Standard Code Libraries

<cmath>

- double sqrt(double x)
- double pow(double x, double y) x^y:. If x > 0, y can be any value.
  If x is 0, y must be > 0.
  If x < 0, y must be an integer
- double sin(double x)
- double cos(double x)
- double tan(double x)
- double exp(double x)
- double log(double x)
- double ceil(double x) Smallest integer ≥ x
- double floor(double x) Largest integer ≤ x
- double fabs(double x) Absolute value, |x|

<cstdlib>

- int abs(int x) Absolute value, |x|
- int rand() Random integer
- void srand(int n) Sets the seed of the random number generator to n.
- void exit(int n) Exits the program with status code n.

<cctype>

- bool isalpha(char c) Tests whether c is a letter.
- bool isdigit(char c) Tests whether c is a digit.
- bool isspace(char c) Tests whether c is white space.
- bool islower(char c) Tests whether c is lowercase.
- bool isupper(char c) Tests whether c is uppercase.
- char tolower(char c) Returns the lowercase of c.
- char toupper(char c) Returns the uppercase of c.

<string>

- istream& getline(istream& in, string s) Gets the next input line from the input stream in and stores it in the string s.
- int string.length() const The length of the string.
- string.substr(int i, int n) const The substring of length n starting at index i.
- string.substr(int i) const The substring from index i to the end of the string.
- const char* string::c_str() const A char array with the characters in this string.

<iostream>

Class istream

- bool istream::fail() const True if input has failed.
- istream& istream::get(char& c) Gets the next character and places it into c.
- istream& istream::unget() Puts the last character read back into the stream, to be read again in the next input operation; only one character can be put back at a time.

<iomanip>

- setw(int n) Sets the width of the next field.
- setprecision(int n) Sets the precision of floating-point values to n digits after the decimal point.
- fixed Selects fixed floating-point format, with trailing zeroes.
- scientific Selects scientific floating-point format, with exponential notation.
- setfill(char c) Sets the fill character to the character c.
- setbase(int n) Sets the number base for integers to base n.
- hex Sets hexadecimal integer format.
- oct Sets octal integer format.
- dec Sets decimal integer format.

<fstream>

Class ifstream

- void ifstream::open(const char n[]) Opens a file with name n for reading.

Class ofstream

- void ofstream::open(const char n[]) Opens a file with name n for writing.

Class fstream

- void fstream::open(const char n[]) Opens a file with name n for reading and writing.

Class fstreambase

- void fstreambase::close() Closes the file stream.

Note:

- fstreambase is the common base class of ifstream, ofstream, and fstream.
- To open a binary file both for input and output, use f.open(n, ios::in | ios::out ios::binary)

<sstream>

Class istringstream

- istringstream::istringstream(string s) Constructs a string stream that reads from the string s.

Class ostringstream

- string ostringstream::str() const Returns the string that was collected by the string stream.

Note:

- Call istringstream(s.c_str()) to construct an istringstream.
- Call s = string(out.str()) to get a string object that contains the characters collected by the ostringstream out.

All STL Containers, C

Note: • C is any STL container such as vector<T>, list<T>, set<T>, multiset<T>, or map<T>.

• int C::size() const The number of elements in the container.
• C::iterator C::begin() Gets an iterator that points to the first element in the container.
• C::iterator C::end() Gets an iterator that points past the last element in the container.
• bool C::empty() const Tests if the container has any elements.

<vector>

Class vector<T>

• vector<T>::vector(int n) Constructs a vector with n elements.
• void vector<T>::push_back(const T& x) Inserts x after the last element.
• void vector<T>::pop_back() Removes (but does not return) the last element.
• T& vector<T>::operator[](int n) Accesses the element at index n.
• T& vector<T>::at(int n)
Accesses the element at index n, checking that the index is in range.
• vector<T>::iterator vector<T>::insert(vector<T>::iterator p, const T& x)
Inserts x before p. Returns an iterator that points to the inserted value.
• vector<T>::iterator vector<T>::erase(vector<T>::iterator p)
Erases the element to which p points. Returns an iterator that points to the next element.
• vector<T>::iterator vector<T>::erase(vector<T>::iterator begin, vector<T>::iterator end)
Erases all the elements between the start and the stop iterator. Returns an iterator that points to the next element.

