http://www.cplusplus.com/reference/iterator/

Заголовочный файл <memory>

Умные указатели призваны для борьбы с утечками памяти, которые сложно избежать в больших проектах. Они особенно удобны в местах, где возникают исключения, так как при последних происходит процесс раскрутки стека и уничтожаются локальные объекты. В случае обычного указателя — уничтожится переменная-указатель, при этом ресурс останется не освобожденным. В случае умного указателя — вызовется деструктор, который и освободит выделенный ресурс.

|  |  |
| --- | --- |
| auto\_ptr | теряет права владения ресурсом при копировании (присваивании, использовании в конструкторе копий, передаче в функцию по значению)  Depricated |
| unique\_ptr | запрещает копирование  изменение прав владения ресурсом осуществляется с помощью вспомогательной функции std::move |
| shared\_ptr | реализует подсчет ссылок на ресурс. Ресурс освободится тогда, когда счетчик ссылок на него будет равен 0  std::make\_shared<> создает объект заданного типа и возвращает shared\_ptr, указывающий на него:  someFunction(std**::**make\_shared**<**Foo**>**(), getRandomKey()); |
| weak\_ptr | Позволяет разрушить циклическую зависимость |

|  |  |
| --- | --- |
| reset(); | сбрасывает права владения |
| get(); | возвращает сырой (классический) указатель |

|  |  |  |
| --- | --- | --- |
| const\_cast | Убирает так называемые cv спецификаторы (cv qualifiers), то есть const и volatile. volatile встречается не очень часто, так что более известно как приведение типов, предназначенное для убирания const. | int\* pJ =  const\_cast<int \*> (pI); |
| **static\_cast** | Может быть использован для приведения одного типа к другому. Если это встроенные типы, то будут использованы встроенные в C++ правила их приведения. Если это типы, определенные программистом, то будут использованы правила приведения, определенные программистом. static\_cast между указателями корректно, только если один из указателей - это указатель на void или если это приведение между объектами классов, где один класс является наследником другого | int \* p = static\_cast<int\*>(malloc(100)); |
| **dynamic\_cast** | Безопасное приведение по иерархии наследования, в том числе и для виртуального наследования.  Используется RTTI (Runtime Type Information), чтобы привести один указатель на объект класса к другому указателю на объект класса. Классы должны быть полиморфными, то есть в базовом классе должна быть хотя бы одна виртуальная функция. | dynamic\_cast<derv\_class \*> (base\_class\_ptr\_expr)  dynamic\_cast<derv\_class &> (base\_class\_ref\_expr) |
| **reinterpret\_cast** | Можно приводить никак не связанные указатели | reinterpret\_cast<whatever \*> (some \*)  reinterpret\_cast<integer\_expression> (some \*)  reinterpret\_cast<whatever \*>(integer\_expression) |

В чём разница: (нет)

i\_ptr = reinterpret\_cast<int\*> (ui\_ptr);

i\_ptr = (int\*) ui\_ptr;

ntohs, ntohl, htons, htonl

http ://www.intuit.ru/studies/courses/2249/52/lecture/1567?page=5

#include <iostream>

#include <time.h>

using std::cout;

using std::endl;

void Primary()

{

size\_t i = 0;

while (i < 1000000000) i++;

clock\_t time1 = clock();

cout <<"time1=" << time1 << endl;

while (i < 10000000000) i++;

cout << "i=" << i << endl;

clock\_t time2 = clock();

cout <<"time2=" << time2 << endl;

clock\_t dtime = time2 - time1;

cout <<"dtime=" << dtime << endl;

cout << "CLOCKS\_PER\_SEC=" << CLOCKS\_PER\_SEC << endl;

cout << "time=" <<(double)dtime / CLOCKS\_PER\_SEC << " s" << endl;

}

------------------------------------------------------ -

setlocale(0, "");

const char\* towns[] = { "Усть-Каменогорск", "Томск", "Москва", "Санкт-Петербург" };

for (const char\* t : towns) cout << t << endl;

cout << sizeof(towns) << endl;

cout << sizeof(towns[0]) << endl;

int n = sizeof(towns) / sizeof(towns[0]);

cout << "Всего " << n << " городов" << endl;

\_getch();

------------------------------------------------------

long sum(long\* begin, long\* end);

long sum(long\* begin, long\* end)

{

long\* p = begin;

int Sum = \*p;

while (++p != end)

{

Sum += \*p;

}

return Sum;

}

long arr[] = { 1, 5, 4, 3, 3 };

cout << sum(arr, arr + sizeof(arr) / sizeof(arr[0]));

------------------------------------------------------ -

const double \* fl(const double ar[], int n);

const double \* f2(const double[], int);

const double \* f3(const double \*, int);

const double \* (\*ра[3])(const double \*, int) = { fl, f2, f3 };

----------------------------------------------------------

При вызове метода setf (ios\_base: rshowpoint) объект переводится в режим

отображения завершающей десятичной точки, даже если последующие цифры являются

нулями. Метод precision () указывает количество цифр, отображаемых справа от

десятичной точки (если объект выводится в режиме fixed). Все эти установки

сохраняются до тех пор, пока не будут изменены в результате следующего вызова метода.

Вызов метода width () позволяет установить ширину поля для следующей операции

вывода. Эта установка действует только для отображения единственного значения, а

затем возвращается в принимаемое по умолчанию состояние. (По умолчанию ширина

поля равна нулю. Затем ширина увеличивается до точного соответствия

отображаемому значению.)

shared\_ptr, unique\_ptr

make\_shared

reinterpret\_cast, static\_cast

map, vector, iterator

make\_pair

ntohs, ntohl, htons, htonl

virtual, override, final

public, private, protected

namespace

u\_short WSAAPI ntohs( \_In\_ u\_short netshort);

u\_long WSAAPI ntohl( \_In\_ u\_long netlong);

u\_short WSAAPI htons( \_In\_ u\_short hostshort);

u\_long WSAAPI htonl( \_In\_ u\_long hostlong);

ИНТУИТ курс.

http://www.intuit.ru/studies/courses/2249/52/lecture/1567?page=5

template<class T1, class T2>

void PrintPair(pair<T1, T2> p)

{

cout << "{" << p.first << ", " << p.second << "}";

}

template<class T1, class T2>

auto gt(Tl х, Т2 у) -> decltype (х + у) (

{

return х + у;

}

// external.ерр — внешние переменные

// Компилировать вместе с support.ерр

#include <iostream>

using namespace std;

// Внешняя переменная

double warming = 0.3;

// support.срр — использование внешних переменных

// Компилировать вместе с external.срр

#include <iostream>

extern double warming; // использование переменной warming из другого файла

const char \* const months[12] = { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December" };

//Первое ключевое слово const защищает от изменений строки, а второе слово const

//гарантирует, что каждый указатель в массиве будет постоянно указывать на ту же самую строку,

//на которую он указывал изначально.

public:

Vector () ;

double xval() const {return x; } // сообщает значение х

синтаксис для объявления функций, не модифицирующих объект, к которому они имеют

непосредственный доступ.

