STL - алгоритмы представляют набор готовых функций, которые могут быть применены к STL коллекциям. Хотя в каждом контейнере поддерживается собственный базовый набор операций, стандартные алгоритмы обеспечивают более широкие и комплексные действия. Кроме этого, они позволяют одновременно работать с двумя контейнерами разных типов. Для работы алгоритмов необходимо подключить библиотеку <algorithm> в начале программы. Все алгоритмы представляют собой функции-шаблоны. Это означает, что их можно использовать с контейнерами любых типов.

Большая часть алгоритмов STL построена по единому принципу. Алгоритм получает на вход пару итераторов - интервал. Если алгоритм осуществлял поиск элемента, то будет возвращен либо итератор, указывающий на соответствующий элемент, либо конец интервала. Конец интервала уже не указывает ни на какой элемент.

За счет использования алгоритмов, облегчается работа разработчика, так как предоставляются готовые реализации многих задач. То есть, нет необходимости придумывать что то новое. С использованием алгоритмов возможно создание очень мощных и эффективных программ. По компактности такой код превосходит код, написанный на таких современных языках, как Java и С#.

Алгоритмы подразделены на три основных группы:

1. Функции для перебора всех членов коллекции и выполнения определенных действий над каждым из них:   
   **count(It beg, It end, const T& v)** – подсчет элементов со значением равным v; **count\_if(It beg, It end, UPredicate op)** – подсчет элементов для которых op(el) == true;  
   **find(It beg, It end, const T& v)** – поиск первого элемента со значением v;  
   **find\_if(It beg, It end, UPredicate op)** – поиск первого элемента для которого унарная операция op == true;  
   **for\_each(first, last, function) -** применяет function ко всем объектам;  
   **mismatch(It1 beg, It1 end, It2 cBeg, BPredicate op)** - ищет первое несовпадение между двумя последовательностями, результат возвращается в виде пары итераторов pair<It1,It2>;  
   **equal(It1 beg, It1 end, It2 cBeg)** - истина, если два диапазона равны;  
   **search(FIt beg, FIt end, FIt2 sBeg, FIt2 sEnd)** - поиск первого вхождения последовательности элементов указанной [sbeg;sEnd); **copy(It sBeg, It sEnd, OIt dBeg)** - копирование последовательности;  
   **copy\_backward(BIt1 sBeg, BIt1 sEnd, BIt2 dEnd)** - обратное копирование;  
   **swap(a,b)** - заменяет один обект другим;  
   **iter\_swap(iter1, iter2) -** обменивает объекты на которые указывают два  итератора;  
   **swap\_ranges(first1, last1, first2)** - обменивает соответсвующие объекты в двух диапазонах;  
   **fill(FIt beg, FIt end, const T& v)** - заполнить указанным значением;  
   **fill\_n(OIt beg, Size n, const T& v)** - заполнить указанным значением;  
   **generate(FIt beg, FIt end, Func op)** - заполнить значениями сгенерированными функцией op;  
   **generate\_n(OIt beg, Size n, Func op)** - заполнить значениями сгенерированными функцией op;  
   **replace(FIt beg, FIt end, const T& v, const T& rv)** - заменить в последовательности значения v на rv;  
   **replace\_if(FIt beg, FIt end, UPredicate op, const T& rv)** - заменить в последовательности на rv все элементы для которых op(el) истина;  
   **transform(It sBeg, It sEnd, OIt dBeg, UnaryF op)** - копирование результатов применения op(el);  
   **remove(first, last, value) -** удаляет все объекты из диапазона, равные value;  
   **remove\_if(first, last, predicate) -** удаляет все объекты, удовлетворяющие значениюpredicate;  
   **remove\_copy(first1, last1, first2, value) -** копирует все объекты не равные value, из диапазона 1 в диапазон 2;  
   **remove\_copy\_if(first1, last1, first2, predicate) -** копирует все объекты не удовлетворяющие значению predicate, диапазона 1 в диапазон 2;  
   **unique(first, last) -**  удаляет все эквивалентные объекты кроме первого;  
   **unique\_copy(first1, last1, first2) -** копирует один объект из последовательности эквивалентных объектов, из  диапазона 1 в диапазон 2;  
   **reverse(first, last) -** Обращает последовательность объектов из диапазона;
2. Функции для сортировки членов коллекции:   
   **sort((It p, It q, Pr pred) -** Сортирует элементы последовательности в порядке возрастания;  
   **stable\_sort(It p, It q, Pr pred) -** Сортирует элементы, сохраняя порядок элементов с одинаковыми значениями относительно друг друга;  
   **partial\_sort(It p, It middle, It q, Pr pred) -** Переставляет элементы последовательности так, что элементы межу итераторами p и q располагаются в том порядке, как если бы последовательность была отсортирована, а элементы в оставшейся части - в произвольном порядке;  
   **nth\_element(It p, It q, It nth, Pr pred) -** Позволяет получить n-й по порядку элемент (n-й по счету, как если бы массив был отсортирован), переставляя элементы таким образом, что все элементы до него меньше, либо равны ему, а элементы после - больше, либо равны ему;  
   **binary\_search(It p, It q, const T &x) -** Возвращает true, если в упорядоченной последовательности есть элемент, значение которого равно x, false в противном случае.; **merge(It p, It q, Itr i, Itr j, Iter out, Pr pred) -** Сортирует две последовательности слиянием;  
   **min(const T &a,const T &b ) -** Возвращает минимальный элемент;  
   **max(const T &a,const T &b ) -** Возвращает максимальный элемент;  
   **min\_element(It p, It q) -** Возвращает итератор на минимальный элемент последовательности;  
   **max\_element(It p, It q) -** Возвращает итератор на максимальный элемент последовательности;
3. Функции для выполнения определенных арифметических действий над членами коллекции: **Accumulate, inner\_product, partial\_sum, adjacent\_difference**

