

# Языки программирования

## Standard Template Library, часть 1

# Стандартная библиотека

- **Потоковые классы**
- **Строковые классы**
- **Контейнерные классы** – классы для хранения данных, реализующие наиболее распространенные структуры данных, например, очередь, стек, линейный список.
- **Алгоритмы** – классы, реализующие процедуры для обработки данных контейнеров различными способами
- **Итераторы** – обобщение указателей, они ссылаются на элементы контейнера. Итераторы связывают алгоритмы с контейнерами.
- **Математические классы** – поддерживают эффективную обработку массивов с плавающей точкой, работу с комплексными числами
- **Диагностические классы** – обеспечивают динамическую идентификацию типов и объектно-ориентированную обработку ошибок

STL

# Достоинства и недостатки STL

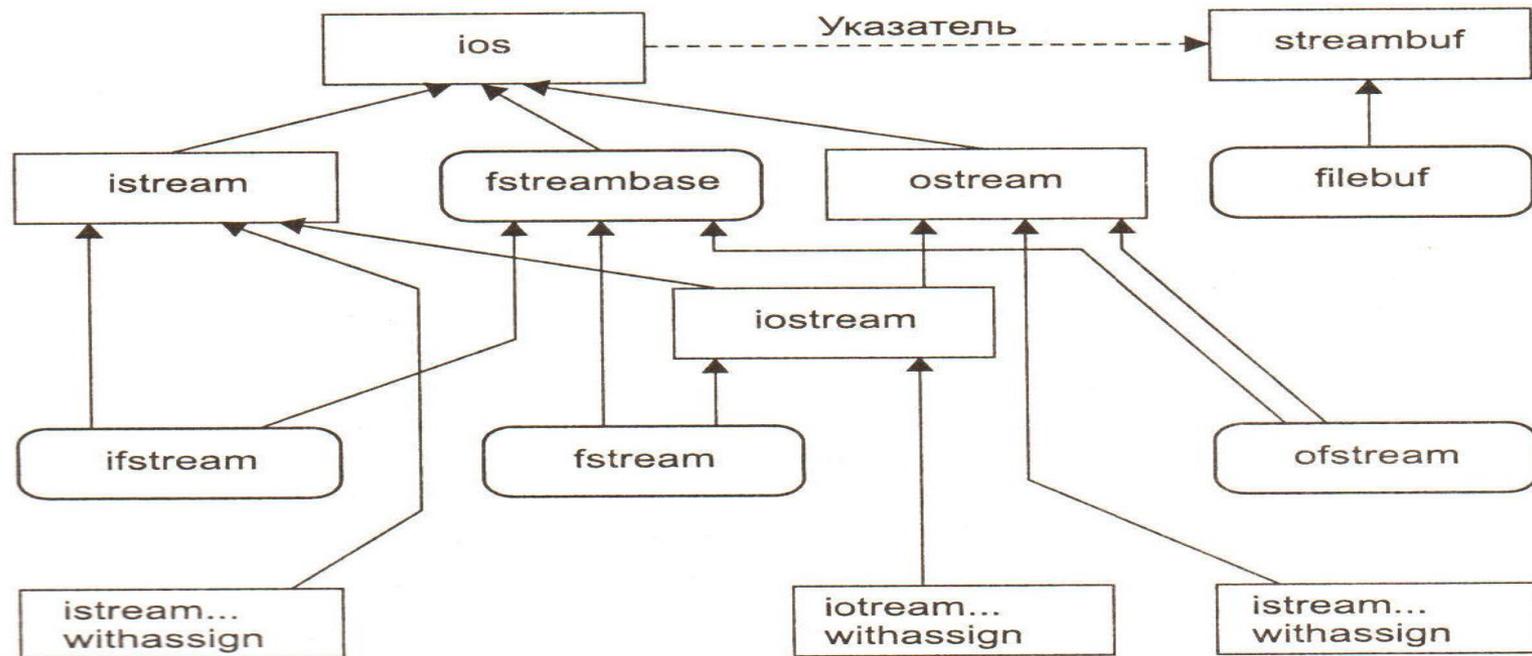
## Достоинства

- Повышение надежности программ, а также их переносимости и универсальности
- Уменьшение объема исходного кода
- Уменьшение времени работы программиста