Ключевое слово thread\_local отражает, что

продолжительность хранения переменной является продолжительностью существования

содержащего ее потока.

Ключевое слово volatile указывает, что значение в ячейке памяти может быть

изменено, даже если в коде программы нет ничего такого, что может

модифицировать ее содержимое.

mutable . С его помощью можно указать, что

отдельный член структуры (или класса) может быть изменен, даже если переменная типа

структуры (или класса) объявлена со спецификатором const.

namespace MEF = myth::elements::fire;

using MEF::flame;

std::cout.setf(std::ios\_base::fixed, std::ios\_base::floatfield);

//Этот вызов устанавливает флаг, который заставляет объект cout использовать

// нотацию с фиксированной точкой. Подобным же образом следующий оператор

// заставляет cout выводить три десятичных знака после точки :

std::cout.precision(3);

// сохранение предыдущего значения точности

std::streamsize prec = std::cout.precision(3);

// восстановление предыдущего значения

std::cout.precision(prec);

// Сохранение исходных флагов

std::ios\_base::fmtflags orig = std::cout.setf(std::ios\_base::fixed);

// Восстановление сохраненных значений

std::cout.setf(orig, std::ios\_base::floatfield);

.....................

double Buf[10];

double\* pD;

pD = new (Buf) double;

cout << pD << endl;

cout << &pD << endl;

cout << \*pD << endl;

cout << "------------" << endl;

cout << Buf << endl;

cout << &Buf << endl;

cout << \*Buf << endl;

Bozo(const char \* fname, const char \* lname); // прототип конструктора

В этом случае его можно использовать для инициализации объекта следующим

образом :

Bozo bozetta = Bozo("Bozetta", "Biggens"); // основная форма

Bozo fufu("Fufu", "O'Dweeb"); // сокращенная форма

Bozo \*pc = new Bozo("Popo", "Le Peu"); // динамический объект

В С++11 можно взамен применять списковую инициализацию :

Bozo bozetta = { "Bozetta", "Biggens" }; // C++11

Bozo fufu{ "Fufu", "O'Dweeb" }; // C++11

Bozo \*pc = new Bozo{ "Popo", "Le Peu" }; // C++11

Bozo(int age);

Тогда в коде можно использовать любую из следующих форм инициализации

объекта :

Bozo dribble = Bozo(44); // первичная форма

Bozo roon(66); // вторичная форма

Bozo tubby = 32; // специальная форма для конструктора с одним аргументом

const Person& Older(const Person& p) const;

const Person& Person::Older(const Person& p) const

{

if (m\_Age > p.Age())

return \*this;

else

return p;

}

const Stock & topval(const Stock & s) const;

//Эта функция имеет неявный доступ к одному объекту и явный — ко второму, и она

//возвращает ссылку на один из двух объектов .Слово const внутри скобок указывает,

//что функция не будет модифицировать объект, к которому получает явный доступ, а

//слово const, которое следует за скобками, устанавливает, что функция не будет

//изменять объект, на который ссылается неявно. Поскольку функция возвращает ссылку на

//один из const - объектов, тип ее возврата также является ссылкой const.

top = stockl.topval(stock2);

top = stock2.topval(stockl);

const Stock & Stock::topval(const Stock & s) const

{

if (s.total\_val > total\_val)

return s; // объект-аргумент

else

return \*this; // вызывающий объект

}

//Тот факт, что возвращаемое значение представляет собой ссылку, означает, что

//возвращаемый объект является тем же самым объектом, который вызвал данный

//метод, а не копией, переданной механизмом возврата.

Между прочим, знание об указателе this позволяет заглянуть "за кулисы" C++.

Например, исходная реализация для Unix использовала утилиту с front, которая

выполняла преобразование программ на C++ в программы на С.Для поддержки

определений методов все, что нужно было сделать — это преобразовать определение метода

C++ вроде такого :

void Stock::show() const

{

cout « "Company: " « company

« " Shares: " << shares << f\nf

<< " Share Price: $" << share\_val

« " Total Worth: $" « total\_val << ' \n•;

}

в следующий код на языке С :

void show(const Stock \* this)

{

cout << "Company: " << this->company

« " Shares: " << this->shares « f\nf

<< " Share Price: $" << this->share\_val

« " Total Worth: $" « this->total\_val « ' \n';

}

To есть квалификатор Stock : : преобразуется в аргумент функции, который

представляет собой указатель на Stock, после чего этот указатель используется для доступа

к членам класса.

Аналогичным образом преобразуются вызовы функций наподобие следующего :

top.show();

в такой вид :

show(stop);

enum class egg { Small, Medium, Large, Jumbo };

enum class t\_shirt { Small, Medium, Large, Xlarge };

В качестве альтернативы вместо ключевого слова class можно использовать

struct.

friend ostream& operator<<(ostream& os, const Person& p);

ostream& operator<<(ostream& os, const Person& p)

{

os << "- - - ";

os << "ID: " << p.m\_ID << ", ";

os << "Name: " << p.m\_Name << ", ";

os << "Age: " << p.m\_Age << ", ";

os << "Gender: " << p.Gender() << endl;

return os;

}

Person persons[] = {

Person("Vladimir", 61, Person::gender::male),

Person("Vera", 59, Person::gender::female),

Person("Maxim", 36, Person::gender::male),

Person("Alexey", 34, Person::gender::male)

};

cout << sizeof(persons) << endl;

cout << sizeof(persons[0]) << endl;

int n = sizeof(persons) / sizeof(persons[0]);

cout << n << endl;

for (Person p : persons) p.Print();

const Person\* person\_max = &person2.Older(person1);

person\_max->Print();

person\_max = persons;

person\_max->Print();

//for (Person p : persons)

// person\_max = &person\_max->Older(p);

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

person\_max = &person\_max->Older(persons[i]);

person\_max->Print();

Person bestFriend = Person("Nikolay");

cout << bestFriend;

cout << persons[0] << persons[1];

К счастью, существует хитрый способ избежать дополнительной работы и в то же

время защититься от будущих аварийных ситуаций в работе программы —

определение необходимых методов как фиктивных закрытых методов :

class Queue

{

private:

Queue(const Queue & q) : qsize(O) { } // упреждающее определение

Queue & operator=(const Queue & q) { return \*this; }

//...

};

Еще два важных отношения : указатель базового класса может указывать на объект

производного класса без явного приведения типа, а ссылка базового класса может

ссылаться на объект производного класса без явного приведения типа :

RatedPlayer rplayerl(1140, "Mallory", "Duck", true);

TableTennisPlayer & rt = rplayer;

TableTennisPlayer \* pt = Srplayer;

rt.NameO; // вызов Name() с помощью ссылки

pt->Name(); // вызов Name() с помощью указателя

Однако указатель или ссылка н а базовый класс позволяет вызывать методы только

базового класса, поэтому с помощью rt или pt невозможно обратиться, например, к

методу ResetRanking

Аналогично объекту базового класса можно присвоить объект производного класса :

Третий вопрос(применение virtual) сложнее, чем первые два.Он определяет,

какой метод используется, если метод вызывается не объектом, а ссылкой или указателем.

