КОНТЕЙНЕРЫ

ARRAY

#include <array>

Объект типа array представляет из себя массив последовательных элементов **фиксированного** размера.

В контейнер array нельзя добавлять новые элементы и удалять уже имеющиеся.

```
template < class T,  std::size_t N > class array;
```

T – тип данных элементов контейнера.

N – количество элементов массива.

объявление

```
array<int, 4> myarray; // состоит из 4 чисел
```

объявление и инициализация

```
array<int, 6> numbers = { 1, 2, 3, 4, 5, 6 };
array<int, 6> numb1 = numbers;
array<string, 3> nms = { "Tom", “Bob", "Kate"};
```

определение

```
myarray = { 0, 1, 2, 3 };
myarray = { 8, 6 }; // 2 и 3 элемент  = 0
```

Основные функции для array:

size( ): возвращает размер массива;

at(i): возвращает элемент по индексу **i** и осуществляет контроль выхода за границы массива (*mas[i] ≈ mas.at(i)*) ;

front( ): возвращает первый элемент;

back( ): возвращает последний элемент;

fill(n): присваивает всем элементам контейнера значение n;

empty( ): возвращает булевый результат, проверки есть ли элементы в контейнере.

```
array<string, 6> nms;
nms.fill("bob");// 6 элементов “bob”

array<string, 3> nms ={"Tom", “Bob", "Kate"};

cout << nms.back( ) ; // "Kate"
cout << nms.front( ); //"Tom"
nms[2]==nms.at(2)
```

Итераторы: begin, end, rbegin, rend.

```
array<int, 20> mas;
    auto first = mas.begin();
    auto last = mas.end();
    while (first != last) {
        cout << *first << " ";
    first++;
    }
```

Массив в WinForms

Массив объектов типа String:

```
array <String^> ^stroks = gcnew array<String^>(3);
```

Массив объектов типа Label:

```
array <Label^> ^lb = gcnew array<Label^>(3);
```

Объявление и инициализация:

```
array <Label^> ^lb =gcnew array<Label^>(3){ label2,
label3, label4 };
```

```
array <Label^> ^lb = { label2, label3, label4 };
```

```
String ^st1, ^st2;
array <String^> ^stroks = gcnew array <String^>(3);
int n1, n2=99, n3=96;
array <Label^> ^lb = gcnew array<Label^>(3);
st1 = textBox1->Text;
stroks = st1->Split(' ');
for (int i = 0; i < 3; i++) {
    lb[i] = gcnew Label();
    lb[i]->Name = "labl" + (i + 1);
    lb[i]->Text = textUp (stroks[i]);
    lb[i]->Location = System::Drawing::Point(n2, n3);
    n2 = n2 + 50; n3 = n3 + 40;
    this->Controls->Add(lb[i]);
}

 String^  textUp(String ^st) {
     st = st->Substring(0, 1)->ToUpper() + st->Substring(1);
     return st; }
```

LIST

#include <list>

Класс list реализован в STL в виде двусвязного списка, каждый узел которого содержит ссылки на последующий и предыдущий элементы.

Список не предоставляет произвольного доступа к своим элементам.

Список поддерживает конструкторы, операцию присваивания, функцию копирования, операции сравнения и итераторы, аналогичные векторам.

Создание списка

```
// пустой список
      std::list<int> list1;
// список list2 состоит из 5 чисел, равных 0
      std::list<int> list2(5);
// список list3 состоит из 5 чисел, равных 2
      std::list<int> list3(5, 2);
// список состоит из чисел 1, 2, 4, 5
      std::list<int> list4{ 1, 2, 4, 5 };
      std::list<int> list5 = { 1, 2, 3, 5 };
// список копия списка list4
      std::list<int> list6(list4);
      std::list<int> list7 = list4;
```

Получение элементов

front( ) возвращает первый элемент.

back( ) возвращает последний элемент.

empty( ) проверяет пустой ли список.

Размер списка

size( ) возвращает размер списка.

resize(n): оставляет в списке n первых элементов. Если список содержит больше элементов, то он усекается до первых n элементов. Если размер списка меньше n, то добавляются недостающие элементы и инициализируются значением по умолчанию.

resize(n, value): оставляет в списке n первых элементов. Если размер списка меньше n, то добавляются недостающие элементы со значением value.

