

Алгоритмизация и программирование

Лекция 3

Операторы перехода (jump)

goto

```
label:  
    /* код */  
goto label;
```

* label – метка, обычный идентификатор

** Внутри функции метка должна быть одна, при этом она видна всюду, в том числе и до своего объявления

break

```
while(true){  
    break;  
}
```

```
switch(2){  
    case 1: break;  
    case 2: break;  
}
```

* Оператор применяется только внутри **циклов** и **switch**

** Прерывает выполнение кода в ближайшем цикле или switch-е и передаёт управление следующему за прерванным стейтменту

continue

```
while(true){  
    // код 1  
    continue;  
    // код 2  
}
```

```
while(true){  
    {  
        // код 1  
        goto cont;  
        // код 2  
    }  
    cont:  
}
```

* Оператор применяется только внутри **циклов**

** Прерывает выполнение текущей итерации ближайшего цикла, т.е. весь код в блоке от continue и ниже пропускается

return

```
int foo(){  
    // код 1  
    return 10;  
    // код 2  
}
```

* Оператор применяется где угодно внутри **функции**

** Прерывает выполнение текущей функции и возвращает управление в точку вызова. Результат работы функции равен значению указанному после оператора.

Переменные и типы данных

Переменная

```
std::string name = "James Bond";
```

Значение ("James Bond")

Идентификатор/имя (name)
может быть 0 или больше

Переменная

Тип (std::string)

Адрес (0x7ffd7ca6b9a0)

Типы данных

Типы данных

Фундаментальные

Числа с плавающей
запятой

- float
- double
- long double

Целочисленные
(интегральные)

- bool
- char
- wchar_t
- char16_t
- char32_t
- signed и unsigned:
 - char
 - short int
 - int
 - long int
 - long long int

`std::nullptr_t`

`void`

Составные

Массивы

Структуры/Классы

Объединения

Перечисления

Указатели

Функции

Фундаментальные

int | целое число

Тип **int** — основной для хранения целых чисел. Размер не указан в стандарте и зависит от платформы.

Стандарт определяет только то, что все типы кратны char и:

$$\text{char} \leq \text{short int} \leq \text{int} \leq \text{long int} \leq \text{long long int}$$

type()	size()	min()	max()
char	1	-128	127
short int	2	-32768	32767
int	4	-2147483648	2147483647
long int	8	-9223372036854775808	9223372036854775807
long long int	8	-9223372036854775808	9223372036854775807

<https://wandbox.org/permlink/KM5rFtg19VLsq5Gj>

type()	size()	min()	max()
unsigned char	1	0	255
unsigned short int	2	0	65535
unsigned int	4	0	4294967295
unsigned long int	8	0	18446744073709551615
unsigned long long int	8	0	18446744073709551615

<https://wandbox.org/permlink/tuvN3T2kTsvCeOce>

signed и unsigned

Модификаторы применяемые к типам: `char`, `short int`, `int`, `long int`, `long long int`.

Если модификатор не указан, то по умолчанию тип `signed`. В переменных такого типа старший бит отвечает за хранение знака. Переменные способны хранить как положительные, так и отрицательные значения. При переполнении - `undefined behavior (UB)`.

Модификатор `unsigned` служит для объявления беззнаковых целых чисел. Все биты числа отвечают за хранение значения. При переполнении проблем нет.

Целое число с заданными размерами

Т.к. стандарт не определяет точных размеров основных целочисленных типов, могут понадобиться дополнительные целочисленные типы с известными размерами. Такие типы содержатся в заголовочном файле `<stdint.h>`

Types	
<code>int8_t</code> <code>int16_t</code> <code>int32_t</code> (optional) <code>int64_t</code>	signed integer type with width of exactly 8, 16, 32 and 64 bits respectively with no padding bits and using 2's complement for negative values (provided if and only if the implementation directly supports the type) (typedef)
<code>int_fast8_t</code> <code>int_fast16_t</code> <code>int_fast32_t</code> <code>int_fast64_t</code>	fastest signed integer type with width of at least 8, 16, 32 and 64 bits respectively (typedef)
<code>int_least8_t</code> <code>int_least16_t</code> <code>int_least32_t</code> <code>int_least64_t</code>	smallest signed integer type with width of at least 8, 16, 32 and 64 bits respectively (typedef)
<code>intmax_t</code>	maximum-width signed integer type (typedef)
<code>intptr_t</code> (optional)	signed integer type capable of holding a pointer to <code>void</code> (typedef)
<code>uint8_t</code> <code>uint16_t</code> <code>uint32_t</code> (optional) <code>uint64_t</code>	unsigned integer type with width of exactly 8, 16, 32 and 64 bits respectively (provided if and only if the implementation directly supports the type) (typedef)
<code>uint_fast8_t</code> <code>uint_fast16_t</code> <code>uint_fast32_t</code> <code>uint_fast64_t</code>	fastest unsigned integer type with width of at least 8, 16, 32 and 64 bits respectively (typedef)
<code>uint_least8_t</code> <code>uint_least16_t</code> <code>uint_least32_t</code> <code>uint_least64_t</code>	smallest unsigned integer type with width of at least 8, 16, 32 and 64 bits respectively (typedef)
<code>uintmax_t</code>	maximum-width unsigned integer type (typedef)
<code>uintptr_t</code> (optional)	unsigned integer type capable of holding a pointer to <code>void</code> (typedef)

assert | утверждение

Чтобы проверить некоторое утверждение, в том числе и убедиться в достаточном размере типа можно воспользоваться конструкцией `assert`.

В случае, если утверждение указанное в теле `assert` не выполняется, приложение упадёт с ошибкой.

Есть два вида таких проверок:

- `assert` – проверка на этапе исполнения программы;
- `static_assert` – проверка на этапе компиляции.

<https://wandbox.org/permlink/5zZTKfFC2RdhwIID>

bool | Логический тип

Интегральный тип `bool` предназначен для хранения логических значений, и имеет 2 литерала:

`false` — ложь и

`true` — истина.

```
bool b1 = false;
```

```
bool b2 = 10 > 5;
```

```
bool b3 = b1 && b2;
```

Любое число кроме нуля и не нулевой указатель преобразуются в `true`.

Ноль и нулевой указатель преобразуются в `false`.

Числа с плавающей запятой

Обычно для хранения дробных значений в C++ используется представление с плавающей запятой вида:

$$x = a \cdot 10^n$$

где a – мантисса, n – порядок. Стандарт: IEEE 754.

