Пример 1: написать функцию, добавляющую в конец списка

вещественных чисел элемент, значение которого равно

среднему арифметическому всех его элементов.

# include < iostream >

# include < list >

using namespace std;

void g (list <double> &lst) {

list < double > :: const\_iterator p = lst.begin ();

double s = 0; int n;

while ( p!=lst.end ()) {

s = s+\*p; n++; p++;

}

if (n != 0) lst.push\_back (s/n); // lst.push\_back (s/lst.size());

}

Пример 2: написать функцию, формирующую по заданному

вектору целых чисел список из элементов вектора с четными

значениями и распечатывающую его.

# include < iostream >

# include < vector >

# include < list >

using namespace std;

void g (vector <int> &v, list <int> &lst) {

int i;

for (i = 0; i < v.size(); i++)

if (!(v[i]%2)) lst.push\_back(v[i]);

list < int > :: const\_iterator p = lst.begin ();

while ( p!=lst.end ()) {

cout << \*p <<endl; p++;

}

}

int main () {

vector < int > v(20); list < int > lst; int i;

for (i=0; i<20; i++) v[i] = i;

cout << "vector is created" <<endl;

g (v, lst);

return 0;

}

Пример 3: написать функцию, формирующую по заданному списку

целых чисел вектор из элементов списка с четными значениями

и распечатывающую его.

# include < iostream >

# include < vector >

# include < list >

using namespace std;

void g (vector <int> &v, list <int> &lst) {

list < int > :: const\_iterator p = lst.begin ();

while ( p!=lst.end ()) {

if (!(\*p%2)) v.push\_back(\*p); p++;

}

for (int i=0; i < v.size(); i++) cout << v[i] <<endl;

}

int main () {

vector < int > v(20); list < int > lst;

for (i=0; i<20; i++) lst.push\_back(i);

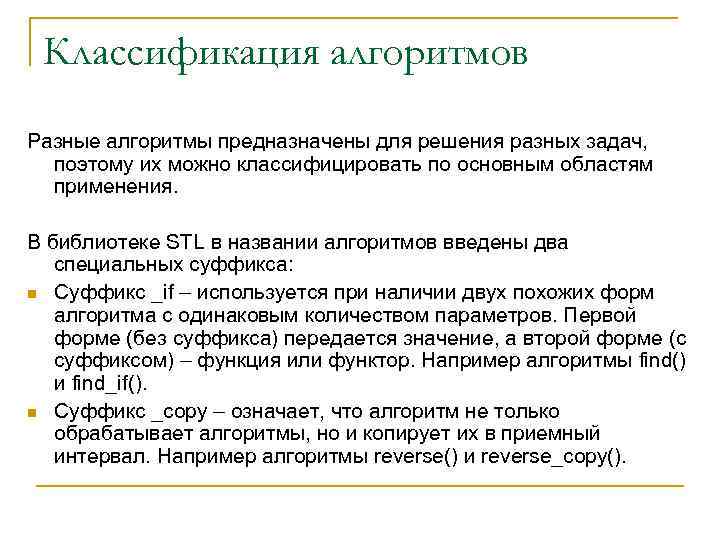
cout << "list is created" <<endl;

g (v, lst);

return 0;

}

Алгоритмы Любой алгоритм STL работает с одним или несколькими интервалами, заданными при помощи итераторов. Для первого интервала всегда задаются две границы: начало и конец в формате [beg, end). Для остальных интервалов обычно задается только одна граница – начало интервала. Алгоритмы работают в режиме замены, а не в режиме вставки. Поэтому перед вызовом алгоритма необходимо убедиться, что приемный интервал содержит достаточное количество элементов. Для повышения возможностей и гибкости некоторые алгоритмы позволяют передавать пользовательские операции, которые вызываются при внутренней работе алгоритма. Такие операции оформляются в виде функций или функторов.

Классификация алгоритмов Разные алгоритмы предназначены для решения разных задач, поэтому их можно классифицировать по основным областям применения. В библиотеке STL в названии алгоритмов введены два специальных суффикса: n Суффикс \_if – используется при наличии двух похожих форм алгоритма с одинаковым количеством параметров. Первой форме (без суффикса) передается значение, а второй форме (с суффиксом) – функция или функтор. Например алгоритмы find() и find\_if(). n Суффикс \_copy – означает, что алгоритм не только обрабатывает алгоритмы, но и копирует их в приемный интервал. Например алгоритмы reverse() и reverse\_copy().