## Недостатки

- Снижение быстродействия программы
- Необходимость изучения STL

# Потоковые классы



 IOSTREAM

 FSTREAM

# Потоковые классы

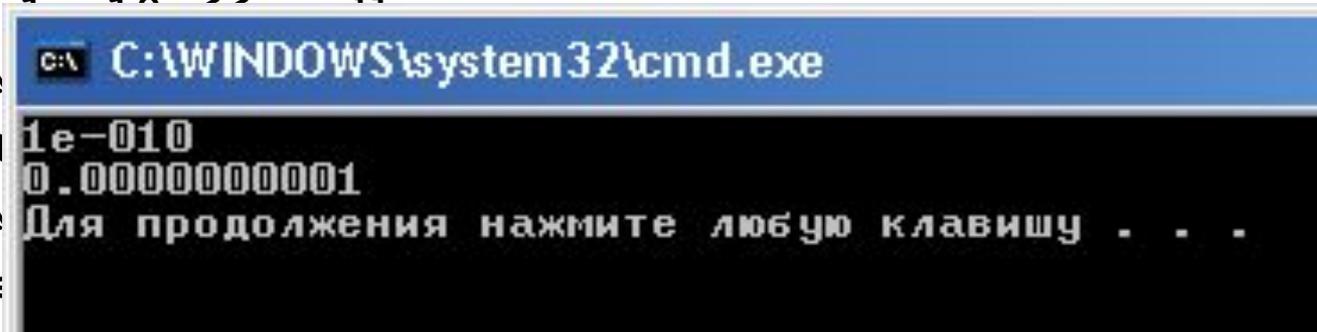
- **Флаг форматирования** – отдельные биты в поле `x_flags` типа `long` класса `ios`. Они работают переключателями, определяющими различные форматы и способы ввода/вывода. Задание флагов осуществляется методом `setf(flags)`, а получение `flags()`;
- **Манипуляторы** – это инструкции форматирования, которые вставляются прямо в поток.

# Флаги форматирования

Флаг	Значение
skipws	Пропуск пробелов при вводе
left	Выравнивание по левому краю
right	Выравнивание по правому краю
internal	Заполнение между знаком или основанием числа и самим числом
dec	Перевод в десятичную форму
oct	Перевод в восьмеричную форму
hex	Перевод в шестнадцатеричную форму
boolalpha	Перевод логического 0 и 1 соответственно в «false» и «true»
showbase	Выводить индикатор основания системы счисления (0 для восьмеричной, 0x для шестнадцатеричной)
showpoint	Показывать десятичную точку при выводе
uppercase	Переводить в верхний регистр буквы X, E и буквы шестнадцатеричной системы счисления (ABCDEF) (по умолчанию — в нижнем регистре)
showpos	Показывать «+» перед положительными целыми числами
scientific	Экспоненциальный вывод чисел с плавающей запятой
fixed	Фиксированный вывод чисел с плавающей запятой
unitbuf	Сброс потоков после вставки
stdio	сброс stdout, stderr после вставки

# Пример использования флагов форматирования

```
//Сохраняем текущие значения флагов
long prev_flags = cout.flags();
//Выводим вещественное число, на экране - экспоненциальная
форма
cout << 1e-10 << endl;
//Задаем точку
cout.setf(cout.flags() | cout.decimal);
//Задаем точность
cout.precision(10);
//Выводим то же число и видим все десять знаков
cout << 1e-10 << endl;
//Восстанавливаем флаги
cout.setf(prev_flags);
```