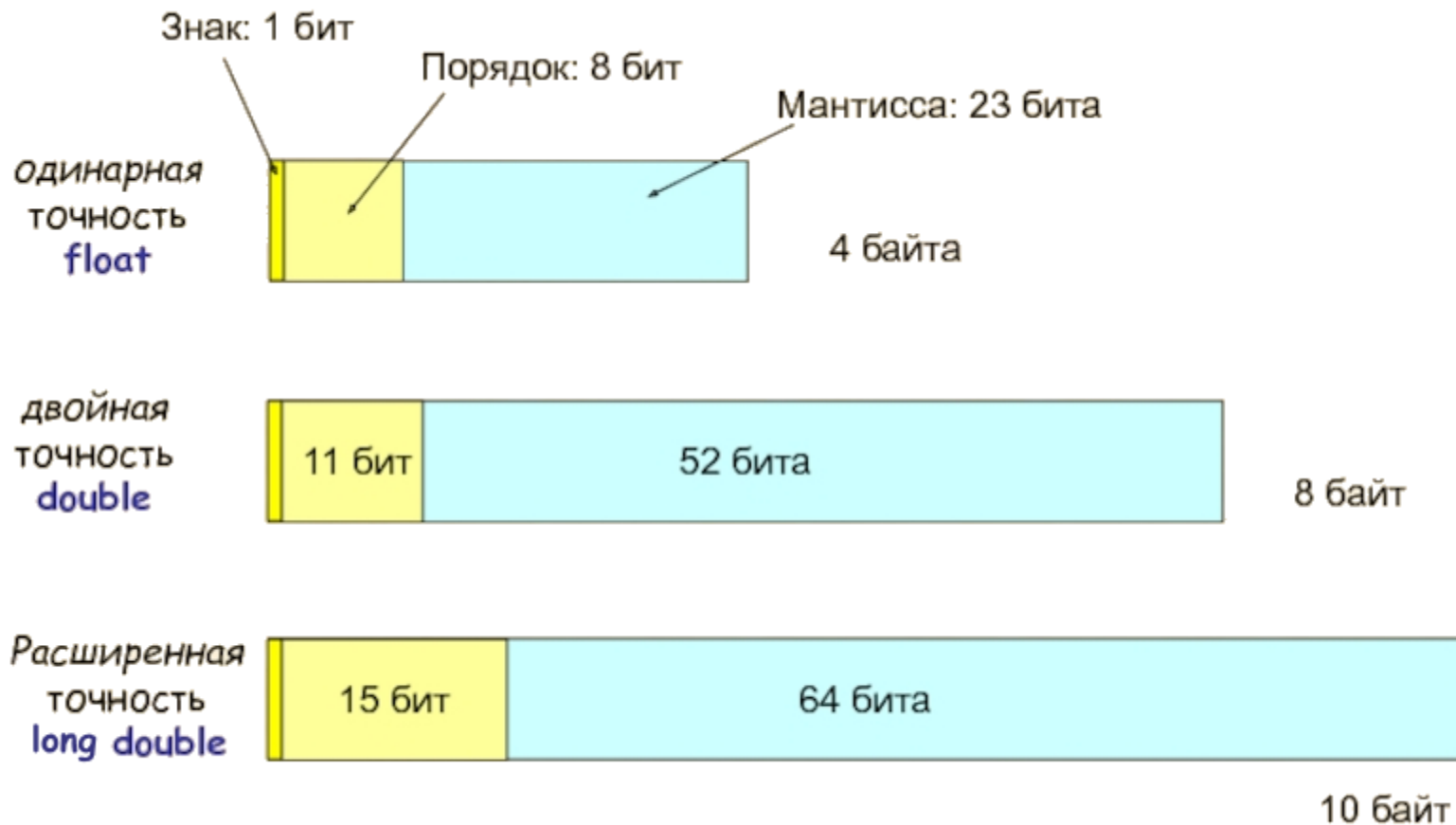
Для хранения вещественных чисел в C++ используются типы: `float`, `double`, `long double`.

Типы между собой отличаются размером, максимальным значением и количеством значащих цифр. В стандарте не указан конкретный формат, размеры и прочее, это значит, что в зависимости от платформы эти значения могут меняться.

<code>type()</code>	<code>size()</code>	<code>min()</code>	<code>max()</code>
<code>float</code>	4	1.17549e-38	3.40282e+38
<code>double</code>	8	2.22507e-308	1.79769e+308
<code>long double</code>	16	3.3621e-4932	1.18973e+4932

<https://wandbox.org/permlink/EBDqUHa2Ci41cVHO>

Числа с плавающей запятой



void | Ничто

Специальный тип **void** предназначен для обозначения отсутствия какого-либо типа в данном месте. Также используется для построения производных типов.

```
void f1 (int a);           // функция ничего не возвращает
int g1(void);              // функция ничего не получает (не используется в C++)
void* z = malloc(512);     // z указывает на "сырую" память
```

Переменную типа void создать нельзя.

Литерала типа void не существует.

std::nullptr_t | Тип для нулевого указателя

Специальный тип `std::nullptr_t` предназначен для литерала `nullptr`, который обозначает особое значение указателя, который не указывает на какой-либо объект.

```
void* pv = nullptr;  
int* iv = nullptr;
```

Альтернатива, используемая в C++, вместо макроса `NULL`, используемого, для тех же целей, в C.

char

ПОТОКОВЫЙ ВВОД/ВЫВОД

Библиотека `iostream` определяет три стандартных потока:

- `cin` стандартный входной поток (`stdin`)
- `cout` стандартный выходной поток (`stdout`)
- `cerr` стандартный поток вывода сообщений об ошибках (`stderr`)

Для выполнения операций ввода-вывода переопределены две операции поразрядного сдвига:

- `>>` получить из входного потока
- `<<` поместить в выходной поток

ПОТОКОВЫЙ ВЫВОД

```
cout << значение;
```

Здесь значение преобразуется в последовательность символов и выводится в выходной поток:

```
cout << n;
```

Возможно многократное назначение потоков:

```
cout << 'значение1' << 'значение2' << ... << 'значение n';
```

Например:

```
int n;  
char j;  
cin >> n >> j;  
cout << "Значение n равно" << n << "j=" << j;
```

ПОТОКОВЫЙ ВВОД

```
cin >> идентификатор;
```

При этом из входного потока читается последовательность символов до пробела, затем эта последовательность преобразуется к типу идентификатора, и получаемое значение помещается в **переменную**:

```
int n;  
cin >> n;
```