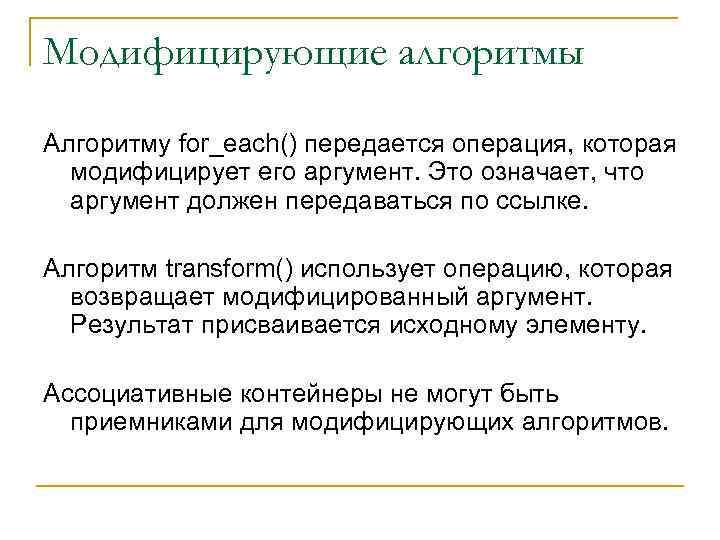
Классификация алгоритмов Все алгоритмы можно разделить на следующие группы: n не модифицирующие алгоритмы, n алгоритмы удаления, n алгоритмы перестановок, n алгоритмы сортировки, n алгоритмы упорядоченных интервалов, n численные алгоритмы. Некоторые алгоритмы можно отнести к нескольким категориям.

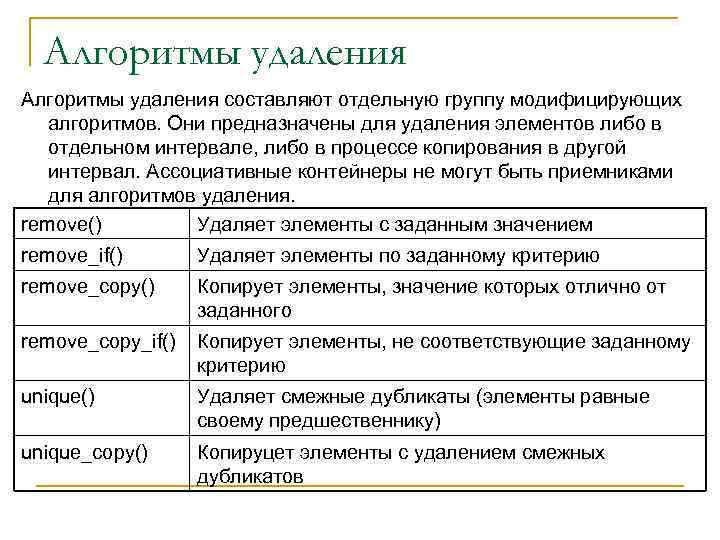
Не модифицирующие алгоритмы сохраняют как порядок следования обрабатываемых элементов, так и их значения. Они работают с итераторами ввода и прямыми итераторами, поэтому они могут вызываться для всех стандартных контейнеров. for\_each() Выполняет операцию с каждым элементом контейнера count() Возвращает количество элементов count\_if() Возвращает количество элементов, удовлетворяющих заданному критерию min\_element() Возвращает элемент с минимальным значением max\_element() Возвращает элемент с максимальным значением find() Ищет первый элемент с заданным значением find\_if() Ищет первый элемент, удовлетворяющий заданному критерию

Не модифицирующие алгоритмы search\_n() Ищет первые n последовательных элементов с заданными свойствами search() Ищет первое вхождение подинтервала find\_end() Ищет последнее вхождение подинтервала find\_first\_of() Ищет первый из нескольких возможных элементов adjacent\_find() Ищет два смежных элемента, равных по заданному критерию equal() Проверяет, равны ли два интервала mismatch() Возвращает первый различающийся элемент в двух интервалах lexicographical\_compare() Проверяет, что один интервал меньше другого по лексикографическому критерию

Модифицирующие алгоритмы изменяют значения элементов. Модификация производится непосредственно внутри интервала или в процессе копирования в другой интервал. Если элементы копируются в приемный интервал, то исходный интервал остается без изменений. for\_each() Выполняет операцию с каждым элементом copy() Копирует интервал, начиная с первого элемента copy\_backwards() Копирует интервал, начиная с последнего элемента transform() Модифицирует (и копирует) элементы, объединяет элементы двух интервалов merge() Производит слияние двух интервалов swap\_ranges() Меняет местами элементы двух интервалов

