```
struct T {                  // Equivalent to: class T { public:
  virtual void f();         // May be overridden at run time by derived
class
  virtual void g()=0; };    // Must be overridden (pure virtual)
class U: public T {};       // Derived class U inherits all members of base
T
class V: private T {};      // Inherited members of T become private
class W: public T, public U {};  // Multiple inheritance
class X: public virtual T {}; // Classes derived from X have base T
directly
```

All classes have a default copy constructor, assignment operator, and destructor, which perform the corresponding operations on each data member and each base class as shown above. There is also a default no-argument constructor (required to create arrays) if the class has no constructors. Constructors, assignment, and destructors do not inherit.

TEMPLATES

```
template <class T> T f(T t);         // Overload f for all types
template <class T> class X {         // Class with type parameter T
  X(T t); };                         // A constructor
template <class T> X<T>::X(T t) {}  // Definition of constructor
X<int> x(3);                         // An object of type "X of int"
template <class T, class U=T, int n=0>  // Template with default
parameters
```

NAMESPACES

```
namespace N {class T {};} // Hide name T
N::T t;                   // Use name T in namespace N
using namespace N;        // Make T visible without N::
```

C/C++ STANDARD LIBRARY

Only the most commonly used functions are listed. Header files without .h are in namespace std. File names are actually lower case.

STDIO.H, CSTDIO (Input/output)

```
FILE* f=fopen("filename", "r");  // Open for reading, NULL (0) if error
  // Mode may also be "w" (write) "a" append, "a+" update, "rb" binary
fclose(f);                  // Close file f
fprintf(f, "x=%d", 3);      // Print "x=3"  Other conversions:
  "%5d %u %-8ld"              // int width 5, unsigned int, long left just.
  "%o %x %X %lx"              // octal, hex, HEX, long hex
  "%f %5.1f"                  // float or double: 123.000000, 123.0
  "%e %g"                     // 1.23e2, use either f or g
  "%c %s"                     // char, char*
  "%%"                        // %
sprintf(s, "x=%d", 3);      // Print to array of char s
printf("x=%d", 3);          // Print to stdout (screen unless redirected)
fprintf(stderr, ...         // Print to standard error (not redirected)
getc(f);                    // Read one char (as an int) or EOF from f
ungetc(c, f);               // Put back one c to f
getchar();                  // getc(stdin);
putc(c, f)                  // fprintf(f, "%c", c);
putchar(c);                 // putc(c, stdout);
fgets(s, n, f);             // Read line into char s[n] from f.  NULL if EOF
gets(s)                     // fgets(s, INT_MAX, f); no bounds check
fread(s, n, 1, f);          // Read n bytes from f to s, return number read
fwrite(s, n, 1, f);         // Write n bytes of s to f, return number
written
fflush(f);                  // Force buffered writes to f
fseek(f, n, SEEK_SET);      // Position binary file f at n
ftell(f);                   // Position in f, -1L if error
rewind(f);                  // fseek(f, 0L, SEEK_SET); clearerr(f);
feof(f);                    // Is f at end of file?
ferror(f);                  // Error in f?
perror(s);                  // Print char* s and error message
clearerr(f);                // Clear error code for f
remove("filename");         // Delete file, return 0 if OK
rename("old", "new");       // Rename file, return 0 if OK
f = tmpfile();              // Create temporary file in mode "wb+"
tmpnam(s);                  // Put a unique file name in char s[L_tmpnam]
```

STDLIB.H, CSTDLIB (Misc. functions)

```
atof(s); atol(s); atoi(s);// Convert char* s to float, long, int
rand(), srand(seed);        // Random int 0 to RAND_MAX, reset rand()
void* p = malloc(n);        // Allocate n bytes.  Obsolete: use new
free(p);                    // Free memory.  Obsolete: use delete
exit(n);                    // Kill program, return status n
system(s);                  // Execute OS command s (system dependent)
getenv("PATH");             // Environment variable or 0 (system dependent)
abs(n); labs(ln);           // Absolute value as int, long
```

STRING.H, CSTRING (Character array handling functions)

Strings are type char[] with a '\0' in the last element used.

```
strcpy(dst, src);           // Copy string. Not bounds checked
strcat(dst, src);           // Concatenate to dst. Not bounds checked
strcmp(s1, s2);             // Compare, <0 if s1<s2, 0 if s1==s2, >0 if
s1>s2
strncpy(dst, src, n);       // Copy up to n chars, also strncat(), strncmp()
strlen(s);                  // Length of s not counting \0
strchr(s,c); strrchr(s,c);// Address of first/last char c in s or 0
strstr(s, sub);             // Address of first substring in s or 0
  // mem... functions are for any pointer types (void*), length n bytes
memmove(dst, src, n);       // Copy n bytes from src to dst
memcmp(s1, s2, n);          // Compare n bytes as in strcmp
memchr(s, c, n);            // Find first byte c in s, return address or 0
memset(s, c, n);            // Set n bytes of s to c
```

CTYPE.H, CCTYPE (Character types)

```
isalnum(c);                 // Is c a letter or digit?
isalpha(c); isdigit(c);     // Is c a letter?  Digit?
islower(c); isupper(c);     // Is c lower case?  Upper case?
tolower(c); toupper(c);     // Convert c to lower/upper case
```