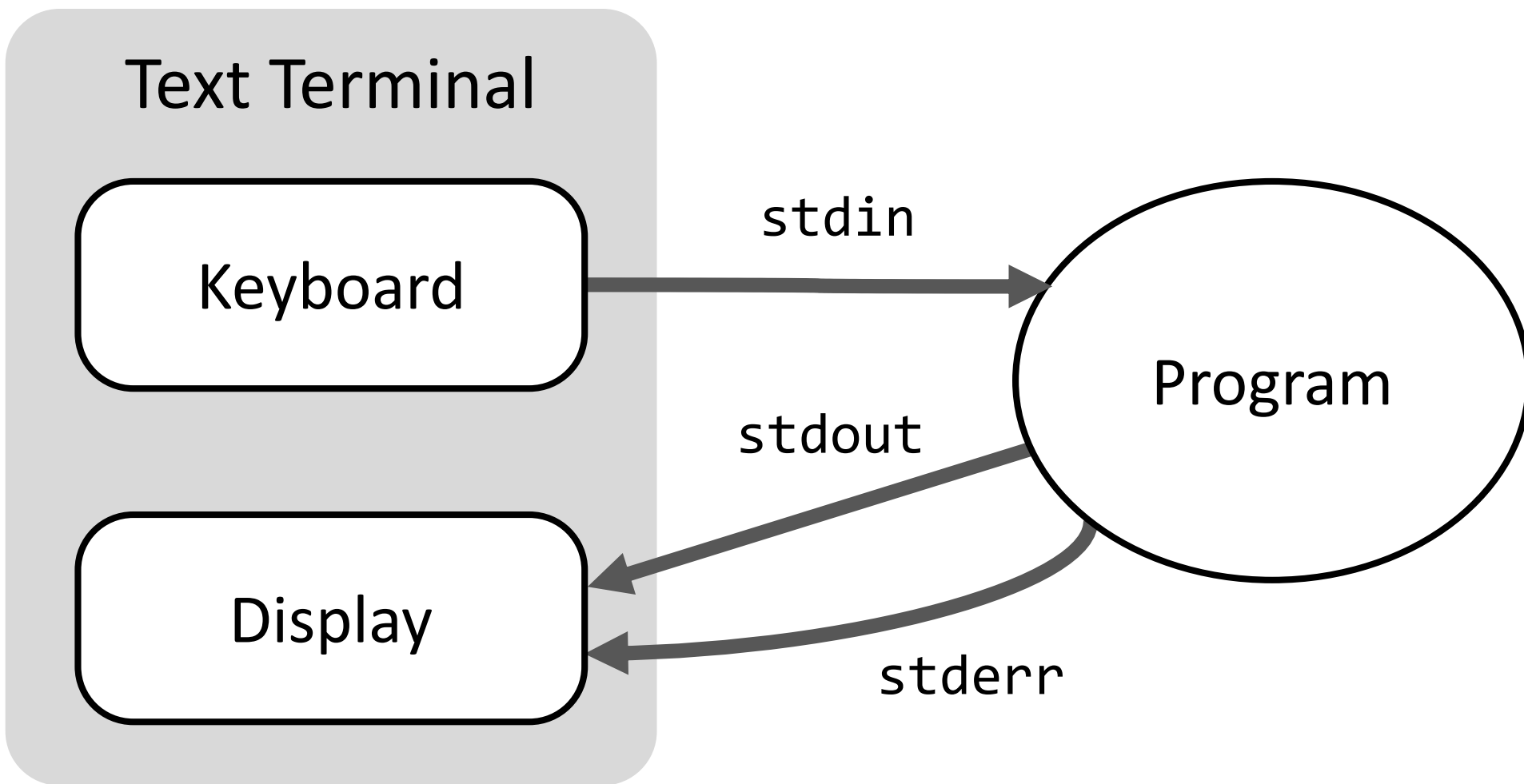
Возможно многократное назначение потоков:

```
cin >> переменная1 >> переменная2 >>...>> переменнаяn;
```

При наборе данных на клавиатуре значения для такого оператора должны быть разделены символами (пробел, **\n**, **\t**).

```
int n;  
char j;  
cin >> n >> j;
```


Стандартные потоки



Крокозябры

Unix:

```
Проснись, Нео...  
Exit Code: 0
```

Windows:

```
Консоль отладки Microsoft Visual Studio  
±Ёюёэшё№, =хю...  
C:\Users\Professional\Desktop\Box\ConsoleApplication1\Release\ConsoleApplication1.exe (процесс 1196) завершил работу с кодом 0.  
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".  
Нажмите любую клавишу, чтобы закрыть это окно...
```

setlocale

Магия которая позволяет побороть крокозябры, но работает не всегда.

Это не единственное решение, есть ещё множество вариантов.

Исходники должны быть в кодировке 1251

```
#include <windows.h>

SetConsoleCP(1251);          // установка кодовой страницы win-ср 1251 в поток ввода
SetConsoleOutputCP(1251);    // установка кодовой страницы win-ср 1251 в поток вывода

system("chcp 1251");
```

<https://wandbox.org/permlink/zEhVfp9IF76ObKvA>

Откуда берутся крокозябры

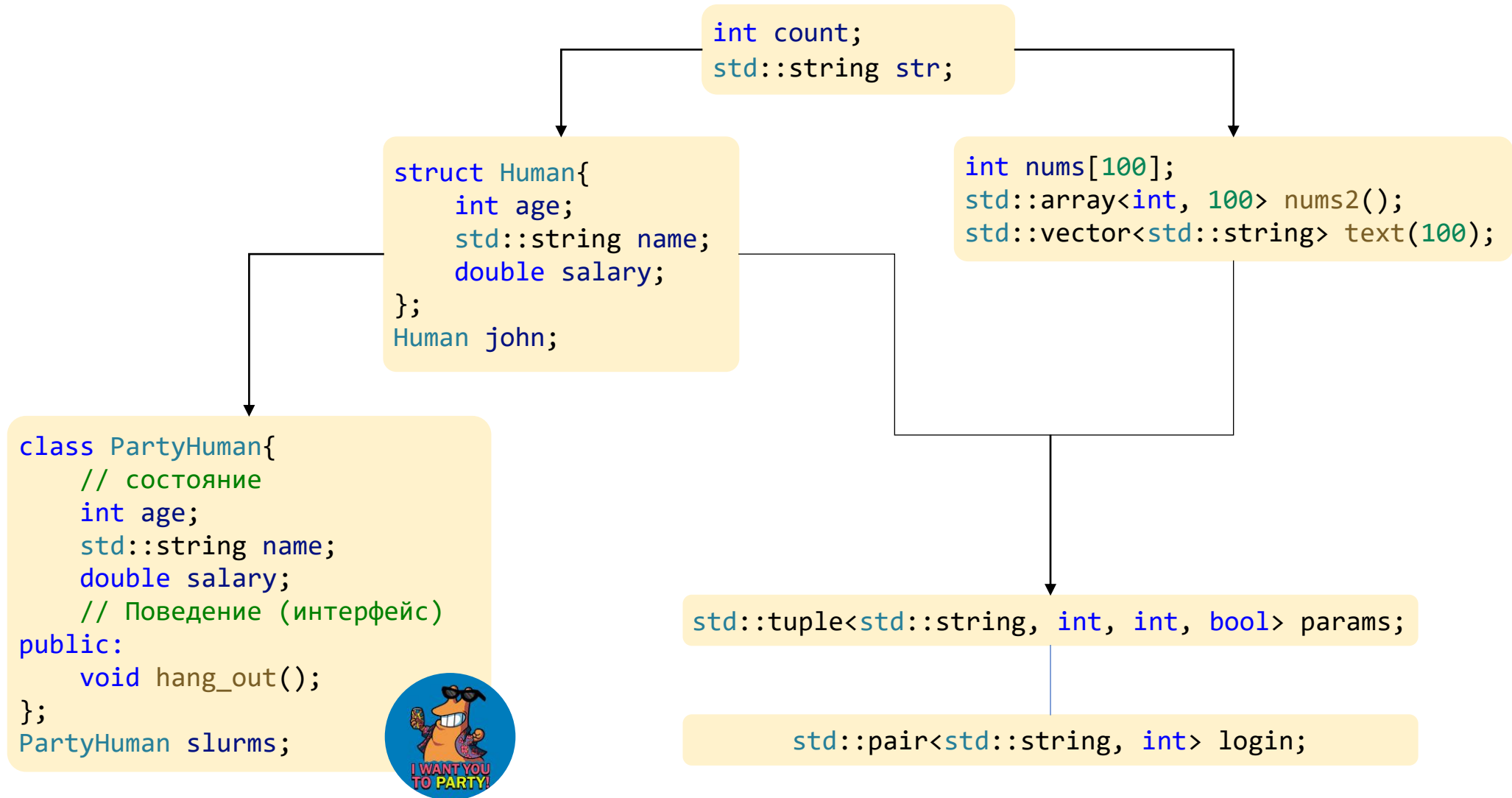
Кодировка Windows-1251 (программа)

