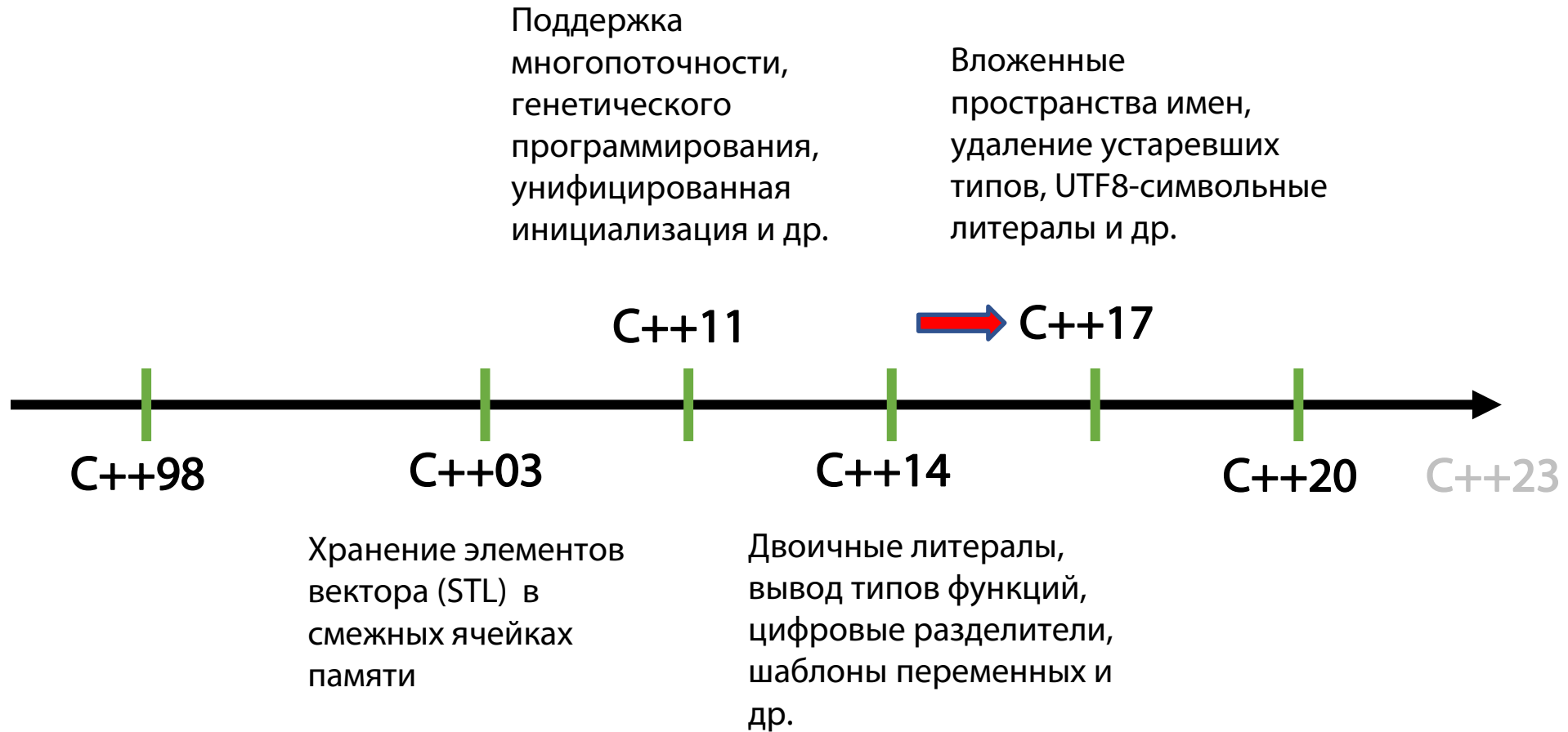


# Стандарты C++



# Тип `void`

Имеет пустое множество допустимых значений.