Без ключевого слова virtual программа выбирает метод, основываясь на типе ссылки

или указателя.Но если присутствует ключевое слово virtual, программа выбирает

метод, основываясь на типе объекта, на который указывает ссылка или указатель.

void fRef(Person& p\_ref)

{

p\_ref.Print();

}

void fPtr(Person\* p\_ptr)

{

p\_ptr->Print();

}

void fVal(Person p)

{

p.Print();

}

void main()

{

Person a;

FamilyPerson b;

fRef(b);

fPtr(&b);

fVal(b);

b = \*((FamilyPerson\*) &a);

fRef(b);

fPtr(&b);

fVal(b);

**}**

// Друзья производного класса могут приводить тип ссылки или указателя на

// производный класс к ссылке или указателю на базовый класс и затем использовать

// данную ссылку или указатель для вызова дружественной функции базового класса.ostream& operator<<(ostream& os, const FamilyPerson& fp)

{

// cout << (const Person&)fp;

//cout << static\_cast<Person>(fp);

cout << dynamic\_cast<const Person&>(fp);

os << "---> F:&=" << &fp.m\_Father << "=";

if (fp.m\_Father != nullptr) os << fp.m\_Father->Name();

os << " M:&=" << &fp.m\_Mother << "=";

if (fp.m\_Mother != nullptr) os << fp.m\_Mother->Name();

os << endl;

return os;

}

// Конструктор в классе-наследнике

hasDMA::hasDMA(const hasDMA & hs)

: baseDMA(hs) // вызывает конструктор копирования базового класса

{

style = new char[std:: strlen(hs.style) + 1];

std::strcpy(style, hs.style);

}

// Конструктор копирования в классе-наследнике

hasDMA::hasDMA(const hasDMA & hs)

: baseDMA(hs) // вызывает конструктор копирования базового класса

{

style = new char[std:: strlen(hs.style) + 1];

std::strcpy(style, hs.style);

}

// Конструктор присваивания в классе-наследнике

hasDMA & hasDMA::operator=(const hasDMA & hs)

{

if (this == &hs)

return \*this;

baseDMA::operator=(hs); // копирование базовой части

delete[] style; // подготовка к операции new для style

style = new char[std::strlen(hs.style) + 1];

std::strcpy(style, hs.style);

return \*this";

}

/\*explicit \*/Person(const string& name = "None", int age = 0, gender g = gender::undefined);

Star::Star double() { ... } // преобразует star в double

Star::Star const char \*(){ ... } // преобразует в const char

Если функция возвращает созданный в ней временный объект, то ссылку

использовать не следует.Например, показанный ниже метод вызывает конструктор для

создания нового объекта, и затем возвращает копию этого объекта :

Vector Vector::operator+(const Vector & b) const

{

return Vector(x + b.x, у + b.y);

}

Если функция возвращает объект, переданный в нее через ссылку или указатель, то

нужно возвращать объект по ссылке.Например, следующий код возвращает по ссылке

либо объект, который вызывает функцию, либо объект, переданный в качестве

аргумента :

const Stock & Stock : itopval(const Stock & s) const

{

if (s.total\_val > total\_val)

return s; // объект-аргумент

else

return \*this; // вызывающий объект

}

const Stock & Stock::topval(const Stock & s) const

{

if (s.total\_val > total\_val)

return s; // объект-аргумент

else

return \*this; // вызывающий объект

}

Здесь метод возвращает ссылку либо на this, либо на s.Поскольку и this, и s

объявлены как const, функция не может изменять их — а значит, и возвращаемая ссылка

также должна быть объявлена как const.

Учтите, что если функция объявляет аргумент как ссылку или указатель на const,

она не сможет передать этот аргумент в другую функцию, кроме случаев, когда эта

другая функция также обещает не изменять аргумент.

Помните, что одно из проявлений отношения является состоит в том, что

указатель на базовый класс может указывать на объект производного класса, а ссылка на

базовый класс может ссылаться на объект производного класса без явного приведения

типа.И помните, что обратное -.

В общем, на вопрос "Можно ли присвоить объект базового класса производному

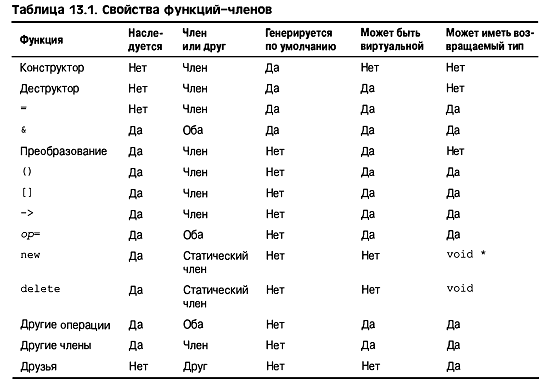
объекту ? " следует ответить: "Возможно". Можно, если производный класс имеет

конструктор, который определяет преобразование объекта базового класса в объект

производного класса.Можно и тогда, когда в производном классе определена операция

присваивания объекта базового класса объекту производного класса.Если ни того, ни

другого нет, присваивание невозможно без явного приведения типа.

****

Конструктор, используемый при

включении, имеет вид :

class Student

{

private:

string name; // используется объект string для имени

valarray<double> scores; // используется объект valarray<double> для оценок

Student(const - char \* str, const double \* pd, int n)

: name(str), scores(pd, n) { } // используются имена объектов для включения

В новой версии примера для наследуемых классов должен использоваться список

инициализаторов членов, в котором для указания конструктора вместо имени члена

применяется имя класса :

class Student : private std::string, private std::valarray<double>

Student(const char \* str, const double \* pd, int n)

: std::string(str), ArrayDb(pd, n) { } // используются имена классов

// для наследования

При включении методы вызывались для объектов :

double Student::Average() const

{

if (scores.size() > 0)

return scores.sum() / scores.size();

else

return 0;

}

Однако здесь наследование позволяет применять имя класса и операцию

разрешения контекста для вызова методов базовых классов :

double Student::Average() const

{

if (ArrayDb::size() > 0)

return ArrayDb::sum() / ArrayDb::size();

else

return 0;

Этот код возвращает ссылку на унаследованный объект string, который

находится в вызывающем объекте Student:

const string & Student::Name() const

{

return (const string &)\*this;

}

Явное указание имени функции с именем ее класса не работает для дружественных

функций, т.к.дружественная функция не принадлежит этому классу.Но для

корректного вызова функций можно использовать явное приведение типа к базовому классу.