<deque>

Class deque<T>

• void deque<T>::push_back(const T& x) Inserts x after the last element.
• void deque<T>::pop_back() Removes (but does not return) the last element.
• void deque<T>::push_front(const T& x) Inserts x before the first element.
• void deque<T>::pop_front() Removes (but does not return) the first element.
• T& deque<T>::front() The first element of the container.
• T& deque<T>::back() The last element of the container.
• T& deque<T>::operator[](int n) Access the element at index n.
• T& deque<T>::at(int n) Access the element at index n, checking index.
• deque<T>::iterator deque<T>::erase(deque<T>::iterator p)
Erases the element to which p points. Returns an iterator that points to the next element.
• deque<T>::iterator deque<T>::erase(deque<T>::iterator begin, deque<T>::iterator end)
Erases all the elements between the start and the stop iterator. Returns an iterator that points to the next element.

<list>

Class list<T>

• void list<T>::push_back(const T& x) Inserts x after the last element.
• void list<T>::pop_back() Removes (but does not return) the last element.
• void list<T>::push_front(const T& x) Inserts x before the first element.
• void list<T>::pop_front() Removes (but does not return) the first element.
• T& list<T>::front() The first element of the container.
• T& list<T>::back() The last element of the container.
• list<T>::iterator list<T>::insert(list<T>::iterator p, const T& x) Inserts x before p. Returns an iterator that points to the inserted value.
• list<T>::iterator list<T>::erase(list<T>::iterator p)
Erases the element to which p points. Returns an iterator that points to the next element.
• list<T>::iterator list<T>::erase(list<T>::iterator begin, list<T>::iterator end)
Erases all the elements between the start and the stop iterator. Returns an iterator that points to the next element.
• void sort() Sorts the list into ascending order.
• void merge(list<T>& x) Merges elements with the sorted list x.

<set>

Class set<T>

• pair< set<T>::iterator, bool > set<T>::insert(const T& x)
If x is not present in the list, inserts it and returns an iterator that points to the newly inserted element and the Boolean value true. If x is present, returns an iterator pointing to the existing set element and the Boolean value false.
• int set<T>::erase(const T& x)
Removes x and returns 1 if it occurs in the set; returns 0 otherwise.
• void set<T>::erase(set<T>::iterator p)
Erases the element at the given position.
• int set<T>::count(const T& x) const
Returns 1 if x occurs in the set; returns 0 otherwise.
• set<T>::iterator set<T>::find(const T& x)
Returns an iterator to the element equal to x in the set, or end() if no such element exists.

Note: • The type T must be totally ordered by a < comparison operator.

<multiset>

Class multiset<T>

• multiset<T>::iterator multiset<T>::insert(const T& x)
Inserts x into the container. Returns an iterator that points to the inserted value.
• int multiset<T>::erase(const T& x)
Removes all occurrences of x. Returns the number of removed elements.
• void multiset<T>::erase(multiset<T>::iterator p)
Erases the element at the given position.
• int multiset<T>::count(const T& x) const
Counts the elements equal to x.
• multiset<T>::iterator multiset<T>::find(const T& x)
Returns an iterator to an element equal to x, or end() if no such element exists.

Note: • The type T must be totally ordered by a < comparison operator.