точкой

# Манипуляторы

Манипулятор	Назначение
ws	Включает пропуск пробелов при вводе
dec	Перевод в десятичную форму
oct	Перевод в восьмеричную форму
hex	Перевод в шестнадцатеричную форму
endl	Вставка разделителя строк и очистка выходного потока
ends	Вставка символа отсутствия информации для окончания выходной строки
flush	Очистка выходного потока
lock	Закрывает дескриптор файла
unlock	Открывает дескриптор файла

# Пример использования манипуляторов

```
#include <iomanip>
```

```
.....
```

```
cout << hex << 31 << " " << oct << 31 << "  
" << dec << 31 << endl;  
cout << setw(10) << 666 << endl;
```

# Контейнеры

- **Последовательные контейнеры** – обеспечивают хранение конечного количества однотипных величин в виде непрерывной памяти
  - Векторы – `vector`
  - Двусторонние очереди – `deque`
  - Линейные двусвязные списки – `list`
  - Их адаптеры:
    - Стеки – `stack`
    - Очереди – `queue`
    - Очереди с приоритетами – `priority_queue`
- **Ассоциативные контейнеры** – обеспечивают быстрый доступ к данным по ключу. Построены на базе сбалансированных деревьев
  - Словари – `map`
  - Словари с дубликатами – `multimap`
  - Множества – `set`
  - Мультимножества (множества с дубликатами) – `multiset`
  - Битовые множества – `bitset`

# Характеристика контейнеров

- Позволяют хранить **данные произвольных типов**
- Обеспечивают **стандартизированный интерфейс** для выполнения операций над хранимыми данными
- **Стандартизирован только интерфейс**, возможны **разные реализации** контейнерных классов разными разработчиками STL/сред программирования

## Типы, определения которых содержатся в каждом контейнерном классе STL

Поле	Пояснение
value_type	Тип элемента контейнера
size_type	Тип индексов, счетчиков элементов и т. д.
iterator	Итератор
const_iterator	Константный итератор
reverse_iterator	Обратный итератор
const_reverse_iterator	Константный обратный итератор
reference	Ссылка на элемент
const_reference	Константная ссылка на элемент
key_type	Тип ключа (для ассоциативных контейнеров)
key_compare	Тип критерия сравнения (для ассоциативных контейнеров)

# Понятие итератора

Итератор является **аналогом указателя** на элемент. Используется для просмотра контейнера в прямом или обратном порядке.

## **Итератор умеет:**

- ссылаться на элемент контейнера (операция \*);
- переходить к следующему элементу контейнера (операция ++).

**Константные итераторы** необходимы для работы с элементами контейнера без возможности их изменения.

# Итераторы просмотра элементов контейнеров

Метод	Пояснение
<code>iterator begin(), const_iterator begin() const</code>	Указывают на первый элемент
<code>iterator end(), const_iterator end() const</code>	Указывают на элемент, следующий за последним
<code>reverse_iterator rbegin(), const_reverse_iterator rbegin() const</code>	Указывают на первый элемент в обратной последовательности
<code>reverse_iterator rend(), const_reverse_iterator rend() const</code>	Указывают на элемент, следующий за последним, в обратной последовательности

# Методы, позволяющие определить сведения о размере контейнера

<b>Метод</b>	<b>Пояснение</b>
size()	Число элементов
max_size()	Максимальный размер контейнера (порядка миллиарда элементов)
empty()	Булевская функция, показывающая, пуст ли контейнер

# Заголовочные файлы для работы с библиотекой STL

- `algorithm`
- `deque`
- `functional`
- `iterator`
- `list`
- `map`
- `memory`
- `numeric`
- `queue`
- `set`
- `stack`
- `utility`
- `vector`

# Характеристика общих операций для последовательных контейнеров

Операция	Метод	vector	deque	list
Вставка в начало	push_front	-	+	+
Удаление из начала	pop_front	-	+	+
Вставка в конец	push_back	+	+	+
Удаление из конца	pop_back	+	+	+
Вставка в произвольное место	insert	(+)	(+)	+
Удаление из произвольного места	erase	(+)	(+)	+
Произвольный доступ к элементу	[ ], at	+	+	-