В принципе, это та же техника, что и при доступе к объектам базового класса в

методах класса.Но в случае с друзьями доступно имя объекта Student, поэтому вместо

\*this в коде используется имя объекта.Например, рассмотрим следующее

определение дружественной функции :

ostream & operator<<(ostream & os, const Student & stu)

{

os « "Scores for " «(const string &) stu « " :\n";

}

template <class T>

class SortedArray

{

... // подробности не показаны

};

Для создания шаблона SortedArray, специализированного для типа const char \*,

в современной нотации нужен примерно такой код :

template <> class SortedArray<const char \*>

SortedArray<int> scores; // используется общее определение

SortedArray<const char \*> dates; // используется специализированное определение

// Общий шаблон

template <class T1, class T2> class Pair { ... };

// Специализация, в которой для Т2 указан тип int

template <class T1> class Pair<T1, int> { ... };

Угловые скобки <>, следующие за ключевым словом template, объявляют

параметры типов, которые пока еще не специализированы, таким образом, второе

объявление указывает для Т2 тип int, но оставляет параметр Т1 открытым.Обратите

внимание, что указание всех типов приводит к пустым угловым скобкам и получению

завершенной явной специализации :

// Специализация, в которой для Т1 и Т2 указан тип int

template <> class Pair<int, int> { ... };

Если у компилятора есть выбор, он применяет наиболее специальный шаблон.Вот

что произойдет для трех приведенных выше шаблонов :

Pair<double, double> pi; // используется общий шаблон Pair

Pair<double, int> p2; // используется частичная специализация Pair<Tl, int>

Pair<int, int> рЗ; // используется явная специализация Pair<int, int>

Можно частично специализировать существующий шаблон, введя специальную

версию для указателей :

template<class Т> // общая версия

class Feeb { ... };

template<class T\*> // частичная специализация с указателем

class Feeb { ... }; // измененный код

Если предоставить тип, который не является указателем, компилятор задействует

общую версию, а если использовать указатель, компилятор выберет специализацию с

указателем :

Feeb<char> fbl; // используется общий шаблон Feeb (Т - это char)

Feeb<char \*> fb2; // используется специализация Feeb Т\* (Т — это char)

#include <iostream> // Работа с вводом-выводом

#include <fstream> // Работа с файлами

#include <conio.h> // Работа с вводом-выводом

#include <string> // Работа со строками

using namespace std;

int main(int argc, char \*argv[])

{

using namespace std;

setlocale(0, "");

ifstream file;

string fName(argv[1]);

file.open(fName);

cout << fName << endl;

if (file.is\_open() == false)

{

cerr << "Can't open file" << endl;

\_getch();

exit(EXIT\_FAILURE);

}

string item;

int count = 0;

getline(file, item);

while (file) // до тех пор, пока нет ошибок ввода

{

++count;

cout << count << " : " << item << endl;

getline(file, item);

}

cout << "Done\n";

file.close();

\_getch();

return 0;

}

struct Point

{

double X;

double Y;

// Конструктор

Point(double x = 0, double y = 0)

{

X = x;

Y = y;

}

};

struct Triangle

{

Point P1;

Point P2;

Point P3;

Triangle(Point a = {0, 0}, Point b = {0, 0}, Point c = {0, 0})

{

P1 = a;

P2 = b;

P3 = c;

}

};

}

#include <iostream>

#include <conio.h>

#include <vector>

#include <algorithm>

#include <iterator>

using namespace std;

typedef unsigned int uint;

template <class T>

void Print(const T& rr) {cout << rr << "=";}

template <class T>

void Up(T& rr) {rr++;}

template <class T>

void PrintVector(vector<T> v)

{

for (uint i : v) cout << i << " ";

cout << endl;

}

bool UpToDown(const int& a, const int& b) {return a > b;}

void main(int argc, char \*argv[])

{

vector<long> m(4);

for (uint i = 0; i < m.size(); i++) m[i] = rand();

m.push\_back(9);

m.erase(m.begin() + 1, m.begin() + 2);

m.insert(m.end(), m.begin(), m.end());

m.insert(m.end(), m.begin(), m.end());

for (uint i = 0; i < m.size(); i++) cout << m[i] << " ";

cout << endl;

for (vector<long>::iterator iter = m.begin(); iter != m.end(); ++iter) cout << \*iter << " ";

cout << endl;

for (auto iter = m.begin(); iter != m.end(); ++iter) cout << \*iter << " ";

cout << endl;

for each (auto i in m) cout << i << " ";

cout << endl;

for (auto i : m) cout << i << " ";

cout << endl;

random\_shuffle(m.begin(), m.end());

PrintVector(m);

sort(m.begin(), m.end());

PrintVector(m);

sort(m.begin(), m.end(), UpToDown);

PrintVector(m);

for\_each(m.begin(), m.end(), Print<long>);

cout << endl;

for (auto i : m) i++;

PrintVector(m);

for (auto& i : m) i++;

PrintVector(m);

for (auto& i : m) Up(i);

PrintVector(m);

int a[] = { 2, 30, 4, 6, 3, 45, 56, 34, 5 };

for (int i : a) cout << i << " "; cout << endl;

int n = sizeof(a) / sizeof(a[0]);

sort(a, a + n);

for (int i : a) cout << i << " "; cout << endl;

//copy(m.begin(), m.begin() + n, a);

//for (int i : a) cout << i << " "; cout << endl;

copy(a, a + n, m.begin());

PrintVector(m);

ostream\_iterator<int, char> out\_iter(cout, "-");

copy(m.begin(), m.end(), out\_iter);

cout << endl;

for (auto i : m) \*out\_iter = i;

cout << endl;

copy(m.rbegin(), m.rend(), out\_iter);

cout << endl;

//copy(istream\_iterator<long, char>(cin), istream\_iterator<long, char>(), m.begin());

//PrintVector(m);

copy(m.begin(), m.begin() + 1, insert\_iterator<vector<long>>(m, m.begin() + 10));

PrintVector(m);

\_getch();

}

typedef double \* iterator;

iterator find\_ar(iterator begin, iterator end, const double & val)

{

iterator ar;

for (ar = begin; ar != end; ar + +)

if (\*ar == val)

return ar;

return end; // признак того, что значение не найдено

}

#!

