Standard Code Libraries

<cmath>

• double sqrt(double x)
Function: Square root, sqrt(x)

• double pow(double x, double y)
Function: Power, 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)
Function: Sine, sin x (x in radians)

• double cos(double x)
Function: Cosine, cos x (x in radians)

• double tan(double x)
Function: Tangent, tan x (x in radians)

• double exp(double x)
Function: Exponential, e^x

• double log(double x)
Function: Natural log, ln (x), x > 0

• double ceil(double x)
Function: Smallest integer ≥ x

• double floor(double x)
Function: Largest integer ≤ x

• double fabs(double x)
Function: Absolute value, |x|

<cstdlib>

• int abs(int x)
Function: Absolute value, |x|

• int rand()
Function: Random integer

• void srand(int n)
Function: Sets the seed of the random number generator to n.

• void exit(int n)
Function: Exits the program with status code n.

<cctype>

• bool isalpha(char c)
Function: Tests whether c is a letter.

• bool isdigit(char c)
Function: Tests whether c is a digit.

• bool isspace(char c)
Function: Tests whether c is white space.

• bool islower(char c)
Function: Tests whether c is lowercase.

• bool isupper(char c)
Function: Tests whether c is uppercase.

• char tolower(char c)
Function: Returns the lowercase of c.

• char toupper(char c)
Function: Returns the uppercase of c.

<string>

• istream& getline(istream& in, string s)
Function: Gets the next input line from the input stream in and stores it in the string s.

• int string::length() const
Member function: The length of the string.

• string string::substr(int i, int n) const
Member function: The substring of length n starting at index i.

• string string::substr(int i) const
Member function: The substring from index i to the end of the string.

• const char* string::c_str() const
Member function: A char array with the characters in this string.

<iostream>

Class istream

• bool istream::fail() const
Function: True if input has failed.

• istream& istream::get(char& c)
Function: Gets the next character and places it into c.

• istream& istream::unget()
Function: 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)
Manipulator: Sets the width of the next field.

• setprecision(int n)
Manipulator: Sets the precision of floating-point values to n digits after the decimal point.

• fixed
Manipulator: Selects fixed floating-point format, with trailing zeroes.

• scientific
Manipulator: Selects scientific floating-point format, with exponential notation.

• setfill(char c)
Manipulator: Sets the fill character to the character c.

• setbase(int n)
Manipulator: Sets the number base for integers to base n.

• hex
Manipulator: Sets hexadecimal integer format.

• oct
Manipulator: Sets octal integer format.

• dec
Manipulator: Sets decimal integer format.

<fstream>

Class ifstream

• void ifstream::open(const char n[])

Function: Opens a file with name n for reading.

Class ofstream

• void ofstream::open(const char n[])

Function: Opens a file with name n for writing.

Class fstream

• void fstream::open(const char n[])

Function: Opens a file with name n for reading and writing.

Class fstreambase

• void fstreambase::close()

Function: 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

Function: 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
Function: The number of elements in the container.

• C::iterator C::begin()
Function: Gets an iterator that points to the first element in the container.

• C::iterator C::end()
Function: Gets an iterator that points past the last element in the container.

• bool C::empty() const
Function: Tests if the container has any elements.

<vector>

Class vector<T>

• vector<T>::vector(int n)
Function: Constructs a vector with n elements.

• void vector<T>::push_back(const T& x)
Function: Inserts x after the last element.

• void vector<T>::pop_back()
Function: Removes (but does not return) the last element.

• T& vector<T>::operator[](int n)
Function: Accesses the element at index n.

• T& vector<T>::at(int n)
Function: 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)
Function: Inserts x before p. Returns an iterator that points to the inserted value.

• vector<T>::iterator vector<T>::erase(vector<T>::iterator p)
Function: 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)
Function: 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)
Function: Inserts x after the last element.

• void deque<T>::pop_back()
Function: Removes (but does not return) the last element.

• void deque<T>::push_front(const T& x)
Function: Inserts x before the first element.

• void deque<T>::pop_front()
Function: Removes (but does not return) the first element.

• T& deque<T>::front()
Function: The first element of the container.

• T& deque<T>::back()
Function: The last element of the container.

• T& deque<T>::operator[](int n)
Function: Access the element at index n.

• T& deque<T>::at(int n)
Function: Access the element at index n, checking index.

• deque<T>::iterator deque<T>::erase(deque<T>::iterator p)
Function: 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)
Function: 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)
Function: Inserts x after the last element.

• void list<T>::pop_back()
Function: Removes (but does not return) the last element.

• void list<T>::push_front(const T& x)
Function: Inserts x before the first element.

• void list<T>::pop_front()
Function: Removes (but does not return) the first element.