<map>

Class map<K, V>

- V& map<K, V>::operator[](const K& k) Accesses the value with key k.
- int map<K, V>::erase(const K& k) Removes all occurrences of elements with key k. Returns the number of removed elements.
- void map<K, V>::erase(map<K, V>::iterator p) Erases the element at the given position.
- int map<K, V>::count(const K& k) const Counts the elements with key k.
- map<K, V>::iterator map<K, V>::find(const K& k) Returns an iterator to an element with key k, or end() if no such element exists.

Note: • The key type K must be totally ordered by a < comparison operator.
• A map iterator points to pair<K, V> entries.

Class multimap<K, V>

- multimap<K, V>::iterator multimap<K, V>::insert(const pair<K, V>& kvpair)

Inserts a key/value pair and returns an iterator pointing to the inserted pair.

- void multimap<K, V>::erase(multimap<K, V>::iterator pos)

Erases the key/value pair at the position pos.

- multimap<K, V>::iterator multimap<K, V>::lower-bound(const K& k)
- multimap<K, V>::iterator multimap<K, V>::upper-bound(const K& k)

Returns the position of the first and after the last key/value pair with key k.

<stack>

Class stack<T>

- T& stack<T>::top() The value at the top of the stack.
- void stack<T>::push(const T& x) Adds x to the top of the stack.
- void stack<T>::pop() Removes (but does not return) the top value of the stack.

<queue>

Class queue<T>

- T& queue<T>::front() The value at the front of the queue.
- T& queue<T>::back() The value at the back of the queue.
- void queue<T>::push(const T& x) Adds x to the back of the queue.
- void queue<T>::pop() Removes (but does not return) the front value of the queue.
- T& priority_queue<T>::top() The largest value in the container.
- void priority_queue<T>::push(const T& x) Adds x to the container.
- void priority_queue<T>::pop() Removes (but does not return) the largest value in the container.

<utility>

Class pair

- pair<F, S>::pair(const F& f, const F& s)

Constructs a pair from a first and second value.

- F pair<F, S>::first

The public field holding the first value of the pair.

- S pair<F, S>::second

The public field holding the second value of the pair.

Algorithms

<algorithm>

- T min(T x, T y) The minimum of x and y.
- T max(T x, T y) The maximum of x and y.
- void swap(T& a, T& b) Swaps the contents of a and b.
- I min_element(I begin, I end) Returns an iterator pointing to the minimum element in the iterator range [begin, end).
- I max_element(I begin, I end) Returns an iterator pointing to the maximum element in the iterator range [begin, end).
- F for_each(I begin, I end, F f) Applies the function f to all elements in the iterator range [begin, end). Returns f.
- I find(I begin, I end, T x) Returns the iterator pointing to the first occurrence of x in the iterator range [begin, end), or end if there is no match.
- I find_if(I begin, I end, F f) Returns the iterator pointing to the first element x in the iterator range [begin, end) for which f(x) is true, or end if there is no match.
- int count(I begin, I end, T x) Counts how many values in the iterator range [begin, end) are equal to x.
- int count_if(I begin, I end, F f) Counts for how many values x in the iterator range [begin, end) f(x) is true.
- bool equal(I1 begin1, I1 end1, I2 begin2)

Tests whether the range [begin1, end1) equals the range of the same size starting at begin2.

- I2 copy(I1 begin1, I1 end1, I2 begin2)

Copies the range [begin1, end1) to the range of the same size starting at begin2. Returns the iterator past the end of the destination of the copy.

- void replace(I begin, I end, T xold, T xnew)

Replaces all occurrences of xold in the range [begin, end) with xnew.

- void replace_if(I begin, I end, F f, T xnew)

Replaces all values x in the range [begin, end) for which f(x) is true with xnew.