“-” – операция в контейнере не реализована

“+” – операция выполняется за постоянное время, не зависящее от размера контейнера  $n$

“(+)” – операция выполняется за время, пропорциональное  $n$

# Векторы - vector

- Являются аналогами динамических массивов с возможностью изменения размера
- Эффективно реализуют операции произвольного доступа к элементам, добавления в конец и удаления с конца
- Неэффективно реализуют операции вставки и удаления в начало или середину

# Пример использования вектора

```
int n, x;
vector<int> v; //объявляем вектор целых чисел
cout << "n = ";
cin >> n;      //вводим количество элементов
for (int i = 0; i < n; i++) { //вводим элементы
```

```
cin >>
```

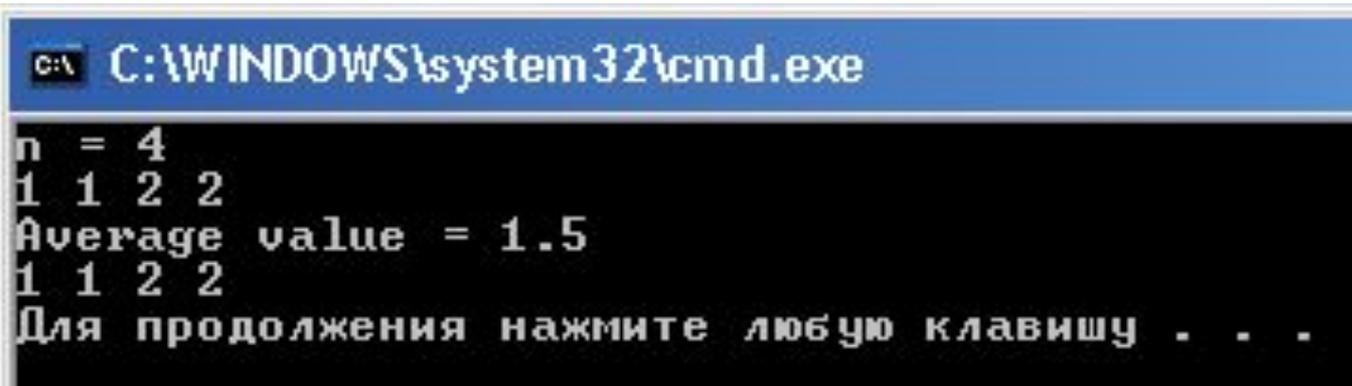
```
конец
```

```
}
```

```
//Вычисл
```

```
double a
```

```
for (int
```



```
C:\WINDOWS\system32\cmd.exe
n = 4
1 1 2 2
Average value = 1.5
1 1 2 2
Для продолжения нажмите любую клавишу . . .
```

Т В

```
    avg += v[i]; //обращение к элементам вектора по индексу
avg /= v.size();
cout << "Average value = " << avg << endl;
//Печать вектора на экран с использованием итератора I
for (vector<int>::iterator i = v.begin(); i != v.end(); i++)
{
    cout << *i << " ";
```

# Конструкторы создания вектора

- Конструктор по умолчанию  
`explicit vector();`
- Конструктор, создающий вектор размера `n` и заполняющий его значением `value`. Если `value` не указано, то для встроенных типов происходит инициализация нулем, для пользовательских – вызывается конструктор по умолчанию  
`explicit vector(size_type n, const T& value = T());`
- Конструктор копирования значений из диапазона итераторов `[first, last)` другого контейнерного класса  
`vector(InputIter first, InputIter last);`
- Конструктор копирования  
`vector(const vector<T>& x);`

`explicit` запрещает неявное преобразование типов при объявлении векторов с инициализацией

