1. Создание вектора (vector)

#include <iostream>

#include <vector> //подключаем библиотеку для работы с векторами

using namespace std;

int main()

{

//vector <тип данных который хранит> имя вектора;

vector<int> v1;

v1.push\_back(5); //добавить в конец вектора

cout << "vector size: " << v1.size() << endl; //кол-во элементов

cout << "last element: " << v1.back() << endl; //последний элемент

//v1.pop\_back() //удаляет последний элемент вектора

v1.clear(); //удаление всех элементов из вектора

cout << "vector size: "<< v1.size() << endl;

system("pause");

return 0;

}

1. Вектор insert(), begin(), end()

#include <iostream>

#include <vector> //подключаем библиотеку для работы с векторами

using namespace std;

int main()

{

vector<int> v1{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

//begin() итератор, который указывает на начало вектора

//end() итератор, который указывает на конец вектора

//v1.insert работает только с итераторами

//первый параметр “куда вставляем”, второй параметр “что вставляем”

v1.insert(v1.begin(), 15); //вставка в 0-й элемент числа 15

v1.insert(v1.begin() + 3, 16); //вставка в 3-й элемент

v1.insert(v1.end(), 17); //вставка в последний элемент

v1.insert(v1.end() - 1, 18); //вставка в предпоследний элемент

//v1.insert(v1.begin() - 1, 15); //ошибка! мы не можем уходить в отрицательные индексы

//v1.insert(v1.end() + 1, 19); //ошибка! в таком случае лучше использовать push\_back()

for (auto elem : v1) { //range based for

cout << elem << endl;

}

for (auto p = v1.begin(); p != v1.end(); p++) { //как раскрывается range based for

auto elem = \*p;

cout << elem << endl;

}

for (int i=0; i < v1.size(); i++) { //как мы привыкли делать

cout << v1[i] << endl;

}

system("pause");

return 0;

}

1. Обращение к элементам вектора

#include <iostream>

#include <vector> //подключаем библиотеку для работы с векторами

using namespace std;

int main()

{

vector<int> v1{ 15, 2, -4, 56, 9};

cout << v1.at(0) << endl;

cout << v1[0] << endl;

//cout << v1.at(5) << endl; //выбрасывает исключение при выходе за границы

//cout << v1[5] << endl; //при выходе за границы может вызвать неопределенное поведение

//v1.begin() это итератор, который работает похожим образом на указатель массива

cout << \*v1.begin() << endl;

cout << \*(v1.begin() + 1) << endl;

cout << \*(v1.begin() + 2) << endl;

cout << \*(v1.begin() + 3) << endl;

cout << \*(v1.begin() + 4) << endl;

system("pause");

return 0;

}

1. Создание и работа со стеком (stack)

#include<iostream>

#include<stack> //библиотека для стеков

using namespace std;

int main()

{

stack<int> s; //стек, который хранит целые числа

s.push(1); //первый элемент в стеке

s.push(2); //второй элемент в стеке

cout << s.top() << endl; //у нас есть доступ только к последнему элементу

s.pop(); //удаляем последний добавленный

cout << s.top() << endl;

s.pop();

cout << boolalpha << s.empty() << endl; //пустой ли стек?

return 0;

}

1. Создание очереди (queue)

#include<iostream>

#include<queue> //библиотека для очередей

using namespace std;

int main()

{

queue<int> q; //очередь, которая хранит целые числа

q.push(1); //первый элемент в очереди

q.push(2); //второй элемент в очереди

q.push(3); //третий элемент в очереди

cout << q.front() << endl; //у нас есть к первому элементу

cout << q.back() << endl; //и к последнему

//элементы в середине нам не получить пока не удалим эти элементы

q.pop(); // важно! С помощью pop() мы удаляем ПЕРВЫЙ

cout << q.front() << endl; //первый вошедший - вышел раньше из очереди

cout << boolalpha << q.empty() << endl; //пустая ли очередь?

return 0;

}

1. Создание дека (deque)

#include<iostream>

#include<deque> //библиотека для деков

using namespace std;

int main()

{

deque<int> d; //дек, который хранит целые числа

d.push\_back(1); //первый элемент

d.push\_front(2); //второй элемент вставляем в начало

d.push\_back(3); //третий элемент вставляем в конец

cout << d.front() << endl; //у нас есть к первому элементу

cout << d.back() << endl; //и к последнему

//элементы в середине нам не получить пока не удалим эти элементы

d.pop\_back(); //удаляем последний

d.pop\_front(); //удаляем первый

cout << d.front() << endl; //остается то что было в середине

cout << boolalpha << d.empty() << endl; //пустой ли дек?

return 0;

}

1. Создание двусвязного списка (list)

#include<iostream>

#include<list> //библиотека для двусвязных списков

using namespace std;

int main()

{

list<int> l; //список, который хранит целые числа

l.push\_back(1); //первый элемент

l.push\_front(2); //второй элемент вставляем в начало

l.insert(++l.begin(), 3); //третий элемент вставляем в первый индекс

for (auto elem : l) { //range based for

cout << elem << endl;

}

l.erase(++l.begin()); //удаляем конкретный элемент через итератор

l.pop\_back(); //удаляем последний

l.pop\_front(); //удаляем первый

cout << boolalpha << l.empty() << endl; //пустой ли список?

return 0;

}

1. Создание словаря (map)

#include<iostream>

#include<string>

#include<map> //библиотека для словарей

using namespace std;

int main()

{

map<int, string> m; //словарь, который хранит ключ: число (и) значение: строка

m[392918] = "Maria";

m[392918] = "Ivan"; //можно менять значение по ключу

m[392917] = "Ivan"; //значения могут повторяться, если ключи разные

//m[125522] = "Anna";

m.insert(make\_pair(125522, "Anna")); //еще один способ добавить пару ключ-значение

m[666666] = "Mayvka";

m[1] = "Power";

//словари хранятся осторированными по ключам

for (auto elem : m) {

cout << "Key " << elem.first << ", value " << elem.second << endl;

}

auto iter = m.find(392917); //поиск итератора по ключу

m.erase(iter); //удаление

return 0;

}

1. Создание вектора

#include<iostream>

#include<set> //библиотека для множеств

using namespace std;

int main()

{

set<int> s; //множество, которое хранит целые числа

s.insert(3);

s.insert(1);

s.insert(3);

s.insert(2);

s.insert(3);

s.insert(3);

s.insert(2);

s.insert(4);

s.erase(4); //удалить значение 3

for (auto elem : s) { //range based for

cout << elem << endl;

}

cout <<"first elem >= 2: "<< \*s.lower\_bound(2) << endl;

cout <<"first elem > 2: " << \*s.upper\_bound(2) << endl;

return 0;

}