Кодировка 866 (консоль)

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
8.	Ъ А	Ѓ Б	, В	ѓ Г	„ Д	… Е	† Ж	‡ З	€ И	‰ Й	Љ К	‹ Л	Њ М	Ќ Н	Ћ О	Ц П
9.	ђ Р	‘ С	, Т	“ У	” Ф	• Х	— Ц	— Ч		™ Щ	љ Ъ	› Ы	њ Ь	ќ Э	ћ Ю	џ Я
A.		Ў б	ў в	Ј г	Ѡ д	Г е	Ј ж	§ з	Ё и	© й	Є к	« л	¬ м		® н	Ї о
B.	° ■	± ■	І ■	і 	г └	μ ≡	¶ 	· π	ё э	№ 	е 	» ┘	ј ┘	Ѕ ┘	ѕ ┘	ї ┘
C.	А L	Б └	В └	Г └	Д —	Е └	Ж └	З 	И L	Й F	К └	Л └	М └	Н =	О └	П └
D.	Р └	С └	Т π	У └	Ф └	Х F	Ц π	Ч 	Ш └	Щ ┘	Ъ г	Ы ■	Ь ■	Э ┘	Ю ┘	Я ■
E.	а р	б с	в т	г у	д ф	е х	ж ц	з ч	и ш	й щ	к ъ	л ы	м ь	н э	о ю	п я
F.	р Ё	с ё	т Є	у е	ф Ї	х ї	ц Ў	ч ў	ш °	щ ·	ъ ·	ы √	ь №	э Ѡ	ю ■	я

Составные типы

Типы



Структуры

Постановка задачи

- Хранить в программе описание характеристик некоторого объекта

Решение I

```
int aliceBirthYear;  
int aliceBirthMonth;  
int aliceBirthDay;  
double aliceHeight;  
double aliceWeight;
```

```
int bobBirthYear;  
int bobBirthMonth;  
int bobBirthDay;  
double bobHeight;  
double bobWeight;
```

Решение I - Проблемы

- Для каждого человека нужно создавать по пять отдельных переменных – **долго, могут быть опечатки**
- Чтобы передать в функцию, нужно перечислить все аргументы – **можно перепутать порядок**

```
print(aliceBirthYear, aliceBirthMonth,  
      aliceBirthDay, aliceHeight, aliceWeight  
);
```

- Как вернуть из функции?

Решение II - Структуры

```
struct human {      // Свой тип данных
    int BirthYear;
    int BirthMonth;
    int BirthDay;
    double Height;
    double Weight;
};    // Точка с запятой обязательно
```

```
human alice, bob;    // Создаём переменные
```

Решение II - Структуры

```
struct human {  
    int BirthYear;  
    int BirthMonth;  
    int BirthDay;  
    double Height;  
    double Weight;  
} alice, bob;
```

Решение II - Структуры

```
struct {  
    int BirthYear;  
    int BirthMonth;  
    int BirthDay;  
    double Height;  
    double Weight;  
} alice, bob;
```

Где можно объявлять структуры?

- Внутри функций

```
void func(){  
    struct num{int i;} var;  
};
```

- Вне функций

```
struct num{int i;} var;  
void func(){  
};
```

- Внутри других структур

```
struct num{  
    int i;  
    struct {int k;} j;  
} var;
```

Что может быть членом структуры?

Если можно создать переменную этого типа, то это может быть членом структуры

Например:

- Примитивные типы: `int`, `double`, `char` ...
- Другие структуры;
- Массивы;
- Строки;
- ...

Как работать со структурой

```
struct Data{  
    int Year;  
    int Month;  
    int Day;  
};
```

```
Data now;  
now.Year = 2018;  
now.Day = 9;  
now.Month = 11;
```


Как работать со структурой

```
now.Year = now.Year + 1; // 2019
```

```
cout << now.Day; // 9
```

```
now.Month = now.Day + now.Year; // 2028
```

```
int *p = &now.Month;
```

Инициализация структуры I

```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};
```

```
// joe.id = 1, joe.age = 32, joe.wage = 60000.0  
Employee joe = { 1, 32, 60000.0 };  
// frank.id = 2, frank.age = 28, frank.wage = 0.0  
Employee frank = { 2, 28 };  
Employee frank { 2, 28 };    // C++11
```

Инициализация структуры II C++11/C++14

```
struct Rectangle {  
    double length = 1.0;  
    double width = 1.0;  
};
```

```
int main() {  
    Rectangle x; // length = 1.0, width =  
1.0  
    x.length = 2.0; // Меняем значение  
    return 0;  
}
```

Инициализация структуры III C++11/C++14

```
struct Rectangle {  
    double length = 1.0;  
    double width = 1.0;  
};  
  
int main() {  
    // C++11 – Ошибка; C++14 – Разрешено  
    Rectangle x = {1.0, 1.0};  
  
    return 0;  
}
```

Присваивание значений структурам I

```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};
```

```
Employee joe;  
joe.id = 1;  
joe.age = 32;  
joe.wage = 60000.0;
```

Присваивание значений структурам II

```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};
```

```
Employee joe = {1, 20, 3.0}, mike;  
mike = joe; // Копирование значений joe в mike
```

```
// Присваивание полям joe новых значений C++14  
joe = {2, 22, 6.3};
```