Добавление элементов

push_back(val): добавляет значение val в конец списка

push_front(val): добавляет значение val в начало списка

insert(pos, val): вставляет элемент val на позицию, на которую указывает итератор pos. Возвращает итератор на добавленный элемент

insert(pos, n, val): вставляет n элементов val начиная с позиции, на которую указывает итератор pos. Возвращает итератор на первый добавленный элемент. Если n = 0, то возвращается итератор pos.

insert(pos, begin, end): вставляет начиная с позиции, на которую указывает итератор pos, элементы из другого контейнера из диапазона между итераторами begin и end. Возвращает итератор на первый добавленный элемент. Если между итераторами begin и end нет элементов, то возвращается итератор pos.

insert(pos, values): вставляет список значений values начиная с позиции, на которую указывает итератор pos. Возвращает итератор на первый добавленный элемент. Если values не содержит элементов, то возвращается итератор pos.

Удаление элементов

clear(p): удаляет все элементы

pop_back(): удаляет последний элемент

pop_front(): удаляет первый элемент

erase(p): удаляет элемент, на который указывает итератор p. Возвращает итератор на элемент, следующий после удаленного, или на конец контейнера, если удален последний элемент

erase(begin, end): удаляет элементы из диапазона, на начало и конец которого указывают итераторы begin и end.

Изменение элементов списка

assign(il): заменяет содержимое контейнера элементами из списка инициализации il.

assign(n, value): заменяет содержимое контейнера n элементами, которые имеют значение value.

assign(begin, end): заменяет содержимое контейнера элементами из диапазона, на начало и конец которого указывают итераторы begin и end.

swap() обменивает значениями два списка.

Пример.
Даны натуральное число n, последовательность чисел x_1, …, x_n.
Получить последовательность x_1-x_n, x_2-x_n, …, x_{n-1}-x_n.

```
int nn;
cout << "\nвведите n=  \n";
cin >> nn;
list <int> x, y;
for (int i = 0; i < nn; i++)
      x.push_back(rand() % 11);
for (int n : x)
      std::cout <<setw(4)<< n ;
std::cout << std::endl;
for (auto it =  x.begin(); it != x.end(); it++)
      y.push_back(*it - x.back());
for (int n : y)
      std::cout << setw(4) << n;
```

DEQUE

#include <deque>

Deque это двусторонняя очередь, реализованная в виде динамического массива, который может расти с обоих концов.

Допускаются две операции, изменяющие ее размер — добавление и выборка элемента в конец и из начала.

Очередь является адаптером, который можно реализовать на основе двусторонней очереди или списка.

Получение элементов очереди

[index]: получение элемента по индексу

at(index): возвращает элемент по индексу

front(): возвращает первый элемент

back(): возвращает последний элемент

```
 std::deque<int> deq;
    for (int i = 0; i < 4; ++i)
    {
deq.push_back(i); // вставляем числа в конец массива
deq.push_front(10 - i); // вставляем числа в начало массива
    }

 // вывод массива на экран
 for (int index = 0; index < deq.size(); ++index)
        std::cout << deq[index] << ' ';

    std::cout <<deq[3]<< '\n';

deq.pop_back(); // удаляем число в конце массива
deq.pop_front(); // удаляем число в начале массива


    std::cout << deq.at(3)<<'\n';
```

STACK

#include <stack>

stack<T> - контейнер, который работает по принципу LIFO (last-in first-out или "последний вошел — первым вышел") — первым всегда извлекается последний добавленный элемент.

Стек можно сравнить со стопкой предметов, например, стопкой тарелок - тарелки добавляются сверху, каждая последующая тарелка кладется поверх предыдущей. А если надо взять тарелку, то сначала берется та, которая в самом верху (которую положили самой последней).

Определение пустого стека

stack<string> stack; // пустой стек строк

size() — возвращает количество элементов в стеке.

empty() — проверяет стек на наличие элементов (если стек пуст, то возвращается true).

push() — добавление в стек элемента

top() — получение элемента.

Можно получить только самый верхний элемент стека.

pop() — удаление элемента.

Удаление производится в порядке, обратном добавлению.

Стек можно инициализировать другим стеком или деком (двусторонней очередью).

```
deque<string> users{"Tom", "Sam", "Bob"};
stack<string> stack {users};
 // удаляем элементы из стека
    while (!stack.empty()) {
        std::cout << stack.top() << std::endl;
        stack.pop();
    }
```