# Пример использования конструкторов (1)

```
vector<int> v1; //используется первый конструктор
vector<double> v2(10, 0.5); //второй конструктор
v2[0] = 666.666;
vector<double> v3(v2.begin(), v2.begin() + 5); //третий конст-р
vector<double> v4 = v2; //четвертый конструктор
```

```
cout << "v1 :";
for (vector<int>::iterator i = v1.begin(); i != v1.end(); i++)
    cout << *i << " ";
cout << endl;
cout << "v2 :";
for (vector<double>::iterator i = v2.begin(); i != v2.end(); i++)
    cout << *i << " ";
cout << endl;
cout << "v3 :";
for (vector<double>::iterator i = v3.begin(); i != v3.end(); i++)
    cout << *i << " ";
cout << endl;
cout << "v4 :";
for (vector<double>::iterator i = v4.begin(); i != v4.end(); i++)
    cout << *i << " ";
cout << endl;
cout << "Для продолжения нажмите любую клавишу . . . _";
```

```
for (vector<double>::iterator i = v3.begin(); i != v3.end(); i++)
    cout << *i << " ";
cout << endl;
cout << "v4 :";
for (vector<double>::iterator i = v4.begin(); i != v4.end(); i++)
    cout << *i << " ";
cout << endl;
```

## Пример использования конструкторов (2)

```
vector<int> v1;  
vector<double> v2(10, 0.5);  
v2[0] = 666.666;  
vector<double> v3(v2.begin(), v2.begin() + 5);  
vector<double> v4 = v2;
```

```
cout << "v1 :" << endl << v1;  
cout << "v2 :" << endl << v2;  
cout << "v3 :" << endl << v3;  
cout << "v4 :" << endl << v4;
```

//Шаблонная функция печати элементов вектора

```
template <class T>
```

```
ostream& operator<< (ostream& out, const vector<T>& v)
```

```
{
```

```
for (vector<T>::const_iterator i = v.begin(), i != v.end(); i++)
```

```
    out << *i << " ";
```

```
cout << endl;
```

Далее в примерах будет использоваться  
данная шаблонная функция печати вектора

# Присваивание векторов

- С помощью операции присваивания

```
vector<T>& operator= (const vector<T>& x);
```

- С помощью методов

```
void assign(size_type n, const T& value);
```

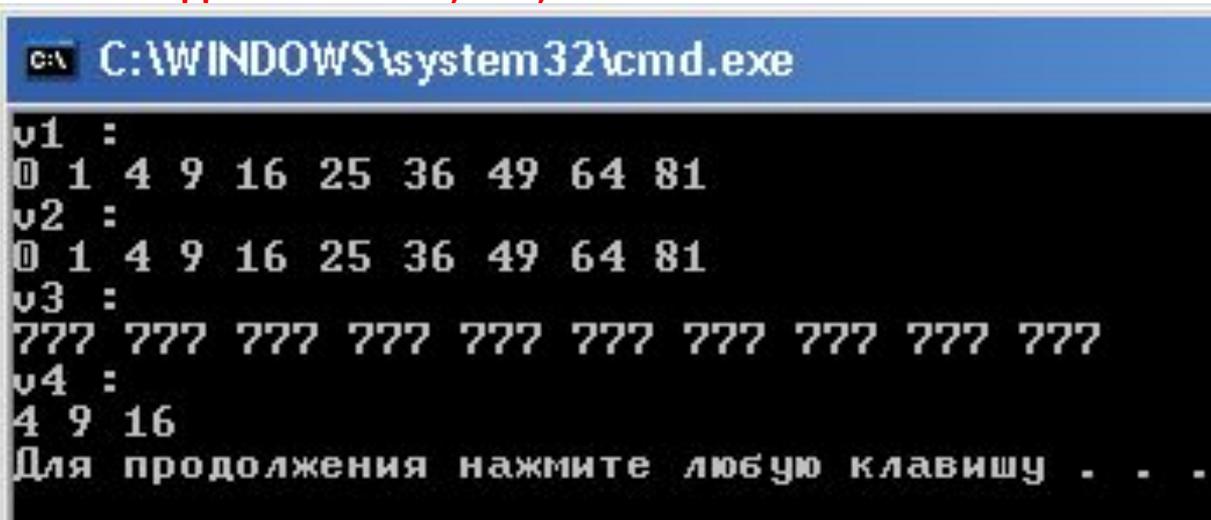
```
void assign(InputIter first, InputIter last);
```

Использование методов аналогично соответствующим конструкторам.