- void fill(I begin, I end, T x) Fills the range [begin, end) with x.
- void fill(I begin, int n, T x) Fills n copies of x into the range that starts at begin.
- I remove(I begin, I end, T x) Removes all occurrences of x in the range [begin, end). Returns the end of the resulting range.
- I remove_if(I begin, I end, F f) Removes all values x in the range [begin, end) for which f(x) is true. Returns the end of the resulting range.
- I unique(I begin, I end) Removes adjacent identical values from the range [begin, end). Returns the end of the resulting range.
- void random_shuffle(I begin, I end) Randomly rearranges the elements in the range [begin, end).
- void next_permutation(I begin, I end) Rearranges the elements in the range [begin, end). Calling it n! times iterates through all permutations.
- void sort(I begin, I end) Sorts the elements in the range [begin, end).
- I nth_element(I begin, I end, int n) Returns an iterator that points to the value that would be the nth element if the range [begin, end) was sorted.
- bool binary_search(I begin, I end, T x) Checks whether the value x is contained in the sorted range [begin, end).

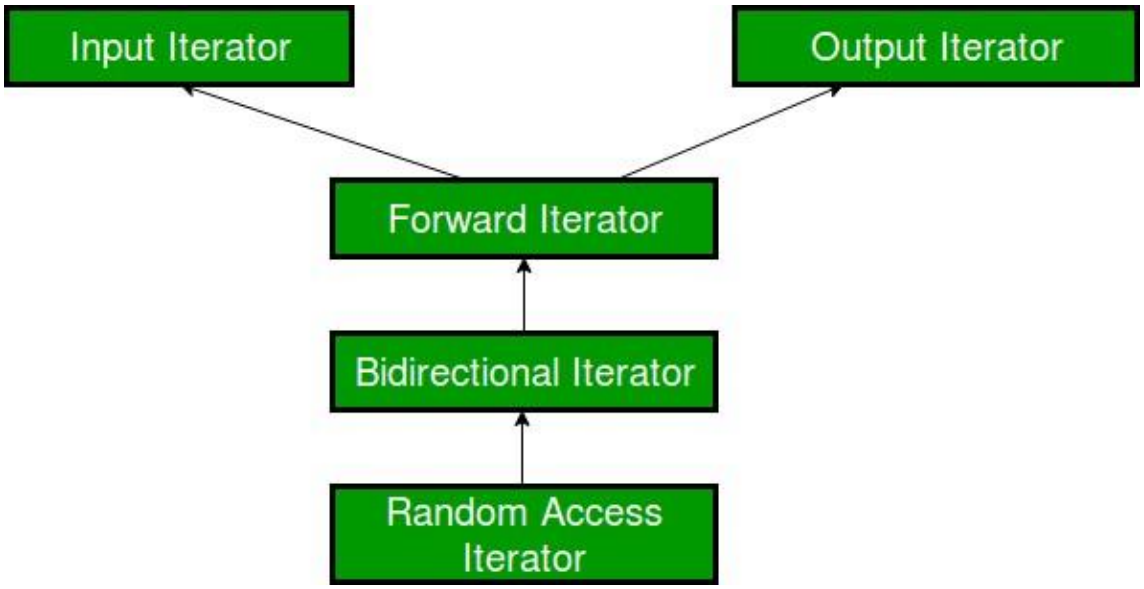
Exceptions

<stdexcept>

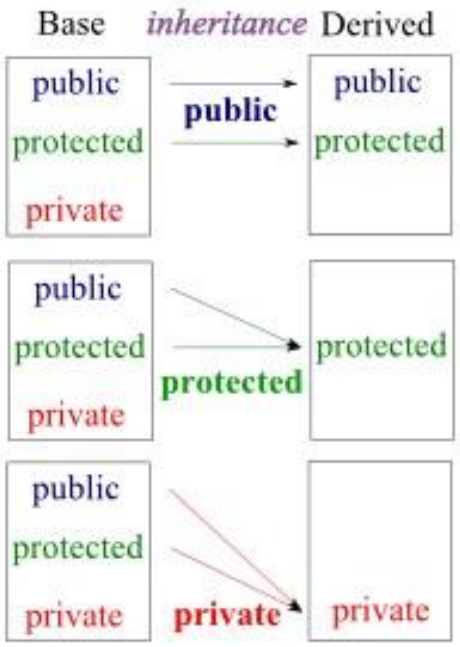
Class exception Base class for all standard exceptions.
Class logic_error An error that logically results from conditions in the program.
Class domain_error A value is not in the domain of a function.
Class invalid_argument A parameter value is invalid.
Class out_of_range A value is outside the valid range.
Class length_error A value exceeds the maximum length.
Class runtime_error An error that occurs as a consequence of conditions beyond the control of the program.
Class range_error An operation computes a value that is outside the range of a function.
Class overflow_error An operation yields an arithmetic overflow.
Class underflow_error An operation yields an arithmetic underflow.