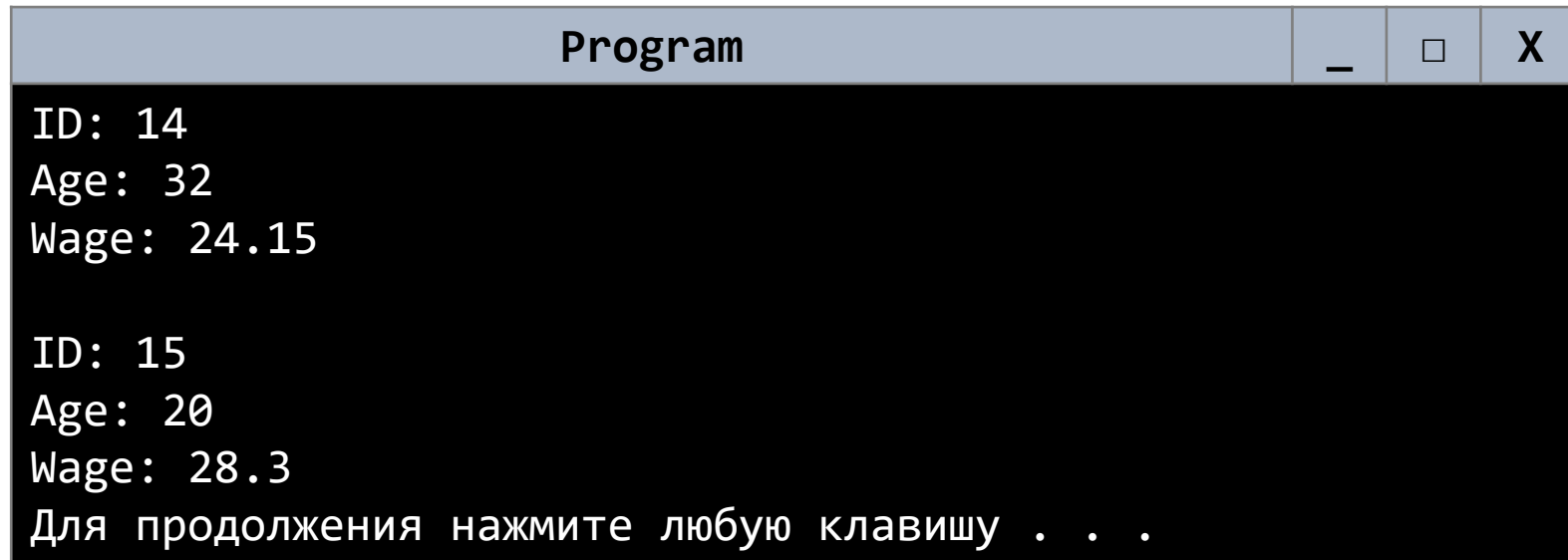
Передача структуры как параметр в функцию

```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};  
  
void printInformation(Employee employee) {  
    std::cout << "ID: " << employee.id << "\n";  
    std::cout << "Age: " << employee.age << "\n";  
    std::cout << "Wage: " << employee.wage << "\n";  
}
```

Передача структуры как параметр в функцию

```
int main() {  
    Employee joe = { 14, 32, 24.15 };  
  
    printInformation(joe);  
    std::cout << "\n";  
  
    printInformation({ 15, 20, 28.3 });  
    return 0;  
}
```


Передача структуры как параметр в функцию



```
Program
ID: 14
Age: 32
Wage: 24.15

ID: 15
Age: 20
Wage: 28.3
Для продолжения нажмите любую клавишу . . .
```

The image shows a screenshot of a Windows console window. The title bar at the top is light blue and contains the text "Program" followed by standard window control buttons (minimize, maximize, close). The main area of the window has a black background with white text. The text displays two sets of data: the first set shows "ID: 14", "Age: 32", and "Wage: 24.15"; the second set shows "ID: 15", "Age: 20", and "Wage: 28.3". At the bottom, there is a prompt in Russian: "Для продолжения нажмите любую клавишу . . .", which translates to "Press any key to continue . . .".

Передача структуры в функцию через указатель

```
void printInformation(Employee *employee) {  
    std::cout << "ID: " << (*employee).id << "\n";  
    std::cout << "Age: " << (*employee).age << "\n";  
    std::cout << "Wage: " << (*employee).wage << "\n";  
}
```

```
void printInformation(Employee *employee) {  
    std::cout << "ID: " << employee->id << "\n";  
    std::cout << "Age: " << employee->age << "\n";  
    std::cout << "Wage: " << employee->wage << "\n";  
}
```

Возврат структур из функций

```
struct Point3d {  
    double x, y, z;  
};
```

```
Point3d getZeroPoint() {  
    Point3d temp = { 0.0, 0.0, 0.0 };  
    return temp;  
}
```

```
int main() {  
    Point3d zero = getZeroPoint();  
    return 0;  
}
```

Дополнительные сведения

Разные типы

```
struct Point3d {  
    double x, y, z;  
};
```

```
struct Vector3d {  
    double x, y, z;  
};
```

```
Point3d p = { 0.0, 0.0, 0.0 };
```

```
Vector3d v;
```

```
v = p; // Ошибка. У v и p разные типы
```

Массив структур

```
struct Point3d {  
    double x, y, z;  
};
```

```
Point3d p[2] = {{}, {1.0, 2.0, 3.0}};
```

```
p[0].x = 1.0;
```

```
std::cout << p[0].x << ' ' << p[0].y << ' ' << p[0].z;
```

Вложенные структуры

```
struct Employee {  
    short id;  
    int age;  
    float wage;  
};
```

```
struct Company {  
    Employee CEO;    // CEO – это структура  
    int numberOfEmployees;  
};
```

```
Company myCompany = {{ 1, 42, 60000.0f }, 5 };  
std::cout << myCompany.CEO.id;
```

Размер структуры и выравнивание I

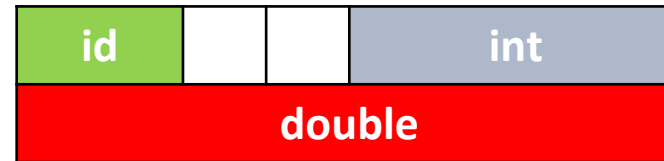
```
struct Employee {  
    short id;      // sizeof(short) == 2  
    int age;       // sizeof(int) == 4  
    double wage;   // sizeof(double) == 8  
};  
  
sizeof(Employee); // 16 != ( 2 + 4 + 8  
)
```


Размер структуры и выравнивание II

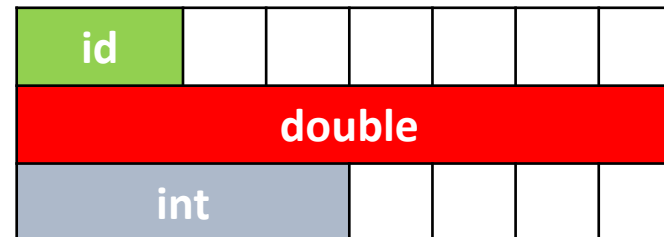
```
struct Employee {  
    short id;      // sizeof(short) == 2  
    double wage;   // sizeof(double) == 8  
    int age;       // sizeof(int) == 4  
};  
  
sizeof(Employee); // 24 != ( 2 + 4 + 8  
)
```