MATH.H, CMATH (Floating point math)

```
sin(x); cos(x); tan(x);     // Trig functions, x (double) is in radians
```

```
asin(x); acos(x); atan(x);// Inverses
atan2(y, x);              // atan(y/x)
sinh(x); cosh(x); tanh(x);// Hyperbolic
exp(x); log(x); log10(x); // e to the x, log base e, log base 10
pow(x, y); sqrt(x);       // x to the y, square root
ceil(x); floor(x);        // Round up or down (as a double)
fabs(x); fmod(x, y);      // Absolute value, x mod y
```

TIME.H, CTIME (Clock)

```
clock()/CLOCKS_PER_SEC;   // Time in seconds since program started
time_t t=time(0);         // Absolute time in seconds or -1 if unknown
tm* p=gmtime(&t);         // 0 if UCT unavailable, else p->tm_X where X
is:
  sec, min, hour, mday, mon (0-11), year (-1900), wday, yday, isdst
asctime(p);               // "Day Mon dd hh:mm:ss yyyy\n"
asctime(localtime(&t));   // Same format, local time
```

ASSERT.H, CASSERT (Debugging aid)

```
assert(e);                // If e is false, print message and abort
#define NDEBUG            // (before #include <assert.h>), turn off assert
```

NEW.H, NEW (Out of memory handler)

```
set_new_handler(handler); // Change behavior when out of memory
void handler(void) {throw bad_alloc();}  // Default
```

IOSTREAM.H, IOSTREAM (Replaces stdio.h)

```
cin >> x >> y;              // Read words x and y (any type) from stdin
cout << "x=" << 3 << endl;  // Write line to stdout
cerr << x << y << flush;    // Write to stderr and flush
c = cin.get();              // c = getchar();
cin.get(c);                 // Read char
cin.getline(s, n, '\n');    // Read line into char s[n] to '\n' (default)
if (cin)                    // Good state (not EOF)?
                            // To read/write any type T:
istream& operator>>(istream& i, T& x) {i >> ...; x=...; return i;}
ostream& operator<<(ostream& o, const T& x) {return o << ...;}
```

FSTREAM.H, FSTREAM (File I/O works like cin, cout as above)

```
ifstream f1("filename");  // Open text file for reading
if (f1)                   // Test if open and input available
  f1 >> x;                // Read object from file
f1.get(s);                // Read char or line
f1.getline(s, n);         // Read line into string s[n]
ofstream f2("filename");  // Open file for writing
if (f2) f2 << x;          // Write to file
```

IOMANIP.H, IOMANIP (Output formatting)

```
cout << setw(6) << setprecision(2) << setfill('0') << 3.1; // print
"003.10"
```

STRING (Variable sized character array)

```
string s1, s2="hello";    // Create strings
s1.size(), s2.size();     // Number of characters: 0, 5
s1 += s2 + ' ' + "world"; // Concatenation
s1 == "hello world"       // Comparison, also <, >, !=, etc.
s1[0];                    // 'h'
s1.substr(m, n);          // Substring of size n starting at s1[m]
s1.c_str();               // Convert to const char*
getline(cin, s);          // Read line ending in '\n'
```

VECTOR (Variable sized array/stack with built in memory allocation)

```
vector<int> a(10);        // a[0]..a[9] are int (default size is 0)
a.size();                 // Number of elements (10)
a.push_back(3);           // Increase size to 11, a[10]=3
a.back()=4;               // a[10]=4;
a.pop_back();             // Decrease size by 1
a.front();                // a[0];
a[20]=1;                  // Crash: not bounds checked
a.at(20)=1;               // Like a[20] but throws out_of_range()
for (vector<int>::iterator p=a.begin(); p!=a.end(); ++p)
  *p=0;                   // Set all elements of a to 0
vector<int> b(a.begin(), a.end());  // b is copy of a
vector<T> c(n, x);        // c[0]..c[n-1] init to x
T d[10]; vector<T> e(d, d+10);      // e is initialized from d
```

DEQUE (array/stack/queue)

deque<T> is like vector<T>, but also supports:

```
a.push_front(x);          // Puts x at a[0], shifts elements toward back
a.pop_front();            // Removes a[0], shifts toward front
```

UTILITY (Pair)

```
pair<string, int> a("hello", 3);  // A 2-element struct
a.first;                  // "hello"
a.second;                 // 3
```

MAP (associative array)

```
map<string, int> a;       // Map from string to int
a["hello"]=3;             // Add or replace element a["hello"]
for (map<string, int>::iterator p=a.begin(); p!=a.end(); ++p)
  cout << (*p).first << (*p).second;  // Prints hello, 3
a.size();                 // 1
```

ALGORITHM (A collection of 60 algorithms on sequences with iterators)

```
min(x, y); max(x, y);     // Smaller/larger of x, y (any type defining <)
swap(x, y);               // Exchange values of variables x and y
sort(a, a+n);             // Sort array a[0]..a[n-1] by <
sort(a.begin(), a.end()); // Sort vector or deque
```