Для различения префиксной и постфиксной версий операции + +в C++ принято

соглашение о том, что operator++ () — это префиксная форма, a operator++ (int)—

постфиксная; аргумент никогда не используется, поэтому в именовании не нуждается.



set<string> family;

set\_union(family1.begin(), family1.end(), family2.begin(), family2.end(), insert\_iterator<set<string>>(family, family.begin()));

Функции set\_intersection() и set\_dif ference() находят пересечение и разность двух наборов и обладают тем же интерфейсом, что и set\_union().

template<class T1, class T2>

void PrintPair(pair<T1, T2> p)

{

cout << "{" << p.first << ", " << p.second << "}";

}

pair<long, string> item2(34, "Someone");

dictionary.insert(item2);

for (auto item : dictionary) cout << item.first << ", " << item.second << endl;

cout << dictionary.count(34) << endl;

auto lb = dictionary.upper\_bound(34);

PrintPair(\*lb);

auto er = dictionary.equal\_range(34);

//pair<multimap<long, string>::iterator, multimap<long, string>::iterator> er = dictionary.equal\_range(34);

for (auto it = er.first; it != er.second; ++it) PrintPair(\*it);

typedef multimap<long, string>::iterator mMapIterator;

pair<mMapIterator, mMapIterator> er = dictionary.equal\_range(34);

for (mMapIterator it = er.first; it != er.second; ++it) PrintPair(\*it);

bool WorseThan(const Review & rl, const Review & r2);

sort(books.begin(), books.end(), WorseThan);

код удалил бы из списка three все элементы, которые больше 100:

bool tooBig(int n) { return n > 100; }

list<int> scores;

scores.remove\_if(tooBig);

#include <iostream>

#include <conio.h>

#include <string>

#include <map>

using namespace std;

struct AddressTagConfig

{

unsigned int sectorAdress;

unsigned int ioa;

bool operator<(AddressTagConfig atc) const

{

return sectorAdress < atc.sectorAdress || sectorAdress == atc.sectorAdress && ioa < atc.ioa;

}

};

struct DataTagConfig

{

unsigned long id;

string name;

string type;

};

void main()

{

map<AddressTagConfig, DataTagConfig> tagsConfig;

pair<AddressTagConfig, DataTagConfig> pair1 = { { 1, 24 }, { 1, "Tag1", "float" } };

pair<AddressTagConfig, DataTagConfig> pair2 = { { 2, 12 }, { 2, "Tag2", "float" } };

pair<AddressTagConfig, DataTagConfig> pair3 = { { 1, 12 }, { 3, "Tag2", "float" } };

tagsConfig.insert(pair1);

tagsConfig.insert(pair2);

tagsConfig.insert(pair3);

for (auto tagC : tagsConfig)

{

cout << "{" << tagC.first.sectorAdress << ", " << tagC.first.ioa << "} = {";

cout << tagC.second.id << ", " << tagC.second.name << ", " << tagC.second.type << "}" << endl;

}

AddressTagConfig tagNeedFound = { 2, 12 };

DataTagConfig tagFonded = (\*tagsConfig.find(tagNeedFound)).second;

cout << "{" << tagFonded.id << ", " << tagFonded.name << ", " << tagFonded.type << "}" << endl;

\_getch();

}

#include <algorithm>

#include <functional>

using namespace std;

double solve(double x)

{

return x / 2;

}

void main()

{

vector<double> numbs = { 36, 39, 42, 45, 48 };

vector<double> numbs = { 36, 39, 42, 45, 48 };

ostream\_iterator<double, char> out(cout, " ");

transform(numbs.begin(), numbs.end(), out, solve);

}

vector<double> numbs1 = { 36, 39, 42, 45, 48 };

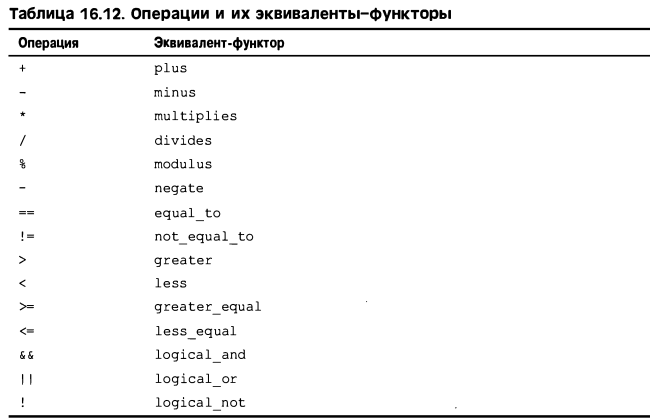
vector<double> numbs2 = { 8, 39, 7, 45, 48 };

ostream\_iterator<double, char> out(cout, " ");

transform(numbs1.begin(), numbs1.end(), numbs2.begin(), out, divides<double>());

cout << endl;

transform(numbs1.begin(), numbs1.end(), out, bind1st(divides<double>(), 2.0) );



#include <iostream>

#include <conio.h>

#include <string>

#include <vector>

#include <set>

#include <map>

#include <algorithm>

#include <functional>

using namespace std;

template <class T>

void PrintVector(vector<T> vector)

{

for (T element : vector) cout << element << " ";

cout << endl;

}

void main()

{

vector<double> numbs = { 1, 2, 4, 5, 2 };

PrintVector(numbs);

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

{

random\_shuffle(numbs.begin(), numbs.end());

PrintVector(numbs);

}

vector<double> numbsCopy(numbs.size());

//numbsCopy.resize(numbs1.size());

transform(numbs.begin(), numbs.end(), numbsCopy.begin(), bind1st(multiplies<double>(), 2));

PrintVector(numbsCopy);

cout << endl;

sort(numbs.begin(), numbs.end());

PrintVector(numbs);

while (next\_permutation(numbs.begin(), numbs.end())) PrintVector(numbs);

string s = "aab";

cout << s << endl;

while (next\_permutation(s.begin(), s.end())) cout << s << endl;

\_getch();

}

// usealgo.срр -- использование нескольких элементов STL

#include <iostream>

#include <string>

#include <vector>

#include <set>

#include <map>

#include <iterator>

#include <algorithm>

#include <cctype>

#include <conio.h>

using namespace std;

char toLower(char ch) { return tolower(ch); }

string & ToLower(string & st);

void display(const string & s);

void main()

{

vector<string> words;

cout << "Enter words (enter quit to quit) :\n"; // запрос на ввод слов

string input;

while (cin >> input && input != "quit")

words.push\_back(input);

cout << "You entered the following words:\n"; // отображение введенных слов

for\_each(words.begin(), words.end(), display);

cout << endl;

// Помещение слов в набор с преобразование букв в строчные

set<string> wordset;

transform(words.begin(), words.end(), insert\_iterator<set<string>>(wordset, wordset.begin()), ToLower);

cout << "\nAlphabetic list of words:\n"; // список слов в алфавитном порядке

for\_each(wordset.begin(), wordset.end(), display);

cout << endl;

// Помещение слова и его частоты в карту

map<string, int> wordmap;

set<string>::iterator si;

for (si = wordset.begin(); si != wordset.end(); si++)

wordmap[\*si] = count(words.begin(), words.end(), \*si);

// Отображение содержимого карты

cout << "\nWord frequency:\n"; // частота появления слов

for (si = wordset.begin(); si != wordset.end(); si++)

cout << \*si << " : " << wordmap[\*si] << endl;

\_getch();

}

string & ToLower(string & st)

{

transform(st.begin(), st.end(), st.begin(), toLower);

return st;

}

void display(const string & s)

{

cout << s << " ";