Note: • All standard exception classes have a constructor: ExceptionClass::ExceptionClass(string reason)

• The exception class has a member function to retrieve



NEW	MALLOC
calls constructor	doesnot calls constructors
It is an operator	It is a function
Returns exact data type	Returns void *
on failure, Throws	On failure, returns NULL
Memory allocated from free store	Memory allocated from heap
can be overridden	cannot be overridden
size is calculated by compiler	size is calculated manually



String

Member functions

(constructor)	Construct string object (public member function)
(destructor)	String destructor (public member function)
operator=	String assignment (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

size	Return length of string (public member function)
length	Return length of string (public member function)
max_size	Return maximum size of string (public member function)
resize	Resize string (public member function)
capacity	Return size of allocated storage (public member function)
reserve	Request a change in capacity (public member function)
clear	Clear string (public member function)
empty	Test if string is empty (public member function)
shrink_to_fit C++11	Shrink to fit (public member function)

Element access:

operator[]	Get character of string (public member function)
at	Get character in string (public member function)
back C++11	Access last character (public member function)
front C++11	Access first character (public member function)

Modifiers:

operator+=	Append to string (public member function)
append	Append to string (public member function)
push_back	Append character to string (public member function)
assign	Assign content to string (public member function)
insert	Insert into string (public member function)
erase	Erase characters from string (public member function)
replace	Replace portion of string (public member function)
swap	Swap string values (public member function)
pop_back C++11	Delete last character (public member function)

String operations:

c_str	Get C string equivalent (public member function)
data	Get string data (public member function)
get_allocator	Get allocator (public member function)
copy	Copy sequence of characters from string (public member function)
find	Find content in string (public member function)
rfind	Find last occurrence of content in string (public member function)
find_first_of	Find character in string (public member function)
find_last_of	Find character in string from the end (public member function)
find_first_not_of	Find absence of character in string (public member function)
find_last_not_of	Find non-matching character in string from the end (public member function)
substr	Generate substring (public member function)
compare	Compare strings (public member function)

Member constants

npos	Maximum value for size_t (public static member constant)

Non-member function overloads

operator+	Concatenate strings (function)
relational operators	Relational operators for string (function)
swap	Exchanges the values of two strings (function)
operator>>	Extract string from stream (function)
operator<<	Insert string into stream (function)
getline	Get line from stream into string (function)

Vector

Member functions

(constructor)	Construct vector (public member function)
(destructor)	Vector destructor (public member function)
operator=	Assign content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

size	Return size (public member function)
max_size	Return maximum size (public member function)
resize	Change size (public member function)
capacity	Return size of allocated storage capacity (public member function)
empty	Test whether vector is empty (public member function)
reserve	Request a change in capacity (public member function)
shrink_to_fit C++11	Shrink to fit (public member function)

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data C++11	Access data (public member function)

Modifiers:

assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)

Allocator:

get_allocator	Get allocator (public member function)

Non-member function overloads

relational operators	Relational operators for vector (function template)
swap	Exchange contents of vectors (function template)

Template specializations

vector<bool>	Vector of bool (class template specialization)