Размер структуры и выравнивание II

```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};
```



```
struct Employee {  
    short id;  
    double wage;  
    int age;  
};
```



Массивы

Массивы

Статические :

```
int arr[10];
```

Динамические:

```
int* arr = new int[10];  
delete[] arr;
```

STL:

```
std::array<int, 10> arr;  
std::vector<int> arr(10);
```

C++ Standard Library Sequence Containers

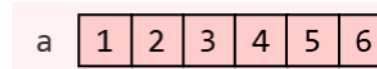
h/cpp/ hackingcpp.com

`array<T, size>`

fixed-size array

```
#include <array>
```

```
std::array<int,6> a {1,2,3,4,5,6};  
cout << a.size();    // 6  
cout << a[2];        // 3  
a[0] = 7;             // 1st element ⇒ 7
```



contiguous memory; random access; fast linear traversal

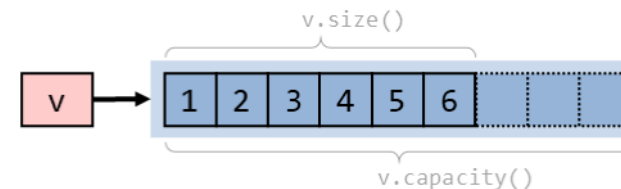
`vector<T>`

dynamic array

C++'s "default" container

```
#include <vector>
```

```
std::vector<int> v {1,2,3,4,5,6};  
v.reserve(9);  
cout << v.capacity();    // 9  
cout << v.size();        // 6  
v.push_back(7);          // appends '7'  
v.insert(v.begin(), 0);  // prepends '0'  
v.pop_back();            // removes last  
v.erase(v.begin()+2);    // removes 3rd  
v.resize(20, 0);         // size ⇒ 20
```



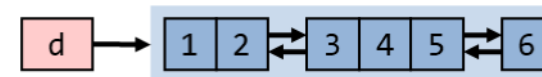
contiguous memory; random access;
fast linear traversal; fast insertion/deletion at the ends

`deque<T>`

double-ended queue

```
#include <deque>
```

```
std::deque<int> d {1,2,3,4,5,6};  
// same operations as vector  
// plus fast growth/deletion at front  
d.push_front(-1); // prepends '-1'  
d.pop_front();    // removes 1st
```



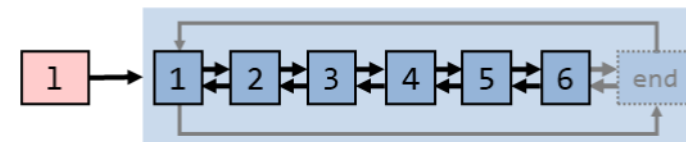
fast insertion/deletion at both ends

`list<T>`

doubly-linked list

```
#include <list>
```

```
std::list<int> l {1,5,6};  
std::list<int> k {2,3,4};  
// O(1) splice of k into l:  
l.splice(l.begin()+1, std::move(k))  
// some special member function algorithms:  
l.reverse();  
l.sort();
```



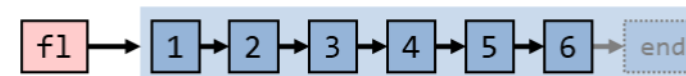
fast splicing; many operations without copy/move of elements

`forward_list<T>`

singly-linked list

```
#include <forward_list>
```

```
std::forward_list<int> fl {2,2,4,5,6};  
fl.erase_after(begin(fl));  
fl.insert_after(begin(fl), 3);  
fl.insert_after(before_begin(fl), 1);
```



lower memory overhead than std::list; only forward traversal

std::vector<ValueType>

C++'s "default"
dynamic array

#include <vector>

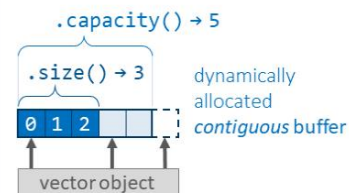
h/cpp hackingcpp.com

Construct A New Vector Object

```
vector<int> v1 {2,9,1,8,5,4} → [2, 9, 1, 8, 5, 4]
vector<int> v2 (begin(v1)+3, end(v1)) → [8, 5, 4]
vector<int> v3 (5, 3) → [3, 3, 3, 3, 3]
vector<int> deep_copy_of_v1 (v1) → [2, 9, 1, 8, 5, 4]
```

C++17 value type deducible from argument type
`vector w {7,4,2}; // vector<int>`

Typical Memory Layout



Assign New Content To An Existing Vector

```
vector<int> v1 {8,5,3}; (deep copy from source)
vector<int> v2 {6,8,1,9};
v1 = v2;
new state of v1: [6, 8, 1, 9]
v1.assign({4,1,3,5}) → [4, 1, 3, 5]
v1.assign(2, 1) → [1, 1]
v1.assign(@InBeg, @InEnd) → [2, 1, 1, 2]
source container: [3, 2, 1, 1, 2, 3]
```

Query/Change Size (= Number of Elements)

```
[8, 5, 3].empty() → false
[8, 5, 3].size() → 3
[8, 5, 3].resize(2) → [8, 5, +]
[8, 5, 3].resize(4, 1) → [8, 5, 3, 1]
[8, 5, 3].resize(6, 1) → [8, 5, 3, 1, 1, 1]
[8, 5, 3].clear() → [+, +, +]
```

Query/Grow Capacity (= Memory Buffer Size)

```
[8, 5, 3].capacity() → 4
[8, 5, 3].reserve(6) → [8, 5, 3, +, +, +]
```

Get Element Values $O(1)$ Random Access

```
[2, 8, 5, 3][1] → 8
[2, 8, 5, 3].front() → 2
[2, 8, 5, 3].back() → 3
```

Change Element Values

```
[2, 8, 5, 3][1] = 7 → [2, 7, 5, 3]
[2, 8, 5, 3].front() = 7 → [7, 8, 5, 3]
[2, 8, 5, 3].back() = 7 → [2, 8, 5, 7]
```

Out of Bounds Access

```
[2, 8, 5, 3][6] → Undefined Behavior (Invalid Index!)
[2, 8, 5, 3].at(6) → Throws Exception (std::out_of_range)
```

Erase Elements $O(n)$ Worst Case

```
vector<int> v {4,8,5,6};
[4, 8, 5, 6].pop_back() → [4, 8, 5, +]
[4, 8, 5, 6].erase(begin(v)+2) → [4, 8, 6, +]
[4, 8, 5, 6].erase(begin(v)+1, begin(v)+3) → [4, 6, +, +]
```

Shrink The Capacity (might be inefficient)

Erasing, resizing or clearing will not shrink the capacity!