}

vector<double> vedl(10), ved2(10), ved3(10);

array<double, 10> vodl, vod2, vod3;

valarray<double> vadl(10), vad2(10), vad3(10);

transform(vedl.begin(), vedl.end(), ved2.begin(), ved3.begin(),

plus<double>());

transform(vodl.begin(), vodl.end(), vod2.begin(), vod3.begin(),

plus<double>());

vad3 = vadl + vad2;

vad3 = log(vadl); // операция log() перегружена

//Или же можно воспользоваться методом apply(), который работает также для неперегруженных функций :

vad3 = vadl.apply(log);

sort(begin(vad1), end(vad1)); // исправление С++11

//если numbers — объект valarray<double>, то следующий оператор

// создает массив значений типа bool, в котором vbool[i] устанавливается равным

// значению numbers[і] > 9, т.е. true или false:

valarray<bool> vbool = numbers > 9;

varint[slice(1, 4, 3)] = 10;

#include <iostream>

#include <string>

#include <algorithm>

#include <conio.h>

using std::cout;

using std::string;

void main()

{

string s = "horse";

reverse(s.begin(), s.end());

cout << s;

\_getch();

}

template <class T>

void Print(initializer\_list<T> il)

{

for (T i : il) cout << i << " ";

cout << endl;

//for (auto it = il.begin(); it != il.end(); it++) cout << \*it << " ";

//for (initializer\_list<T>::iterator it = il.begin(); it != il.end(); it++) cout << \*it << " ";

}

Print({ "One", "Two", "Three", "Four", "Five" });

#include <iostream>

#include <string>

#include <algorithm>

#include <conio.h>

#include <vector>

#include<algorithm>

using namespace std;

template <class T>

void Print(vector<T> v)

{

for (T x : v) cout << x << " ";

cout << endl;

}

bool isMod2(long x) { return x % 2 == 0; }

bool isMod3(long x) { return x % 3 == 0; }

class f\_mod

{

public:

f\_mod(long d = 1) : mod(d){}

bool operator()(long x){ return x % mod == 0; }

private:

long mod;

};

void main()

{

const int N = 3000000;

vector<long> v(N);

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

//Print(v);

// Функциональные объекты

// 1. Указатель на функцию

long count2\_1 = count\_if(v.begin(), v.end(), isMod2);

cout << count2\_1 << endl;

long count3\_1 = count\_if(v.begin(), v.end(), isMod3);

cout << count3\_1 << endl;

// 2. Функтор

long count2\_2 = count\_if(v.begin(), v.end(), f\_mod(2));

cout << count2\_2 << endl;

long count3\_2 = count\_if(v.begin(), v.end(), f\_mod(3));

cout << count3\_2 << endl;

//3. Лямбда-выражение

long count2\_3 = count\_if(v.begin(), v.end(), [](long x){return x % 2 == 0; });

cout << count2\_3 << endl;

long count3\_3 = count\_if(v.begin(), v.end(), [](long x){return x % 3 == 0; });

cout << count3\_3 << endl;

// 4. Задание имени лямбда-выражения

auto mod2 = [](long x){return x % 2 == 0; }; //cout << mod2(6) << endl;

auto mod3 = [](long x){return x % 3 == 0; }; //cout << mod3(6) << endl;

long count4\_1 = count\_if(v.begin(), v.end(), mod2);

cout << count4\_1 << endl;

long count4\_2 = count\_if(v.begin(), v.end(), mod3);

cout << count4\_2 << endl;

// 5. Захватывание переменных в лямбда-выражении

long count5\_1 = 0;

long count5\_2 = 0;

//for\_each(v.begin(), v.end(), [&count5\_1](long x){count5\_1 += x % 2 == 0; });

for\_each(v.begin(), v.end(), [&](long x){count5\_1 += x % 2 == 0; count5\_2 += x % 3 == 0; });

cout << count5\_1 << endl;

cout << count5\_2 << endl;

\_getch();

}

//Когда указано только имя, например, [z], доступ к переменной производится по значению.

//Если имя предваряется символом &, как в[&count], доступ к переменной осуществляется по ссылке.

//Применение[&] предоставляет доступ ко всем автоматическим переменным по ссылке,

//а[=] — доступ ко всем автоматическим переменным по значению.

//Можно также смешивать и сочетать.Например, [ted, &ed] обеспечит доступ к ted по значению, а к ed — по ссылке;

//[&, ted] предоставит доступ к ted по значению, а ко всем остальным автоматическим переменным — по ссылке;

//[=, &ed] обеспечит доступ к ed по ссылке, а к остальным автоматическим переменным — по значению.

// Если не только return

[](double х)->double{int у = х; return х — у; } // возвращаемым типом

// является double

#include <iostream>

#include <conio.h>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

struct TagConfig

{

unsigned int sectorAdress;

unsigned int ioa;

unsigned long id;

string name;

string type;

void Print() { cout << "{" << sectorAdress << ", " << ioa << ", " <<id << ", " << name << ", " << type << "}" << endl; }

};

template <class T>

void PrintVector(vector<T> v)

{

for (T x : v) x.Print();

cout << endl;

}

void main()

{

vector<TagConfig> tagsConfig = {

{ 1, 24, 1, "Tag1", "float" },

{ 2, 12, 2, "Tag2", "float" },

{ 1, 12, 3, "Tag2", "float" }

};

PrintVector(tagsConfig);

TagConfig foundTag;

for\_each(tagsConfig.begin(), tagsConfig.end(), [&foundTag](TagConfig tc){if (tc.id == 2 && tc.ioa == 12) foundTag = tc; });

cout << endl;

foundTag.Print();

\_getch();

}

If your vector is not ordered, use the approach MSN suggested:

if(std::find(vector.begin(), vector.end(), item)!=vector.end()){

// Find the item

}

If your vector is ordered, use binary\_search method Brian Neal suggested:

if(binary\_search(vector.begin(), vector.end(), item)){

// Find the item

}

binary search yields O(log n) worst-case performance, which is way more efficient than the first approach. In order to use binary search, you may use qsort to sort the vector first to guarantee it is ordered.

## Alternative 1:

[**std::find()**](http://en.cppreference.com/w/cpp/algorithm/find) requires a that there is a suitable equality comparator for your node data type, which may be as simple as this:

bool operator ==(node const& l, node const& r)

{

return l.data == r.data;

}

Then, given a required node, you can search for the element. This returns an iterator (or apointer if you're using a plain old array). If you need the index, this requires a little calculation:

auto i = std::find(v.begin(), v.end(), required);

if (i != v.end())

{

std::cout << i->data << " found at index " << i - v.begin() << std::endl;

}

else

{

std::cout << "Item not found" << std::endl;

}

## Alternative 2:

If creating a node is too expensive or you don't have an equality operator, a better approach would be to use [**std::find\_if()**](http://en.cppreference.com/w/cpp/algorithm/find), which takes a predicate (here I use a lambda because it's succinct, but you could [use a functor like in this answer](http://stackoverflow.com/a/14919397/78845)):

// Alternative linear search, using a predicate...

auto i = std::find\_if(v.begin(), v.end(), [](node const& n){return n.data == 444;});

if (i != v.end())

{

std::cout << i->data << " found at index " << i - v.begin() << std::endl;

}

else

{

std::cout << "Item not found" << std::endl;

}

Or is there a quicker way?

