1. В каких ситуациях применяют типы std::pair и std::tuple

Стоит сказать, что std::pair может иметь только два члена данных, компонентами могут выступать .first, .second, make\_pair()

Std::pair используется для хранения пар

Std::tuple может иметь более двух пар, tuple – расширенная концепция пар на произвольное количество элементов. В качестве компонентов могут выступать: get, make\_tuple и др

Отметим, что двухэлементный кортеж можно инициализировать объектом типа pair. Кроме того, можно присваивать объект типа pair двухэлементному котрежу.

Класс Pair<> содержит специальный конструктор для инициализации его элементов с помощью котрежей.

1. Когда следует использовать контейнер std::array?

Std::array <T, N> - фиксированный массив, память располагается на стеке, является более безопасным по сравнению со встроенным массивом.

Массивы копируют свои элементы во внутренний статический массив. Элементы всегда располагаются в определенном порядке, т.е массивы разновидность упорядоченной коллекции, пользователь может получить прямой доступ к каждому элементу массива за константное время, при условии, что он знает его позицию

1. Когда следует использовать std::vector?

Во-первых, vector – динамический массив, элементы вектора всегда располагаются в определенном порядке, таким образом, вектор – разновидность упорядоченной коллекции(подобно array). Векторы обеспечивают хорошую производительность, если вставка или удаление элементов происходит в конце.

Почему вектор имеет хорошую производительность? Одна из причин заключается в выделении большого объема памяти, чем требуется для хранения всех элементов.

Над векторами можно проводить создание, копирование, удаление

1. Когда следует использовать std::deque?

Std::deque – двусторонняя очередь, идеально подходит для операций вставок с двух концов

1. Когда следует использовать std::list?

Std::list – двусвязный список, подходит для вставки и удаления элементов в произвольное место.

Стоит отметить, что внутренняя структура списка отлична от внутренней структуры array, вектора или дека, объект списка содержит два указателя – якоря, ссылающийся на первый и последний элемент, каждый элемент содержит указатель на предыдущий и следующий элементы

1. Когда следует использовать std::forward\_list?

Std::forward\_list – это односвязный список, этот список обладает тем свойством, что от него требуется максимальная экономия памяти, в связи с этим даже нет size(), существуют только однонаправленные итераторы, якорь последовательного списка не имеет указатель на последний элемент, в связи с этим в списке не подразумевается использовать back, push\_back, pop\_back

1. Какие адаптеры контейнеров есть в стандартной библиотеке?

* Стек (LIFO)
* Очередь (FIFO)
* Очередь с приоритетом (FIFO + sort)

1. Когда следует использовать контейнер circular buffer из boost

Circular buffer – циклический буфер, удобно использовать для хранения истории фиксированной длины при постоянном поступлении новых данных, когда все ячейки buffer заполняются, то ячейки начинают перезаписываться с начала

1. Почему контейнер circular buffer из boost не может войти в стандартную библиотеку?

Не сможет войти, т.к возможна ситуация, когда begin > end

1. Какие типы данных для работы с многомерными массивами вы можете назвать?

* Вектор векторов (vector vector)
* Multiarray
* Обычный массив

2.1

#include <iostream>

#include <vector>

#include <string>

int main() {

std::vector <std::string> sentence;

sentence.push\_back("Hello");

sentence.insert(sentence.end(), ("How", "are", "you?"));

std::cout << "max\_size(): " << sentence.max\_size() << std::endl;

std::cout << "size(): " << sentence.size() << std::endl;

std::cout << "capacity(): " << sentence.capacity() << std::endl;

sentence.reserve(6);

sentence.insert(find(sentence.begin(), sentence.end(), "you?"), "always or not");

sentence.insert(sentence.begin(), ("There", "is", "a coomon text"));

sentence.back() = "!";

std::cout << "size(): " << sentence.size() << std::endl;

std::cout << "capacity(): " << sentence.capacity() << std::endl;

}

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <array>

#include <chrono>

#include <iomanip>

#include <deque>

#include <list>

#include <forward\_list>

class Timer

{

private:

using clock\_t = std::chrono::steady\_clock;

using time\_point\_t = clock\_t::time\_point;

using duration = clock\_t::duration;

public:

Timer(const std::string& name, std::ostream& stream = std::cout) :

m\_name(name), m\_stream(stream), m\_begin(clock\_t::now()), m\_time(duration::zero()), m\_running(true) {}

~Timer() noexcept

{

try

{

pause();

m\_stream << m\_name << " counted " << std::chrono::duration\_cast <std::chrono::microseconds> (m\_time).count() << " microseconds\n";

}

catch (...)

{

// std::abort();

}

}

void pause()

{

if (m\_running)

{

m\_time = m\_time + clock\_t::now() - m\_begin;

m\_running = false;

}

}

void proceed()

{

if (!m\_running)

{

m\_begin = clock\_t::now();

m\_running = true;

}

}

int Calculus()

{

return std::chrono::duration\_cast <std::chrono::microseconds> (m\_time).count();

}

private:

const std::string m\_name;

time\_point\_t m\_begin;

duration m\_time;

bool m\_running;

private:

std::ostream& m\_stream;

};

int main()

{

const int n = 10000;

std::array <int, n> array;

std::vector<int> vector(n);

std::deque <int> deque(n);

std::list <int> list;

std::forward\_list <int> forward\_list;

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

{

vector[i] = n - i;

array[i] = n - i;

deque[i] = n - i;

list.push\_back(n - i);

forward\_list.insert\_after(forward\_list.before\_begin(), i + 1);

}

int a;

{

Timer timer\_array("Timer for array");

std::sort(array.begin(), array.end());

a = timer\_array.Calculus();

}

int b;

{

Timer timer\_list("Timer for list");

list.sort();

b = timer\_list.Calculus();

}

int c;

{

Timer timer\_forward\_list("Timer for forward\_list");

forward\_list.sort();

c = timer\_forward\_list.Calculus();

}

int d;

{

Timer timer\_vector("Timer for vector");

std::sort(vector.begin(), vector.end());

d = timer\_vector.Calculus();

}

int e;

{

Timer timer\_deque("Timer for deque");

std::sort(deque.begin(), deque.end());

c = timer\_deque.Calculus();

}

/\*struct Timer\_result {

int a, b, c, d, e;

};

auto с\_time = [](Timer\_result& t1, Timer\_result& t2) {

return t1.m\_time < t2.m\_time;

};

std::sort(Timer\_result.begin(), Timer\_result.end(), auto c\_time);

}\*/