• T& list<T>::front()
Function: The first element of the container.

• T& list<T>::back()
Function: The last element of the container.

• list<T>::iterator list<T>::insert(list<T>::iterator p, const T& x)
Function: Inserts x before p. Returns an iterator that points to the inserted value.

• list<T>::iterator list<T>::erase(list<T>::iterator p)
Function: 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)
Function: Erases all the elements between the start and the stop iterator. Returns an iterator that points to the next element.

• void sort()
Function: Sorts the list into ascending order.

• void merge(list<T>& x)
Function: Merges elements with the sorted list x.

<set>

Class set<T>

• pair< set<T>::iterator, bool > set<T>::insert(const T& x)
Function: 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)
Function: Removes x and returns 1 if it occurs in the set; returns 0 otherwise.

• void set<T>::erase(set<T>::iterator p)
Function: Erases the element at the given position.

• int set<T>::count(const T& x) const
Function: 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)
Function: Inserts x into the container. Returns an iterator that points to the inserted value.

• int multiset<T>::erase(const T& x)
Function: Removes all occurrences of x. Returns the number of removed elements.

• void multiset<T>::erase(multiset<T>::iterator p)
Function: Erases the element at the given position.

• int multiset<T>::count(const T& x) const
Function: Counts the elements equal to x.

• multiset<T>::iterator multiset<T>::find(const T& x)
Function: 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)
Function: Accesses the value with key k.

• int map<K, V>::erase(const K& k)
Function: Removes all occurrences of elements with key k. Returns the number of removed elements.

• void map<K, V>::erase(map<K, V>::iterator p)
Function: Erases the element at the given position.

• int map<K, V>::count(const K& k) const
Function: Counts the elements with key k.

• map<K, V>::iterator map<K, V>::find(const K& k)
Function: 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)
Function: Inserts a key/value pair and returns an iterator pointing to the inserted pair.

• void multimap<K, V>::erase(multimap<K, V>::iterator pos)
Function: 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)
Function: Returns the position of the first and after the last key/value pair with key k.

<stack>

Class stack<T>

• T& stack<T>::top()
Function: The value at the top of the stack.

• void stack<T>::push(const T& x)
Function: Adds x to the top of the stack.

• void stack<T>::pop()
Function: Removes (but does not return) the top value of the stack.

<queue>

Class queue<T>

• T& queue<T>::front()
Function: The value at the front of the queue.

• T& queue<T>::back()
Function: The value at the back of the queue.

• void queue<T>::push(const T& x)
Function: Adds x to the back of the queue.

• void queue<T>::pop()
Function: Removes (but does not return) the front value of the queue.

• T& priority_queue<T>::top()
Function: The largest value in the container.

• void priority_queue<T>::push(const T& x)
Function: Adds x to the container.

• void priority_queue<T>::pop()
Function: 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)
Function: The minimum of x and y.

• T max(T x, T y)
Function: The maximum of x and y.

• void swap(T& a, T& b)
Function: Swaps the contents of a and b.

• I min_element(I begin, I end)
Function: Returns an iterator pointing to the minimum element in the iterator range [begin, end).

• I max_element(I begin, I end)
Function: Returns an iterator pointing to the maximum element in the iterator range [begin, end).

• F for_each(I begin, I end, F f)
Function: Applies the function f to all elements in the iterator range [begin, end). Returns f.

• I find(I begin, I end, T x)
Function: 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)
Function: 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)
Function: Counts how many values in the iterator range [begin, end) are equal to x.

• int count_if(I begin, I end, F f)
Function: Counts for how many values x in the iterator range [begin, end) f(x) is true.

• bool equal(I1 begin1, I1 end1, I2 begin2)
Function: Tests whether the range [begin1, end1) equals the range of the same size starting at begin2.

• I2 copy(I1 begin1, I1 end1, I2 begin2)
Function: 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)
Function: Replaces all occurrences of xold in the range [begin, end) with xnew.

• void replace_if(I begin, I end, F f, T xnew)
Function: 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)
Function: Fills the range [begin, end) with x.

• void fill(I begin, int n, T x)
Function: Fills n copies of x into the range that starts at begin.

• I remove(I begin, I end, T x)
Function: 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)
Function: 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)
Function: Removes adjacent identical values from the range [begin, end). Returns the end of the resulting range.

• void random_shuffle(I begin, I end)
Function: Randomly rearranges the elements in the range [begin, end).

• void next_permutation(I begin, I end)
Function: Rearranges the elements in the range [begin, end). Calling it n! times iterates through all permutations.

• void sort(I begin, I end)
Function: Sorts the elements in the range [begin, end).

• I nth_element(I begin, I end, int n)
Function: 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)
Function: 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