Again, it depends. std::find() and std::find\_if() run in linear time (**O**(n)), the same as your for-loop.

That said, using std::find() or std::find\_if() won't involve random access or indexing into the container (they use iterators) but they may require a little bit of extra code compared with your for-loop.

## Alternative 3:

If running time is critical and your array is sorted (say with [std::sort()](http://en.cppreference.com/w/cpp/algorithm/sort)), you could perform a binary-search, which runs in logarithmic time (**O**(log n)). [**std::lower\_bound()**](http://en.cppreference.com/w/cpp/algorithm/lower_bound) implements a binary search for the first element that is not less than the given value. It does not take a predicate unfortunately but requires a suitable less-than comparator for your node data type, such as:

bool operator <(node const& l, node const& r)

{

return l.data < r.data;

}

The invocation is similar to std::find() and returns an iterator, but requires an extra check:

auto i = std::lower\_bound(v.begin(), v.end(), required);

if (i != v.end() && i->data == required.data)

{

std::cout << i->data << " found at index " << i - v.begin() << std::endl;

}

else

{

std::cout << "Item not found" << std::endl;

}

#include <iostream>

#include <conio.h>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

struct TagConfig

{

unsigned int sectorAdress;

unsigned int ioa;

unsigned long id;

string name;

string type;

void Print() { cout << "{" << sectorAdress << ", " << ioa << ", " <<id << ", " << name << ", " << type << "}" << endl; }

};

bool operator< (TagConfig l, TagConfig r)

{

return l.sectorAdress < r.sectorAdress || l.sectorAdress == r.sectorAdress && l.ioa < r.ioa;

}

bool operator== (TagConfig l, TagConfig r)

{

return l.sectorAdress == r.sectorAdress && l.ioa == r.ioa;

}

template <class T>

void PrintVector(vector<T> v)

{

for (T x : v) x.Print();

cout << endl;

}

void main()

{

vector<TagConfig> tagsConfig = {

{ 1, 24, 1, "Tag1", "float" },

{ 2, 12, 2, "Tag2", "float" },

{ 1, 12, 3, "Tag2", "float" }

};

PrintVector(tagsConfig);

sort(tagsConfig.begin(), tagsConfig.end());

PrintVector(tagsConfig);

TagConfig needTag = { 1, 24};

TagConfig foundTag;

//vector<TagConfig>::iterator it;

auto it = lower\_bound(tagsConfig.begin(), tagsConfig.end(), needTag);

if ( it < tagsConfig.end() && (foundTag = \*it) == needTag)

{

cout << "Find!" << endl;

foundTag.Print();

}

else

{

cout << "Not find!" << endl;

}

\_getch();

}

short 2

int 4

long 8

long long 8

float 4

double 4

long double 16

Директивы компиляции.

QMAKE\_CXXFLAGS += -std=c++11

#array to function

**template** <**typename** T, size\_t ***N***>  
**constexpr** size\_t GetSize(T (&) [***N***]) **noexcept**{  
 **return *N***;  
};

#include **<iostream>  
template** <**typename** T, size\_t ***N***>  
**void** PrintArray(T (&array) [***N***]) **noexcept**{  
 **for** (**const auto** element : array)  
 std::cout << element << **" "**;  
};  
  
**int** main()  
{  
 **char** array[] = **"Spring is coming!"**;  
 PrintArray(array);  
 **return** 0;  
}

#error #type

*// Show Error in IDE using compiler*

**template** <**typename** T>  
**class** \_;  
  
**int** main()  
{  
 **constexpr** size\_t N = 10;  
 **int** array[N];  
 \_<**decltype**(array)> show;

**return** 0;  
}

std::cout << **typeid**(std::vector<**int**>::iterator).name();

#include **<boost/type\_index.hpp>**

**template**<**class** T>  
**void** ShowType(T */\*object\*/*)  
{  
 **using** boost::typeindex::type\_id\_with\_cvr;  
   
 std::cout << **"["** << **sizeof**(T) << **"] "**;  
 std::cout << type\_id\_with\_cvr<T>().pretty\_name() << **"\n"**;  
}

#string

#include **<iostream>**#include **<boost/type\_index.hpp>**#include **<vector>  
  
using** boost::typeindex::type\_id\_with\_cvr;  
  
**int** main()  
{  
 std::vector<**int**> x = {1, 2, 3, 4, 5};  
 **auto** y = x.begin();  
  
 std::cout << type\_id\_with\_cvr<**decltype**(x)>().pretty\_name() << **"\n"**;  
 std::cout << type\_id\_with\_cvr<**decltype**(y)>().pretty\_name() << **"\n"**;  
 **return** 0;  
}

#include **<vector>**#include **<iostream>  
  
auto** Print() {}  
  
**template** <**class** T1, **class**... T2>  
**auto** Print(T1&& arg1, T2&&... arg2)  
{  
 std::cout << arg1;  
*// std::cout << "size: (" << sizeof...(arg2) << ")\n";* Print(arg2...);  
}  
  
**int** main()  
{  
 std::vector<**int**> x = {1, 2, 3, 4, 5};  
  
 **auto**&& p1 = x[0];  
 **auto**&& p2 = x[1];  
  
 Print(p1, **", "**, p2, **", "**, x[2], **"; "**, **"exactly!"**);  
  
 **return** 0;  
}

*// #forward\_list and #last*

#include **<forward\_list>**#include **<boost/type\_index.hpp>**#include **<iostream>  
  
using** boost::typeindex::type\_id\_with\_cvr;  
  
**template** <**class** T>  
**void** Print(T container)  
{  
 std::cout << type\_id\_with\_cvr<T>().pretty\_name() << **"\n"**;  
  
 **for** (**const auto**& element : container)  
 std::cout << element << **" "**;  
 std::cout << **"\n"**;  
}  
  
  
**int** main()  
{  
 std::forward\_list<**int**> list = {5, 6, 7};  
  
 **auto** it = list.before\_begin();  
 **auto** lastIt = it;  
  
 **while** (it != list.end())  
 {  
 lastIt = it;  
 ++it;  
 }  
 list.insert\_after(lastIt, 999);  
 Print(list);  
}

#include **<map>  
  
int** main()  
{  
 std::map<size\_t, std::string> people;  
  
 **try** {  
 people.at(5) = **"Test person"**;  
 }  
 **catch** (const std::exception& ex)  
 {  
 std::cout << **"Error: "** << ex.what() << **"\n"**;  
 **return** 1;  
 }

}

#sort #lambda

std::sort(people.begin(), people.end(), [](**const** Person& a, **const** Person& b)  
{  
 **return** a.rating > b.rating;  
});

#find\_if

**auto** result = std::find\_if(people.begin(), people.end(), [](**const** Person& a)  
{  
 **return** 20 <= a.rating && a.rating <= 100;  
})