Модифицирующие алгоритмы fill() Заменяет каждый элемент заданным значением fill\_n() Заменяет n элементов заданным значением generate() Заменяет каждый элемент результатом операции generate\_n() Заменяет n элементов результатом операции replace() Заменяет элементы с заданным значением другим значением replace\_if() Заменяет элементы, соответствующие критерию, заданным значением replace\_copy() Заменяет элементы с заданным значением при копировании интервала replace\_copy\_if() Заменяет элементы, соответствующие критерию, при копировании интервала

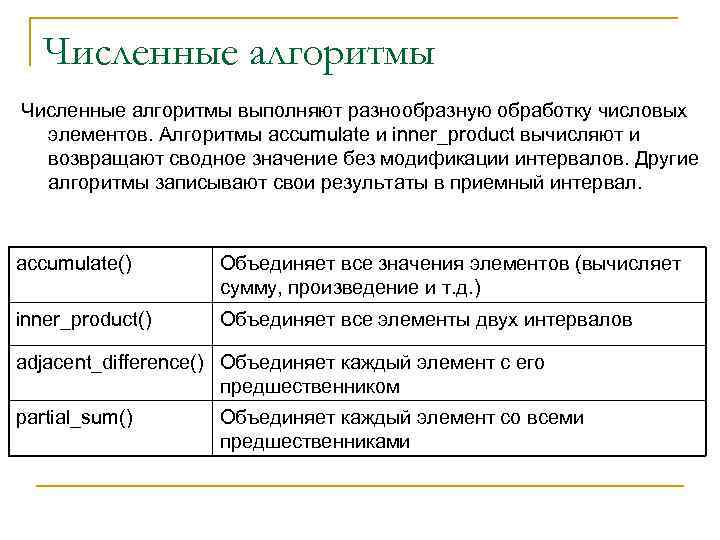
Модифицирующие алгоритмы Алгоритму for\_each() передается операция, которая модифицирует его аргумент. Это означает, что аргумент должен передаваться по ссылке. Алгоритм transform() использует операцию, которая возвращает модифицированный аргумент. Результат присваивается исходному элементу. Ассоциативные контейнеры не могут быть приемниками для модифицирующих алгоритмов.

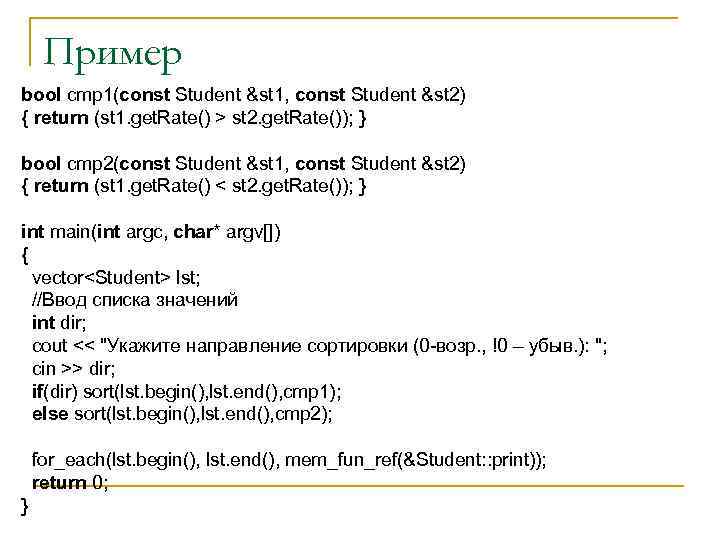
Алгоритмы удаления составляют отдельную группу модифицирующих алгоритмов. Они предназначены для удаления элементов либо в отдельном интервале, либо в процессе копирования в другой интервал. Ассоциативные контейнеры не могут быть приемниками для алгоритмов удаления. remove() Удаляет элементы с заданным значением remove\_if() Удаляет элементы по заданному критерию remove\_copy() Копирует элементы, значение которых отлично от заданного remove\_copy\_if() Копирует элементы, не соответствующие заданному критерию unique() Удаляет смежные дубликаты (элементы равные своему предшественнику) unique\_copy() Копируцет элементы с удалением смежных дубликатов