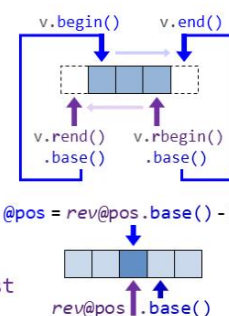
```
vector<int> v (1024, 0); // capacity is at least 1024
v.resize(40); // capacity unchanged!
v.shrink_to_fit(); // may shrink (not guaranteed)
v.swap(vector<int>(v)); // shrinks but has copy overhead
```

Obtain Iterators

$O(1)$ Random Incrementing
[0, 1, 2, 3].begin() → @first
[0, 1, 2, 3].end() → @one_behind_last

Obtain Reverse Iterators

```
[0, 1, 2, 3].rbegin() → rev@last
[0, 1, 2, 3].rend() → rev@one_before_first
```



```
[2, 8, 5, 3].data() → pointer_to_first
```

Avoid expensive memory allocations:
• **reserve** capacity before appending / inserting if you know the (approximate) number of elements to be stored in advance!

Append Elements $O(1)$ Amortized Complexity

```
[8, 5, 3].push_back(7) → [8, 5, 3, 7]
```

Insert Elements at Arbitrary Positions $O(n)$ Worst Case

```
vector<int> v {8,5,3};
[8, 5, 3].insert(begin(v), 2) → [2, 8, 5, 3]
[8, 5, 3].insert(begin(v)+1, 7) → [8, 7, 5, 3]
[8, 5, 3].insert(begin(v)+1, 3, 7) → [8, 7, 7, 7, 5, 3]
[8, 5, 3].insert(begin(v)+1, {6,9,7}) → [8, 6, 9, 7, 5, 3]
[8, 5, 3].insert(begin(v)+1, @InBegin, @InEnd) → [8, 1, 8, 9, 5, 3]
source container: [3, 1, 8, 9, 2, 3]
```

Insert & Construct Elements in Place $O(n)$ Worst Case

```
vector<pair<string,int>> v {"a",1}, {"w",7};
[a,1][w,7].emplace_back("b",4) → [a,1][w,7][b,4]
[a,1][w,7].emplace(begin(v)+1, "z",5) → [a,1][z,5][w,7]
```

Итераторы (начало)

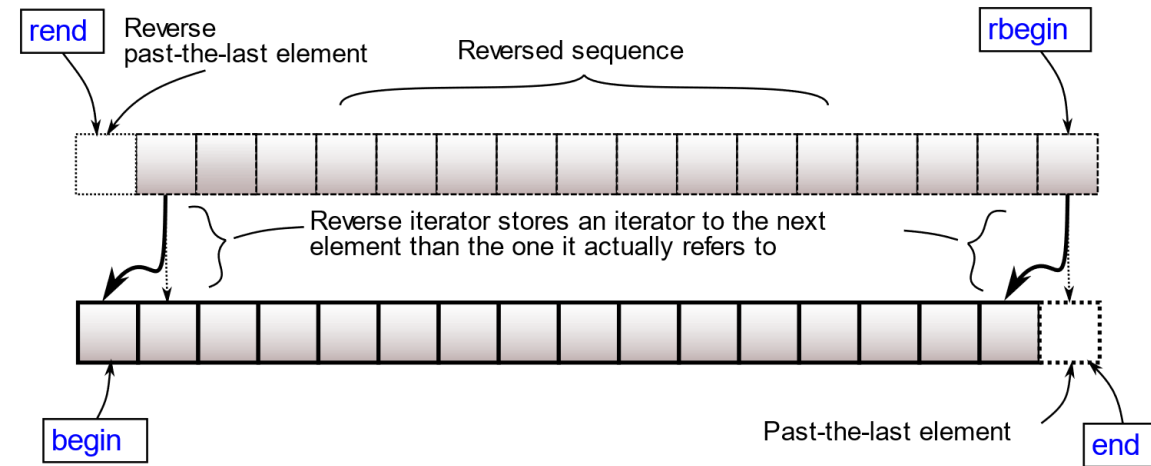
Итератор — это структура данных, предназначенная, для того чтобы перебирать элементы контейнера (последовательности), при этом не задумываясь, с каким именно контейнером происходит работа.

Получить итераторы у контейнера можно через:






- методы `begin()` и `end()`: `array.begin()`;
- одноимённые функции: `begin(array)`;

Чтобы ходить по контейнеру в обратном направлении используются реверс-итераторы: `rbegin()` и `rend()`.

Итераторы (начало)



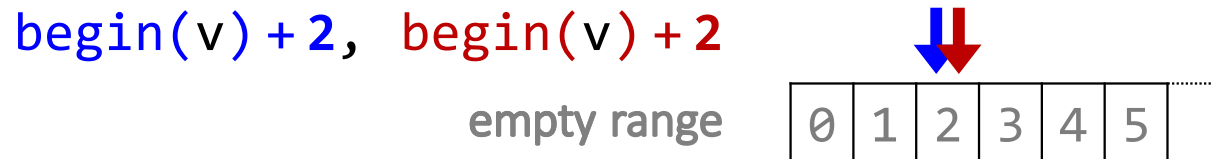
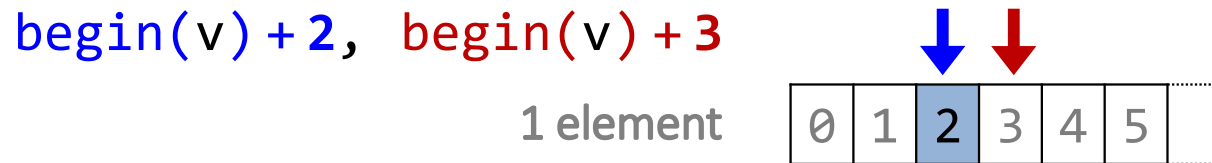
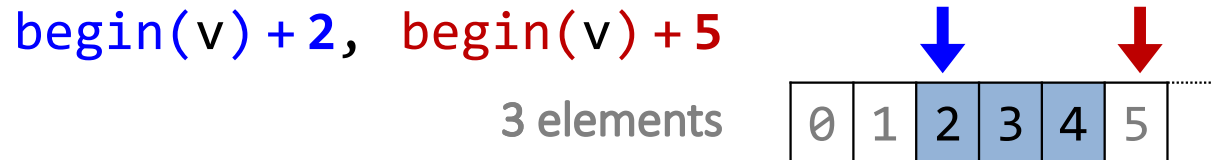
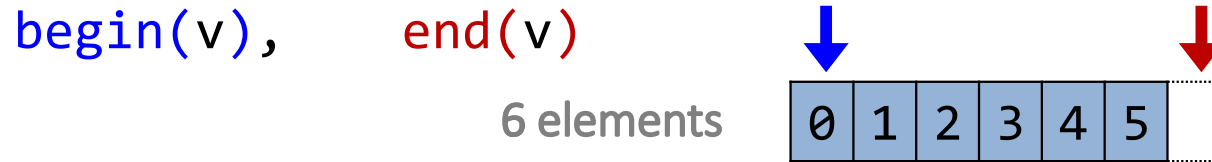
Итераторы (начало)

```
vector<int> v { 1, 2, 3, 4, 5, 6, 7, 8, 9,  };  
auto i = begin(v);   
int x = *i; // x: 1  
++i; // advance by 1   
auto j = begin(v) + 3;   
int y = *j; // y: 4  
auto e = end(v);   
*j = 47; // change element value: 4 → 47
```

DO NOT ACCESS 'END' WITH '!!'
(does not refer to valid memory)
ONLY USE AS POSITION SPECIFIER!

