Operaciones para el tipo de dato `string`

string s	Constructor por defeto
string s (“hola”)	Constructor con inicializador
string s (aString)	Constructor de copia
s[i]	Acceso al elemento i-ésimo del string
s.substr(int pos,int len)	Subcadena que comienza en pos y tiene longitud len
s.c_str()	Devuelve una cadena estilo C igual al string
s.insert(int pos,string str)	Insetar antes de pos el string str
s.erase (int start, int len)	Eliminar desde s[start] hasta s[start+len]
s.replace(int start, int len,str)	Sustituir desde s[start] hasta s[start+len] por str
s.length()	Longitud del string
s.resize(int,char)	Cambia el tamaño, rellenando con un valor
s.empty()	Cierto si el string es vacío
s = s2	Asignación de strings
s += s2	Concatenación de strings
s + s2	Nuevo string resultado de concatenar s y s2
s ==s2 s != s2	Igualdad y desigualdad de strings
s < s2 s <= s2	Comparaciones de strings (orden lexicográfico)
s > s2 s >= s2	Comparaciones de strings (orden lexicográfico)
s.find(string str, int pos)	Devuelve la posición en donde comienza la subcadena str desde s[pos].
s.find_first_of(str,pos)	Posición en donde se encuentra el primer carácter que pertenece a str desde s[pos].
s.find_first_not_of(str,pos)	Posición en donde se encuentra el primer carácter que no está en str desde s[pos].
s.find_last_of(str,pos)	Posición en donde se encuentra el último carácter que pertenece a str desde s[pos].
s.find_las_not_of(str,pos)	Posición en donde se encuentra el último carácter que no está en str desde s[pos].
Operaciones E/S	
stream >> str	Entrada de strings
stream << str	Salida de strings
getline(stream,str,char)	Añade a str todos los caracteres de una línea de la entrada estándar hasta encontrar el carácter char. Por defecto char es igual a ‘\n’.

Operaciones para el tipo de dato `list`

Constructores y asignación	
list<T> v	Constructor por defecto
list<T> l (aList);	Constructor de copia
l = aList	Asignación
Acceso a elementos	
l.front()	Primer valor de la colección
l.back()	Último valor de la colección
Inserción y borrado	
l.push_front (T)	Añade un elemento al principio de la lista
l.push_back (T)	Añade un elemento al final de la lista
l.insert (iterator, T)	Inserta un nuevo elementos antes del iterador
l.swap (list<T>)	Intercambia valores con otra lista
l.pop_front ()	Borra el primer elemento de la lista
l.pop_back ()	Borra el último elemento de la lista
l.remove(T)	Eliminar todos los elementos iguales a uno dado
l.remove_if(predicate)	Eliminar todos los valores que cumplan una condición
l.erase (iterator)	Borra el elemento indicado por el iterador
l.erase (iterator, iterator)	Borra un rango de valores
Tamaño	
l.size ()	Número de elementos en la lista
l.empty ()	Cierto si la lista está vacía
Iteradores	
list<T>::iterator itr	Declara un nuevo iterador
l.begin ()	Iterador que referencia al primer elemento
l.end ()	Iterador que referencia al siguiente al último
list<T>::reverse_iterator ritr	Declara un nuevo reverse_iterator
l.rbegin ()	Reverse_iterator que referencia al último elemento
l.rend ()	Reverse_iterator que referencia al anterior al primero
Otros métodos	
l.reverse()	Invierte la lista
l.sort()	Ordena los elementos de menor a mayor
l.merge(list<T>)	Mezcla con otra lista ordenada
l.sort(comparision)	Ordena los elementos según una función