# Пример использования присваивания

```
vector<int> v1, v2, v3, v4;  
for (int i = 0; i < 10; i++)  
    v1.push_back(i * i);  
v2 = v1; //Вызов операции =  
//Присвоение v3 десяти значений 777  
v3.assign(10, 777);  
//Присвоение v4 значений из диапазона итераторов [v1.begin() + 2,  
    v1.begin() + 5),  
//т.е. значений с индексами 2, 3, 4  
v4.assign(v1.begin() + 2, v1.begin() + 5);
```

```
cout << "v1 :"  
cout << "v2 :"  
cout << "v3 :"  
cout << "v4 :"
```



```
C:\WINDOWS\system32\cmd.exe  
v1 :  
0 1 4 9 16 25 36 49 64 81  
v2 :  
0 1 4 9 16 25 36 49 64 81  
v3 :  
777 777 777 777 777 777 777 777 777 777  
v4 :  
4 9 16  
Для продолжения нажмите любую клавишу . . .
```

# Доступ к элементам вектора

- С помощью операции индексирования `[]`:  
`reference operator[] (size_type i);`

- По индексу с помощью метода:  
`reference at(size_type i);`

При выходе за границу вектора генерируется исключение `out_of_range`

- Обращение к первому элементу вектора  
`reference front();`
- Обращение к последнему элементу вектора  
`reference back();`

# Знаки операций, определенные для векторов

Знаки операций	Назначение
=	Присвоение
<, >, <=, >=, ==, !=	Лексикографическое сравнение векторов

## Получение размера памяти и ее резервирование

- Получение размера занимаемой памяти  
`size_type capacity() const;`
- Ручное резервирование памяти под *n* элементов вектора. Память выделяется, но сами элементы не добавляются  
`void reserve(size_type n);`

Память под вектор выделяется динамически блоками по 256 или 1024 элемента. Перераспределение памяти происходит динамически при превышении текущего объема. После перераспределения любые итераторы, указывающие на вектор становятся недействительными. Функция `reserve` позволяет предварительно зарезервировать память под то, количество элементов, которые могут оказаться в векторе в ходе выполнения программы.

# Замечание

Если заранее известен размер вектора, то чтобы сократить накладные расходы по работе с памятью, постарайтесь зарезервировать память

# Пример использования резервирования

```
int n, x;
vector<int> v; //объявляем вектор целых чисел
cout << "n = ";
cin >> n;      //вводим количество элементов
v.reserve(n);
for (int i = 0; i < n; i++) { //вводим элементы
    cin >> x;v.push_back(x); //добавляем очередной элемент в
    конец вектора
}
//Вычисление среднего арифметического
double avg = 0.0;
for (int i = 0; i < v.size(); i++)
    avg += v[i]; //обращение к элементам вектора по индексу
avg /= v.size();
cout << "Average value = " << avg << endl;
//Печать вектора на экран с использованием итератора I
for (vector<int>::iterator i = v.begin(); i != v.end(); i++)
{
```

# Добавление в конец вектора, удаление из конца

- Добавление в конец вектора

```
void push_back(const& T value);
```

- Удаление из конца вектора

```
void pop_back();
```

# Методы вставки значений в вектор

- Вставка значения `value` в позицию, на которую указывает итератор `pos`, возвращаемое значение – итератор, указывающий на место вставки

```
iterator insert(iterator pos,  
                const T& value);
```