Так же алгоритмы можно поделить на следующие виды:

1. Алгоритмы поиска. Все алгоритмы поиска возвращают итератор на элемент, а не сам элемент.
2. Алгоритмы сортировки
3. Алгоритмы удаления элементов

**Примеры использования некоторых функций**

Пример 1. Использования функции swap – замена местами двух элементов.

#include <iostream>

#include <vector>

using namespace std;

void main()

{

vector<int> v;

for(int i=0;i<5;i++)

{

v.push\_back(i);

cout<<v[i]<<'\t';

}

cout<<endl;

swap(v[2], v[4]);

for(int i=0;i<5;i++)

{

cout<<v[i]<<'\t';

}

cout<<endl;

}

Пример 2. Использование функции transform. Создаем вектор и дважды трансформируем его, сначала увеличив каждый элемент на единицу, затем увеличив каждый элемент на свое же значение.

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

int increase (int i){ return ++i; }

void main ()

{

vector<int> v;

vector<int>::iterator i;

for (int i=1; i<6; i++)

v.push\_back (i\*10);

transform (v.begin(), v.end(), v.begin(), increase);

cout << "Vector contains:\n";

for (i=v.begin(); i!=v.end(); i++)

cout<< \*i<<'\t';

cout << endl;

}

Пример 3. Использование функции generate и for\_each. Создаем вектор и заполняем его начальными значениями. Затем с помощью функции generate перезаписываем этот же вектор случайными числами. И в последнем действии показываем весь вектор на экран с помощью функции for\_each.

#include <iostream>

#include <algorithm>

#include <vector>

#include <time.h>

using namespace std;

int generate1(){return rand()%100;}

void print(int v) {cout<<v<<'\t';}

void main ()

{

srand(time(0));

vector<int> v;

for(int i=0;i<7;i++)

v.push\_back(i);

generate(v.begin(), v.end(), generate1);

for\_each(v.begin(),v.end(),print);

cout<<"\n";

}

Пример 4. Использование функции find. Сперва вектор заполняется пользователем с клавиатуры. Затем пользователь вводит число, которое нужно найти и программа, с помощью функции find, осуществляет поиск, возвращая итератор, указывающий на найденный элемент. Финальное действие – это вывод найденного числа.

#include <iostream>

#include <algorithm>

#include <vector>

#include <time.h>

using namespace std;

void print(int v) {cout<<v<<' ';}

void main ()

{

vector<int> v;

vector<int>::iterator i;

int n=1;

while(n!=0)

{

cout<<"Enter number. End - 0"<<endl;

cin>>n;

if(n!=0){v.push\_back(n);}

}

cout<<"Enter number for search"<<endl;

cin>>n;

i=find(v.begin(), v.end(), n);

if (i==v.end()) cout<<"Not found"<<endl;

else {cout<<"Found"<<endl;}

for\_each(i,i+1,print);

}