Fstream

Public member functions

(constructor)	Construct object and optionally open file (public member function)
open	Open file (public member function)
is_open	Check if a file is open (public member function)
close	Close file (public member function)
rdbuf	Get the associated filebuf object (public member function)
operator= C++11	Move assignment (public member function)
swap C++11	Swap internals (public member function)

Public member functions inherited from `istream`

operator>>	Extract formatted input (public member function)
gcount	Get character count (public member function)
get	Get characters (public member function)
getline	Get line (public member function)
ignore	Extract and discard characters (public member function)
peek	Peek next character (public member function)
read	Read block of data (public member function)
readsome	Read data available in buffer (public member function)
putback	Put character back (public member function)
unget	Unget character (public member function)
tellg	Get position in input sequence (public member function)
seekg	Set position in input sequence (public member function)
sync	Synchronize input buffer (public member function)

Public member functions inherited from `ostream`

operator<<	Insert formatted output (public member function)
put	Put character (public member function)
write	Write block of data (public member function)
tellp	Get position in output sequence (public member function)
seekp	Set position in output sequence (public member function)
flush	Flush output stream buffer (public member function)

Public member functions inherited from `ios`

good	Check whether state of stream is good (public member function)
eof	Check whether eofbit is set (public member function)
fail	Check whether either failbit or badbit is set (public member function)
bad	Check whether badbit is set (public member function)
operator!	Evaluate stream (not) (public member function)
operator bool C++11	Evaluate stream (public member function)
rdstate	Get error state flags (public member function)
setstate	Set error state flag (public member function)
clear	Set error state flags (public member function)
copyfmt	Copy formatting information (public member function)
fill	Get/set fill character (public member function)
exceptions	Get/set exceptions mask (public member function)
imbue	Imbue locale (public member function)
tie	Get/set tied stream (public member function)
rdbuf	Get/set stream buffer (public member function)
narrow	Narrow character (public member function)
widen	Widen character (public member function)

Public member functions inherited from `ios_base`

flags	Get/set format flags (public member function)
setf	Set specific format flags (public member function)
unsetf	Clear specific format flags (public member function)
precision	Get/Set floating-point decimal precision (public member function)
width	Get/set field width (public member function)
imbue	Imbue locale (public member function)
getloc	Get current locale (public member function)
xalloc	Get new index for extensible array [static] (public static member function)
iword	Get integer element of extensible array (public member function)
pword	Get pointer element of extensible array (public member function)
register_callback	Register event callback function (public member function)
sync_with_stdio	Toggle synchronization with cstdio streams [static] (public static member function)

Non-member function overloads

swap C++11	Swap file streams (function template)

C++ QUICK REFERENCE

PREPROCESSOR

```
// Comment to end of line
/* Multi-line comment */
#include <stdio.h> // Insert standard header file
#include "myfile.h" // Insert file in current directory
#define X some text // Replace X with some text
#define F(a,b) a+b // Replace F(1,2) with 1+2
#define X \
some text // Line continuation
#undef X // Remove definition
#if defined(X) // Condional compilation (#ifdef X)
#else // Optional (#ifndef X or #if !defined(X))
#endif // Required after #if, #ifdef
```

LITERALS

```
255, 0377, 0xff // Integers (decimal, octal, hex)
2147483647L, 0x7fffffffl // Long (32-bit) integers
123.0, 1.23e2 // double (real) numbers
'a', '\141', '\x61' // Character (literal, octal, hex)
'\n', '\\', '\'', '\"' // Newline, backslash, single quote, double quote
"string\n" // Array of characters ending with newline and \0
"hello" "world" // Concatenated strings
true, false // bool constants 1 and 0
```