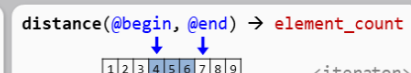
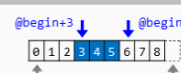
Итераторы (начало)

```
std::vector<int> v {0, 1, 2, 3, 4, 5};
```



C++ Standard Library Algorithms

Iterator
Ranges



Non-Modifying Sequence Operations

min_element(@begin, @end) → @minimum

minmax_element(@begin, @end) → {@minimum, @maximum}

any_of(@begin, @end, f(o)→bool) → true if f yields true for any, all or none of @begin, @end, f(o)→bool → false otherwise

find_if(@begin, @end, f(o)→bool) → @1st match
find(@begin, @end, value) → @end if no match

count_if(@begin, @end, f(o)→bool) → number of occurrences
count(@begin, @end, value) → occurrences

equal(@begin1, @end1, @begin2) → true if all elements in both ranges are equal

mismatch(@begin1, @end1, @begin2) → {@mismatch_in1, @mismatch_in2}

search(@beg1, @end1, @beg2, @end2) → @1st occurrence of sequence 2 inside sequence 1
@end1 otherwise

Binary Search On Sorted Sequences ⇒ O(log n)

lower_bound(@begin, @end, value) → @1st element not < value
@end if no such element exists

upper_bound(@begin, @end, value) → @1st element > value
@end if no such element exists

equal_range(@begin, @end, value) → {@1st item not < value or @end if none such found, @1st item > value or @end if none such found}

Reordering Elements

reverse(@begin, @end)

sort(@begin, @end, f(o, o)→bool)
sort(@begin, @end, std::less)

stable_sort(@begin, @end, compare(o, o)→bool)
compare(o, o)→bool
compare_case_insensitive
"stable" = preserves the relative order of equivalent elements

nth_element(@begin, @nth, @end)
< nth
≥ nth
element at nth position → element that would be in that position in a sorted sequence

partition(@begin, @end, f(o)→bool) → @part2
is_odd

rotate(@begin, @newfst, @end) → @old_begin

next_permutation(@begin, @end) → true if new permutation is lexicographically greater

Manipulate Sorted Sequences ⇒ O(n)

merge(@1beg, @1end, @2beg, @2end, @out)
sorted input
sorted output

set_union(@1beg, @1end, @2beg, @2end, @out)

Changing Values

copy(@begin, @end, @out)

transform(@begin, @end, @out, f(o)→■)
f(w) f(x) f(y)

generate(@begin, @end, f())
ascending_step_2

replace(@begin, @end, old, new)

replace_if(@begin, @end, f(o)→bool, new)
is_even

remove(@begin, @end, value) → @end_of
remove_if(@begin, @end, f(o)→bool) → remaining
is_even

unique(@begin, @end) → @end_of_remaining

erase(container, value) → erased_count
2
3

Numeric Algorithms

reduce(@begin, @end, w = ●(), ⊕ = ○+○)
reduce(@begin, @end, w, ⊕(□, ○)→■)
arbitrary evaluation order!

transform_reduce(@begin, @end, @begin2, @begin3, w, ⊕ = □+○, ⊗ = ○×○)
transform_reduce(@begin, @end, @begin2, @begin3, w, ⊕(□, ○)→■, ⊗(○, ○)→■)
transform_reduce(@begin, @end, @begin2, @begin3, w, ⊕(□, ○)→■, f(○)→■)
arbitrary evaluation order!

inclusive_scan(@begin, @end, @out, ⊕ = □+○)
inclusive_scan(@begin, @end, @out, ⊕(□, ○)→■, w)
→ @end

Sequence Queries

all_of (C++11)
any_of (C++11)
none_of (C++11)
count (C++11)
count_if (C++11)
find (C++11)
find_if (C++11)
find_if_not (C++11)
find_end (C++11)
find_first_of (C++11)
adjacent_find (C++11)
for_each (C++11)
for_each_n (C++17)
sample (C++20)
equal (C++11)
mismatch (C++11)
search (C++11)
search_n (C++11)
lexicographical_compare (C++11)
lexicographical_compare_three_way (C++20)

Reordering Elements

reverse (C++11)
reverse_copy (C++11)
rotate (C++11)
rotate_copy (C++11)
shift_left (C++20)
shift_right (C++20)
shuffle (C++11)
swap (C++11)
swap_ranges (C++11)
iter_swap (C++11)

Partitioning

is_partitioned (C++11)
partition (C++11)
stable_partition (C++11)
partition_copy (C++11)
partition_point (C++11)

Permutations

is_permutation (C++11)
next_permutation (C++11)
prev_permutation (C++11)

Sorting

sort (C++11)
stable_sort (C++11)
partial_sort (C++11)
partial_sort_copy (C++11)
is_sorted (C++11)
is_sorted_until (C++11)
nth_element (C++11)

Changing Elements

copy (C++11)
copy_backward (C++11)
copy_if (C++11)
copy_n (C++11)
move (C++11)
move_backward (C++11)
fill (C++11)
fill_n (C++11)
generate (C++11)
generate_n (C++11)
transform (C++11)
transform_copy (C++11)
transform_copy_if (C++11)
replace (C++11)
replace_copy (C++11)
replace_copy_if (C++11)
remove (C++11)
remove_copy (C++11)
remove_copy_if (C++11)
unique (C++11)
unique_copy (C++11)

Binary Search on Sorted Ranges

binary_search (C++11)
lower_bound (C++11)
upper_bound (C++11)
equal_range (C++11)
includes (C++11)

Merging of Sorted Ranges

merge (C++11)
replace_merge (C++11)
set_union (C++11)
set_intersection (C++11)
set_difference (C++11)
set_symmetric_difference (C++11)

Heaps

make_heap (C++11)
sort_heap (C++11)
push_heap (C++11)
pop_heap (C++11)
is_heap (C++11)
is_heap_until (C++11)

Minimum/Maximum

min (C++11)
max (C++11)
min_element (C++11)
max_element (C++11)
minmax (C++11)
minmax_element (C++11)
clamp (C++17)

Numeric

accumulate (C++11)
adjacent_difference (C++11)
inner_product (C++11)
partial_sum (C++11)
iota (C++11)
reduce (C++11)
inclusive_scan (C++11)
exclusive_scan (C++11)
transform_reduce (C++11)
transform_inclusive_scan (C++11)
transform_exclusive_scan (C++11)