- Вставка в позицию `pos` вектора `n` одинаковых элементов со значением `value`

```
void insert(iterator pos, size_type n,  
            const T& value);
```

- Вставка в позицию `pos` вектора из элементов другого контейнера из диапазона итераторов `[first last)`

```
void insert(iterator pos,  
            InputIter first,  
            InputIter last);
```

# Пример вставки значений в вектор

```
vector<int> v, v1;  
for (int i = 0; i < 10; i++)  
    v.push_back(i);
```

```
cout << v;
```



```
C:\WINDOWS\system32\cmd.exe  
0 1 2 3 4 5 6 7 8 9  
0 777 1 2 3 4 5 6 7 8 9  
0 777 1 2 3 4 5 6 7 8 9 69 69 69 69 69  
0 1 4 9 16 0 777 1 2 3 4 5 6 7 8 9 69 69 69 69 69  
Для продолжения нажмите любую клавишу . . .
```

```
v.insert(v.end(), 5, 69);
```

```
cout << v;
```

```
for (int i = 0; i < 5; i++)  
    v1.push_back(i * i);
```

//Вставка в начало вектора v всех значений вектора v1

```
v.insert(v.begin(), v1.begin(), v1.end());
```

# Методы удаления элементов из вектора

- Удаление одного элемента. `pos` – итератор, указывающий на него. Возвращаемое значение – итератор, указывающий на элемент после удаленного

```
iterator erase(iterator pos);
```

- Удаление нескольких элементов из диапазона итераторов [`first`, `last`)

```
iterator erase(iterator first,  
                iterator last);
```

# Пример удаления значений из вектора

```
vector<int> v;
```

```
for (int i = 0; i < 10; i++)  
    v.push back(i);
```

```
cout << v;
```

```
//у;
```

```
//v
```

```
v.e:
```

```
cout << v;
```

```
//Удаление предпоследнего элемента
```

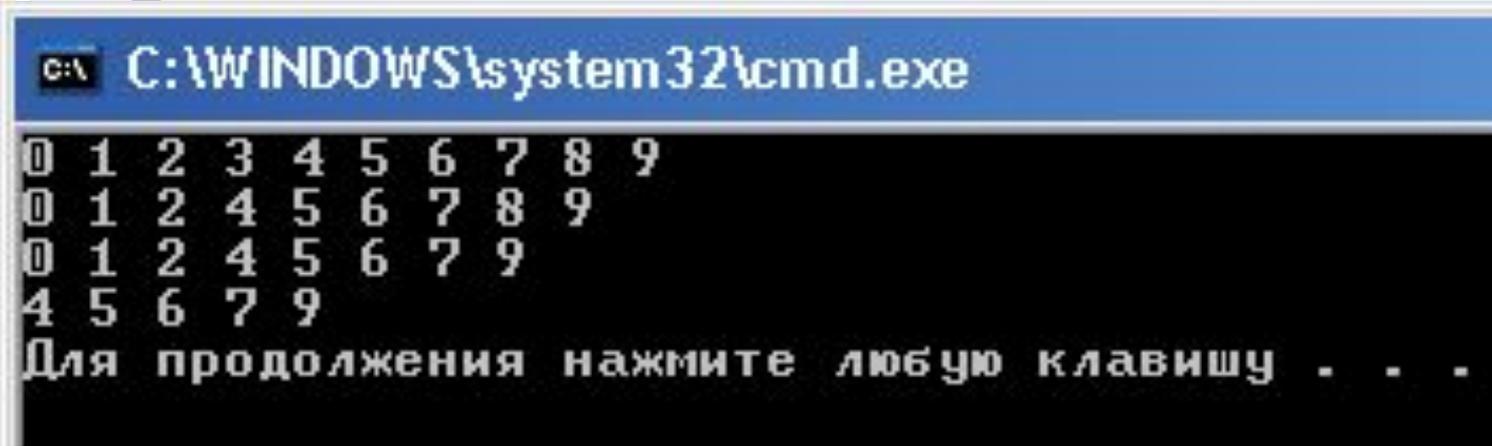
```
v.erase(v.end() - 2);
```

```
cout << v;
```

```
//Удаление первых трех элементов
```

```
v.erase(v.begin(), v.begin() + 3);
```

```
cout << v;
```



```
C:\WINDOWS\system32\cmd.exe  
cout << v;  
//у; 0 1 2 3 4 5 6 7 8 9  
//v 0 1 2 4 5 6 7 8 9  
v.e: 4 5 6 7 9  
cout << v;  
Для продолжения нажмите любую клавишу . . .
```