DECLARATIONS

```
int x; // Declare x to be an integer (value undefined)
int x=255; // Declare and initialize x to 255
short s; long l; // Usually 16 or 32 bit integer (int may be either)
char c='a'; // Usually 8 bit character
unsigned char u=255; signed char s=-1; // char might be either
unsigned long x=0xffffffffL; // short, int, long are signed
float f; double d; // Single or double precision real (never unsigned)
bool b=true; // true or false, may also use int (1 or 0)
int a, b, c; // Multiple declarations
int a[10]; // Array of 10 ints (a[0] through a[9])
int a[]={0,1,2}; // Initialized array (or a[3]={0,1,2}; )
int a[2][3]={{1,2,3},{4,5,6}}; // Array of array of ints
char s[]="hello"; // String (6 elements including '\0')
int* p; // p is a pointer to (address of) int
char* s="hello"; // s points to unnamed array containing "hello"
void* p=NULL; // Address of untyped memory (NULL is 0)
int& r=x; // r is a reference to (alias of) int x
enum weekend {SAT,SUN}; // weekend is a type with values SAT and SUN
enum weekend day; // day is a variable of type weekend
enum weekend {SAT=0,SUN=1}; // Explicit representation as int
enum {SAT,SUN} day; // Anonymous enum
typedef String char*; // String s; means char* s;
const int c=3; // Constants must be initialized, cannot assign to
const int* p=a; // Contents of p (elements of a) are constant
const p=a; // p (but not contents) are constant
const int* const p=a; // Both p and its contents are constant
const int& cr=x; // cr cannot be assigned to change x
```

STORAGE CLASSES

```
int x; // Auto (memory exists only while in scope)
static int x; // Global lifetime even if local scope
extern int x; // Information only, declared elsewhere
```

STATEMENTS

```
x=y; // Every expression is a statement
int x; // Declarations are statements
; // Empty statement
{ // A block is a single statement
int x; // Scope of x is from declaration to end of block a; // In C,
declarati ons must precede statements
}
if (x) a; // If x is true (not 0), evaluate a
else if (y) b; // If not x and y (optional, may be repeated)
else c; // If not x and not y (optional)
while (x) a; // Repeat 0 or more times while x is true
for (x; y; z) a; // Equivalent to: x; while(y) {a; z;}
do a; while (x); // Equivalent to: a; while(x) a;
switch (x) { // x must be int
case X1: a; // If x == X1 (must be a const), jump here
case X2: b; // Else if x == X2, jump here
default: c; // Else jump here (optional)
}
break; // Jump out of while, do, or for loop, or switch
continue; // Jump to bottom of while, do, or for loop
return x; // Return x from function to caller
try { a; }
catch (T t) { b; } // If a throws a T, then jump here
catch (...) { c; } // If a throws something else, jump here
```

FUNCTIONS

```
int f(int x, int); // f is a function taking 2 ints and returning
int
void f(); // f is a procedure taking no arguments
void f(int a=0); // f() is equivalent to f(0)
f(); // Default return type is int
inline f(); // Optimize for speed
f() { statements; } // Function definition (must be global)
T operator+(T x, T y); // a+b (if type T) calls operator+(a, b)
T operator-(T x); // -a calls function operator-(a)
T operator++(int); // postfix ++ or -- (parameter ignored)
extern "C" {void f();} // f() was compiled in C
Function parameters and return values may be of any type. A function must
eith er be declared or defined before
it is used. It may be declared first and defined later. Every program
consists  of a set of a set of global variable
declarations and a set of function definitions (possibly in separate files),
o ne of which must be:
int main() { statements... } or
```

```
int main(int argc, char* argv[]) { statements... }
argv is an array of argc strings from the command line. By convention, main re turns status 0 if successful, 1 or
higher for errors.
Functions with different parameters may have the same name (overloading). Oper ators except :: . .* ?: may be overloaded. Precedence order is not affected.
New operators may not be created.
```