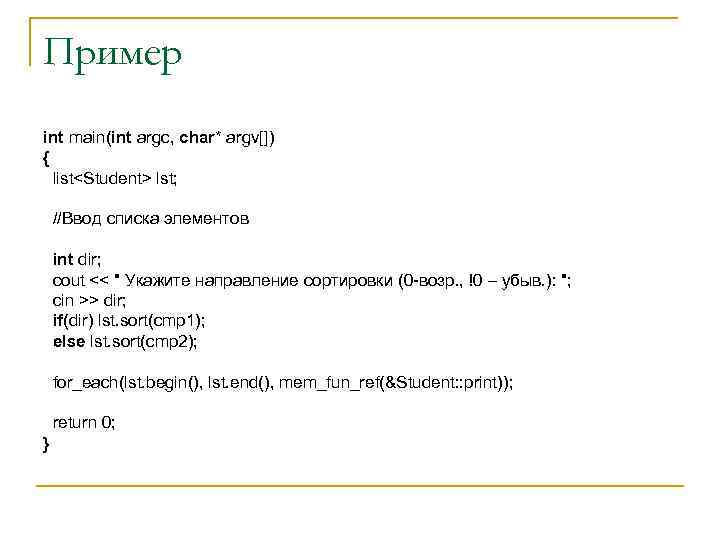
Алгоритмы перестановок Алгоритмами перестановок называют алгоритмы, изменяющие порядок следования элементов (но не их значения) посредством присваивания и перестановки их значения. reverse() Переставляет элементы в обратном порядке reverse\_copy() Копирует элементы, переставленные в обратном порядке rotate() Производит циклический сдвиг элементов rotate\_copy() Копирует элементы с циклическим сдвигом next\_permutation() Переставляет элементы prev\_permutation() Переставляет элементы random\_shuffle() Переставляет элементы в случайном порядке partition() Изменяет порядок следования элементов так, что элементы соответствующие критерию, оказываются спереди stable\_partition() То же, но сохранением относительного расположения элементов

Алгоритмы сортировки sort() Сортирует все элементы (алгоритм quicksort) stable\_sort() Сортирует с сохранением порядка следования равных элементов (алгоритм mergesort) partial\_sort() Сортирует до тех пор, пока первые n элементов не будут упорядочены (алгоритм heapsort) nth\_element() Сортирует элементы слева и справа от элемента в позиции n make\_heap() Преобразует интервал в кучу push\_heap() Добавляет элемент в кучу pop\_heap() Удаляет элемент из кучи sort\_heap() Сортирует кучу

Алгоритмы упорядоченных интервалов binary\_search() Проверяет, содержит ли интервал заданный элемент includes() Проверяет, что каждый элемент интервала также является элементом другого интервала lower\_bound() Находит первый элемент со значением, большим либо равным заданному upper\_bound() Находит первый элемент со значением, большим заданного equal\_range() Возвращает количество элементов в интервале, равных заданному merge() Выполняет слияние элементов двух интервалов set\_union() Вычисляет упорядоченное объединение двух интервалов set\_intersection() Вычисляет упорядоченное пересечение двух интервалов set\_difference() Вычисляет упорядоченный интервал, содержащий разность двух интервалов set\_symmetric\_differense() Вычисляет упорядоченный интервал, содержащий исключающую разность двух интервалов inplace\_merge() Выполняет слияние двух последовательных, упорядоченных интервалов

Численные алгоритмы выполняют разнообразную обработку числовых элементов. Алгоритмы accumulate и inner\_product вычисляют и возвращают сводное значение без модификации интервалов. Другие алгоритмы записывают свои результаты в приемный интервал. accumulate() Объединяет все значения элементов (вычисляет сумму, произведение и т. д. ) inner\_product() Объединяет все элементы двух интервалов adjacent\_difference() Объединяет каждый элемент с его предшественником partial\_sum() Объединяет каждый элемент со всеми предшественниками

Пример bool cmp 1(const Student &st 1, const Student &st 2) { return (st 1. get. Rate() > st 2. get. Rate()); } bool cmp 2(const Student &st 1, const Student &st 2) { return (st 1. get. Rate() < st 2. get. Rate()); } int main(int argc, char\* argv[]) { vector lst; //Ввод списка значений int dir; cout << "Укажите направление сортировки (0 -возр. , !0 – убыв. ): "; cin >> dir; if(dir) sort(lst. begin(), lst. end(), cmp 1); else sort(lst. begin(), lst. end(), cmp 2); for\_each(lst. begin(), lst. end(), mem\_fun\_ref(&Student: : print)); return 0; }

Пример int main(int argc, char\* argv[]) { list lst; //Ввод списка элементов int dir; cout << " Укажите направление сортировки (0 -возр. , !0 – убыв. ): "; cin >> dir; if(dir) lst. sort(cmp 1); else lst. sort(cmp 2); for\_each(lst. begin(), lst. end(), mem\_fun\_ref(&Student: : print)); return 0; }

<https://youtu.be/3df6HZA5JB4> сортировка разных видов, хорошо!

<https://youtu.be/xa5HOUwxQfY> функторы смотреть