# Прочие методы работы с векторами

- Обмен местами векторов

```
void swap(vector<Y>& x) ;
```

- Очистка вектора

```
void clear() ;
```

- Изменение размера вектора,  $n$  – новый размер. Если  $n$  меньше текущего размера вектора, то удаляются последние элементы, если больше – добавляются новые элементы равные `value` в конец вектора

```
void resize(size_type n, T value = T()) ;
```

# Матрицы на основе векторов строятся как векторы векторов

```
int nr, nc;  
vector< vector<double> > mat;
```

```
cout << "nr = ";
```

```
cin >> nr;
```

```
cout << "nc = ";
```

```
cin >> nc;
```

```
mat.resize(nr);
```

```
for (int i = 0; i < nr; i++)
```

```
    mat[i].resize(nc);
```

```
cout << "Enter matrix :";
```

```
for (int i = 0; i < nr; i++)
```

```
    for (int j = 0; j < nc; j++)
```

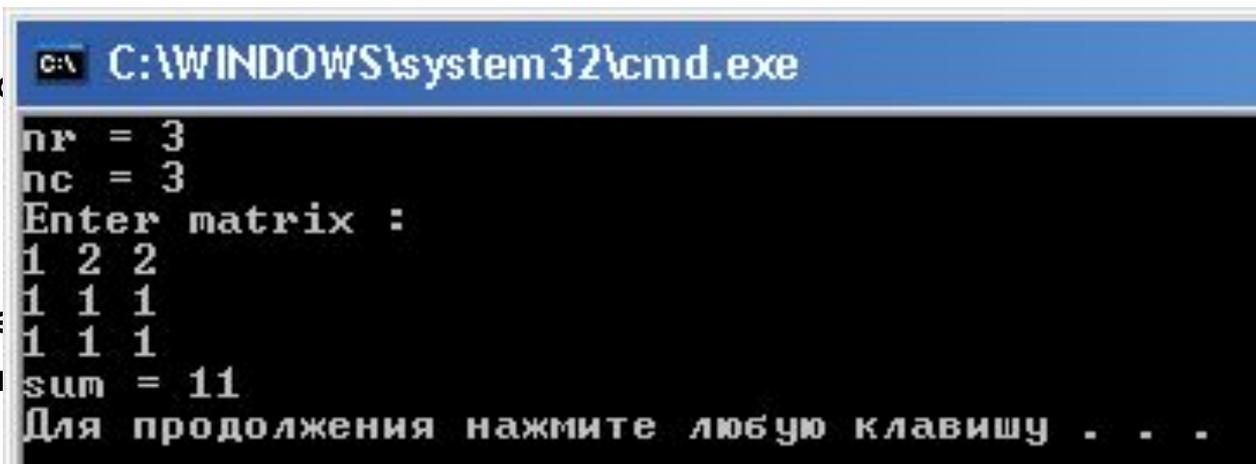
```
        cin >> mat[i][j];
```

```
double sum = 0.0;
```

```
for (int i = 0; i < nr; i++)
```

```
    for (int j = 0; j < nc; j++)
```

```
        sum += mat[i][j];
```



```
C:\WINDOWS\system32\cmd.exe  
nr = 3  
nc = 3  
Enter matrix :  
1 2 2  
1 1 1  
1 1 1  
sum = 11  
Для продолжения нажмите любую клавишу . . .
```