EXPRESSIONS

```
Operators are grouped by precedence, highest first. Unary operators and assign ment evaluate right to left. All
others are left to right. Precedence does not affect order of evaluation, whic h is undefined. There are no run time
checks for arrays out of bounds, invalid pointers, etc.
T::X // Name X defined in class T
N::X // Name X defined in namespace N
::X // Global name X
t.x // Member x of struct or class t
p->x // Member x of struct or class pointed to by p
a[i] // i'th element of array a
f(x,y) // Call to function f with arguments x and y
T(x,y) // Object of class T initialized with x and y
x++ // Add 1 to x, evaluates to original x (postfix)
x-- // Subtract 1 from x, evaluates to original x
typeid(x) // Type of x
typeid(T) // Equals typeid(x) if x is a T
dynamic_cast<T>(x) // Converts x to a T, checked at run time
static_cast<T>(x) // Converts x to a T, not checked
reinterpret_cast<T>(x) // Interpret bits of x as a T
const_cast<T>(x) // Converts x to same type T but not const
sizeof x // Number of bytes used to represent object x
sizeof(T) // Number of bytes to represent type T
++x // Add 1 to x, evaluates to new value (prefix)
--x // Subtract 1 from x, evaluates to new value
~x // Bitwise complement of x
!x // true if x is 0, else false (1 or 0 in C)
-x // Unary minus
+x // Unary plus (default)
&x // Address of x
*p // Contents of address p (*&x equals x)
new T // Address of newly allocated T object
new T(x, y) // Address of a T initialized with x, y
new T[x] // Address of allocated n-element array of T
delete p // Destroy and free object at address p
delete[] p // Destroy and free array of objects at p
(T) x // Convert x to T (obsolete, use ..._cast<T>(x))
x * y // Multiply
x / y // Divide (integers round toward 0)
x % y // Modulo (result has sign of x)
x + y // Add, or &x[y]
x - y // Subtract, or number of elements from *x to *y
x << y // x shifted y bits to left (x * pow(2, y))
x >> y // x shifted y bits to right (x / pow(2, y))
x < y // Less than
x <= y // Less than or equal to
x > y // Greater than
x >= y // Greater than or equal to
x == y // Equals
x != y // Not equals
x & y // Bitwise and (3 & 6 is 2)
x ^ y // Bitwise exclusive or (3 ^ 6 is 5)
x | y // Bitwise or (3 | 6 is 7)
x && y // x and then y (evaluates y only if x (not 0))
x || y // x or else y (evaluates y only if x is false
(0))
x = y // Assign y to x, returns new value of x
x += y // x = x + y, also -= *= /= <<= >>= &= |= ^=
x ? y : z // y if x is true (nonzero), else z
throw x // Throw exception, aborts if not caught
x , y // evaluates x and y, returns y (seldom used)
```

CLASSES

```
class T { // A new type
private: // Section accessible only to T's member
functions
protected: // Also accessable to classes derived from T
public: // Accessable to all
int x; // Member data
void f(); // Member function
void g() {return;} // Inline member function
void h() const; // Does not modify any data members
int operator+(int y); // t+y means t.operator+(y)
int operator-(); // -t means t.operator-()
T(): x(1) {} // Constructor with initialization list
T(const T& t): x(t.x) {} // Copy constructor
T& operator=(const T& t) {x=t.x; return *this; } // Assignment operator
~T(); // Destructor (automatic cleanup routine)
explicit T(int a); // Allow t=T(3) but not t=3
operator int() const {return x;} // Allows int(t)
friend void i(); // Global function i() has private access
friend class U; // Members of class U have private access
static int y; // Data shared by all T objects
static void l(); // Shared code. May access y but not x
class Z {}; // Nested class T::Z
typedef int V; // T::V means int
};
void T::f() { // Code for member function f of class T
this->x = x;} // this is address of self (means x=x;)
int T::y = 2; // Initialization of static member (required)
T::l(); // Call to static member
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 destruct or, which perform the
```

```
corresponding operations on each data member and each base class as shown above. There is also a default noargument
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
// Make T visible without N::
using namespace 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
```