.

Atomic types

```
int * i = new int;
// Use i
delete i;
```

Arrays

```
int * i = new int[100];
// Use i[0..100]
delete[] 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]specifier`

Flags

#	Alternate
[number]	Pad with spaces to this length in chars
0	Pad with zeros
-	Left align
+	Explicit sign symbol
(space)	Leave space for '+'
'	Group thousands
I	Use locale digits

Width

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

Precision

[number]	Specifies decimal places (for char*, max length)
*	*as in 'Width' field*

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