#copy\_if

std::copy\_if(people.begin(), people.end(), std::back\_inserter(result), [](**const** Person& person)  
{  
 **return** person.rating == 15;  
});

#remove #erase

std::vector<**int**> list = {1, 2, 3, 4, 9, 5, 56, 3, 5, 5, 5, 5,};  
**auto** result = std::remove(list.begin(), list.end(), 5);  
list.erase(result, list.end());

name.erase(std::remove(name.begin(), name.end(), **' '**), name.end());

#remove\_if #erase

**auto** result = std::remove\_if(people.begin(), people.end(), [](Person x)  
{  
 **return** x.rating < 100;  
});

name.erase(std::remove\_if(name.begin(), name.end(), [](**char** x)  
{  
 **return** x == **' '** ||  
 x == **','** ||  
 x == **'.'**;  
}), name.end());

#accumulate #multiplies

**int** list[] = {1, 2, 3};  
**const auto**& result = std::accumulate(std::begin(list), std::end(list), 1, std::multiplies<**int**>());

**int** list[] = {1, 2, 3};  
**const auto**& result = std::accumulate(std::begin(list), std::end(list), 1, [](**int** x, **int** y)  
{  
 **return** x \* y;  
});

#accumulate #string

std::vector<**int**> v{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
  
**auto** sum = std::accumulate(v.begin(), v.end(), 0);  
  
**auto** product = std::accumulate(v.begin(), v.end(), 1, std::multiplies<**int**>());  
  
**auto** s = std::accumulate(std::next(v.begin()), v.end(), std::to\_string(v[0]), [](std::string a, **int** b)  
{  
 **return** a + **'-'** + std::to\_string(b);  
});

**#equal #mismatch #distance**

std::vector<**int**> v1 = {8, 2, 3, 4, 5, 8 };  
  
**int** v2[] = {8, 2, 3, 4, 5, 8};  
  
std::cout << std::equal(std::begin(v1), std::end(v1), std::begin(v2), std::end(v2)) << **"\n"**;  
**auto** result = std::mismatch(std::begin(v1), std::end(v1), std::begin(v2), std::end(v2));  
  
**if** (result.first == v1.end())  
 std::cout << **"equal"** << **"\n"**;  
**else** std::cout << **"mismatch index:"** << std::distance(std::begin(v1), result.first) << **"\n"**;

**#distance**

std::vector<int> v = {8, 90, 75, 500, 14};

std::vector<int>::const\_iterator find\_it = std::max\_element( v.begin(), v.end() );

if( find\_it != v.end() )

{

int index = find\_it - v.begin();

std::cout << index << std::endl;

}

delete[]((char\*)data\_) - удаляем массив типа char

delete[] data - удаляет массив типа T (в т.ч. вызывает деструкторы T)

Удалять нужно то же, что и выделяли.

"operator delete[] (data\_)"  - вызов функции, через которую реализован оператор delete. Эта функция принимает void \* и не вызывает деструкторы.

**Стандартом разрешены (неявные) преобразования от T\* к T const\*, но не T\*\* к T const\*\*.**

Iterator Invalidation Rules (C++0x)

I've not found a tidy, all-in-one table of all the C++0x container iterator invalidation rules.

So, with references, here they are!

**Insertion**

|  |  |  |
| --- | --- | --- |
| **Container** | **Rules** | **n3242 ref.** |
| **Sequence containers** | | |
| vector | all iterators and references before the point of insertion are unaffected, unless the new container size is greater than the previous capacity (in which case all iterators and references are invalidated) | 23.3.6.5/1 |
| deque | all iterators and references are invalidated, unless the inserted member is at an end (front or back) of the deque (in which case all iterators are invalidated, but references to elements are unaffected) | 23.3.3.4/1 |
| list | all iterators and references unaffected | 23.3.5.4/1 |
| forward\_list | all iterators and references unaffected (applies to *insert\_after*) | 23.3.4.5/1 |
| array | (n/a) |  |
| **Associative containers** | | |
| set | all iterators and references unaffected | 23.2.4/9 |
| multiset |
| map |
| multimap |
| **Unordered associative containers** | | |
| unordered\_set | all iterators invalidated when rehashing occurs, but references unaffected | 23.2.5/8 |
| unordered\_multiset |
| unordered\_map |
| unordered\_multimap |
| **Container adaptors** | | |
| stack | inherited from underlying container |  |
| queue |
| priority\_queue |

**Erasure**

|  |  |  |
| --- | --- | --- |
| **Container** | **Rules** | **n3242 ref.** |
| **Sequence containers** | | |
| vector | every iterator and reference after the point of erase is invalidated | 23.3.6.5/3 |
| deque | erasing the last element invalidates only iterators and references to the erased elements and the past-the-end iterator; erasing the first element invalidates only iterators and references to the erased elements; erasing any other elements invalidates all iterators and references (including the past-the-end iterator) | 23.3.3.4/1 |
| list | only the iterators and references to the erased element is invalidated | 23.3.5.4/1 |
| forward\_list | only the iterators and references to the erased element is invalidated (applies to *erase\_after*) | 23.3.4.5/1 |
| array | (n/a) |  |
| **Associative containers** | | |
| set | only iterators and references to the erased elements are invalidated | 23.2.4/9 |
| multiset |
| map |
| multimap |
| **Unordered associative containers** | | |
| unordered\_set | only iterators and references to the erased elements are invalidated | 23.2.5/13 |
| unordered\_multiset |
| unordered\_map |
| unordered\_multimap |
| **Container adaptors** | | |
| stack | inherited from underlying container |  |
| queue |
| priority\_queue |

**Resizing**

|  |  |  |
| --- | --- | --- |
| **Container** | **Rules** | **n3242 ref.** |
| **Sequence containers** | | |
| vector | as per insert/erase | 23.3.6.5/12 |
| deque | as per insert/erase | 23.3.3.4/1 |
| list | as per insert/erase | 23.3.5.3/1 |
| forward\_list | as per insert/erase | 23.3.4.5/25 |
| array | (n/a) |  |

**Bootnote 1**

Unless otherwise specified (either explicitly or by defining a function in terms of other functions), invoking a container member function or passing a container as an argument to a library function shall not invalidate iterators to, or change the values of, objects within that container. [23.2.1/11]

**Bootnote 2**

No swap() function invalidates any references, pointers, or iterators referring to the elements of the containers being swapped. [ Note: The end() iterator does not refer to any element, so it may be invalidated. —end note ] [23.2.1/10]

**Bootnote 3**

Other than the above caveat regarding swap() and the erasure rule for deque, [it's not clear whether "end" iterators are subject to the above listed per-container rules](http://stackoverflow.com/questions/6440392/end-iterator-invalidation-rules); you should assume, anyway, that they are.

[(this post can also be found on Stack Overflow)](http://stackoverflow.com/questions/6438086/iterator-invalidation-rules/6442829#6442829)

[(also see the same post regarding C++03)](http://kera.name/articles/2011/06/iterator-invalidation-rules/)