Operaciones para los tipos de datos `vector` y `deque`

Constructores	
vector<T> v;	Constructor por defecto
vector<T> (int, T)	Constructor con tamaño y valor inicial dados
vector<T> v (aVector);	Constructor de copia
Acceso a elementos	
v[i]	Acceso por índice, también puede asignarse
v.front()	Primer valor de la colección
v.back()	Último valor de la colección
Inserción	
v.push_front (T)	Añade un elemento al principio del vector (solo deque)
v.push_back (T)	Añade un elemento al final del vector
v.insert (iterator, T)	Inserta un nuevo elementos antes del iterador
v.swap (vector<T>)	Intercambia valores con otro vector
Borrado	
v.pop_front ()	Borra el primer elemento del vector (solo deque)
v.pop_back ()	Borra el último elemento del vector
v.erase (iterator)	Borra el elemento indicado por el iterador
v.erase (iterator, iterator)	Borra un rango de valores
Tamaño	
v.capacity ()	Número máximo de elementos del *buffer*
v.size ()	Número de elementos en el vector
v.resize (unsigned, T)	Cambia el tamaño, rellenando con un valor
v.reserve (unsigned)	Pone el tamaño del *buffer*
v.empty ()	Cierto si el vector está vacío
Iteradores	
vector<T>::iterator itr	Declara un nuevo iterador
v.begin ()	Iterador que referencia al primer elemento
v.end ()	Iterador que referencia al siguiente al último
vector<T>::reverse_iterator ritr	Declara un nuevo reverse_iterator
v.rbegin ()	Reverse_iterator que referencia al último elemento
v.rend ()	Reverse_iterator que referencia al anterior al primero

Otros objetos y funciones útiles

```
back_inserter(contendor l); // genera un iterador especial para hacer que
           algoritmos que reemplazan elementos inserten en lugar de reemplazar
istream_iterator<T>(istream &i); // iterador para leer elementos de tipo T desde i
ostream_iterator<T>(ostream &o, const char *s); // iterador para escribir
                                        elementos de tipo T en o separador por s
```

Algoritmos STL

La primer columna indica el prototipo, la segunda la descripción de qué hace y la tercera el valor o tipo de retorno.

f,*l*,*pos* denotan iteradores
i,*j*,*k* denotan enteros
x,*y* denotan elementos
p,*q* denotan funciones que reemplazan un operador bool

accumulate(f, l, i)	suma todos los elementos entre f y l, inciando el acumulador en i	int
count(f, l, x, n)	cuenta las apariciones de x y guarda la cantidad en n	
count_if(f, l, p, n)	cuenta cuantos elementos satisfacen p y guarda la cantidad en n	
equal(f1, l1, f2)	determina si las secuencias son iguales	bool
equal(f1, l1, f2, p)	determina si las secuencias son iguales comparando con p	bool
find(f, l, x)	busca la primer ocurrencia del elemento x	iterator
find_if(f, l, p)	busca el primer elemento que cumpla con p	iterator
max_element(f, l)	busca el mayor elemento en un rango dado	iterador
max_element(f, l, p)	busca el mayor elemento en un rango dado comparando con p	iterador
min_element(f, l)	busca el menor elemento en un rango dado	iterador
min_element(f, l, p)	busca el menor elemento en un rango dado comparando con p	iterador

copy(f1, l1, f2)	copia elementos desde f1...l1 a f2	ending position of output range
fill(f1, l1, x)	reemplaza todos en f1...l1 por x	
fill_n(f1, n, x)	reemplaza n elementos desde f1 por x	
iter_swap(f1,f2)	intercambia dos elementos	
random_shuffle(f,l)	reordena en forma aleatoria	
remove(f, l, x)	elimina todas las ocurrencias de x	iterador al comienza de la parte que ya no se utiliza del contenedor
remove_if(f, l, p)	elimina todos los elementos que cumplen con p	iterador al comienza de la parte que ya no se utiliza del contenedor
replace(f, l, x, y)	reemplaza x por y	
replace_if(f, l, p, y)	reemplaza los elementos que cumplan con p por y	
reverse(f1, l1)	invierte el orden	
swap(x,y)	intercambia dos elementos	
swap_ranges(f1, l1, f2)	intercambia los elementos de f1...l1 con los de f2...	iterador al final del segundo rango
unique(f, l)	remueve elementos repetidos (si está ordenado)	iterador al comienza de la parte que ya no se utiliza del contenedor
unique(f, l, p)	idem al anterior pero comparando con p	iterador al comienza de la parte que ya no se utiliza del contenedor

binary_search(f, l, x)	busca x entre f y l	bool
merge(f1, l1, f2, l2, f3)	copia de f1...l1 y f2...l2 a f3... manteniendo el orden	iterador al final del rango f3...
sort(f, l)	ordena los elementos entre f y l	
sort(f, l, p)	ordena los elementos entre f y l comparando con p como <	