- Функция, которая не возвращает значение

```
void f(int a) { cout << a << endl; }
```

- Указатель, который может указывать на объекты любого типа  
(небезопасно)

```
void *ptr;
```

# Целочисленный тип данных `int`

Модификаторы знака	
<code>signed</code>	Представление целого числа со знаком (по умолчанию)
<code>unsigned</code>	Представление целого числа без знака

Модификаторы размера	
<code>short</code>	16-битное представление целого числа
<code>long</code>	32-битное представление целого числа
<code>long long</code>	64-битное представление целого числа

Модификаторы знака и размера могут комбинироваться

# Целочисленный тип данных `int`

Модификаторы знака	
<code>signed</code>	Представление целого числа со знаком (по умолчанию)
<code>unsigned</code>	Представление целого числа без знака

Модификаторы размера	
<code>short</code>	16-битное представление целого числа
<code>long</code>	32-битное представление целого числа
<code>long long</code>	64-битное представление целого числа

Модификаторы знака и размера могут комбинироваться

**Q: Каков диапазон допустимых значений типа `long int`?**

# Целочисленный тип данных `int`

```
int x;  
cout << "The size of x is " << sizeof(x) * 8 << " bits" << endl;  
  
long int y;  
cout << "The size of y is " << sizeof(y) * 8 << " bits" << endl;  
  
short unsigned z;  
cout << "The size of z is " << sizeof(z) * 8 << " bits" << endl;  
  
int16_t t;  
cout << "The size of t is " << sizeof(t) * 8 << " bits" << endl;
```

The size of x is 32 bits  
The size of y is 32 bits  
The size of z is 16 bits  
The size of t is 16 bits

# Символьные типы данных

char	8-битное представление символов
char16_t	Для поддержки представления 16 и 32-битных символов Unicode
char32_t	

```
char c1 = 'p';  
char c2 = 98;  
char c3 = -85;  
cout << c2 << '\\n';  
cout << c3;
```

б  
л

# Логический тип данных `bool`

Диапазон включает два значения: `true` (1) и `false` (0).

```
int k = 3;
if (k >= 2)
    cout << "yes";
else
    cout << "no";
```

yes

```
bool isEqual(int x, int y)
{
    return (x == y);
}
x = 2; y = 4;
cout << isEqual(x, y);
```

0

```
if (6)
    cout << "yes"
else
    cout << "no"
```

???

# Типы данных с плавающей точкой C++

Тип	Минимальный размер
float	32-битное представление вещественных чисел
double	64-битное представление вещественных чисел
long double	

**`sizeof(long double) >= sizeof(double) >= sizeof(float)`**



# Типы данных с плавающей точкой C++

```
double d = 1 / 10;  
double r = 0.1;  
  
if (d == r)  
    cout << "equal";  
else  
    cout << "not equal";
```

???

```
double d = 1.0;  
double r = 0.1 + 0.1 + 0.1 +  
0.1 + 0.1 + 0.1 + 0.1 + 0.1 +  
0.1 + 0.1;  
  
if (d == r)  
    cout << "equal";  
else  
    cout << "not equal";
```

???

# Типы данных с плавающей точкой C++

```
double d = 1 / 10;  
double r = 0.1;  
  
if (d == r)  
    cout << "equal";  
else  
    cout << "not equal";
```

not equal

```
double d = 1.0;  
double r = 0.1 + 0.1 + 0.1 +  
0.1 + 0.1 + 0.1 + 0.1 + 0.1 +  
0.1 + 0.1;
```

```
if (d == r)  
    cout << "equal";  
else  
    cout << "not equal";  
cout << setprecision(17);  
cout << d << endl;  
cout << r << endl;
```

not equal  
1  
0.999999999999999989

# Типы данных с плавающей точкой C++

## Сравнение чисел с ограниченной точностью

```
double d = 1.0;
double r = 0.1 + 0.1 + 0.1 +
0.1 + 0.1 + 0.1 + 0.1 + 0.1 +
0.1 + 0.1;
double epsilon = 0.0001;

if (fabs(d - r) < epsilon)
    cout << "equal";
else
    cout << "not equal";

equal
```

# Типы данных с плавающей точкой C++



## Бесконечности и неопределенности

<pre><b>double</b> zero = 0.0; <b>double</b> pinf = 1.0 / zero; <b>double</b> ninf = -5.0 / zero; <b>double</b> nan = zero / zero; <b>double</b> nan1 = <b>sqrt</b>(-6);  <b>cout</b> &lt;&lt; pinf &lt;&lt; <b>endl</b>; <b>cout</b> &lt;&lt; ninf &lt;&lt; <b>endl</b>; <b>cout</b> &lt;&lt; nan &lt;&lt; <b>endl</b>; <b>cout</b> &lt;&lt; nan1 &lt;&lt; <b>endl</b>;</pre>	<pre>inf -inf -nan(ind) -nan(ind)</pre>
--	---

# Структура программы на C++

## Документация

- Общие пояснения к программе в комментариях

## Ссылки

- Подключение заголовочных файлов и пространств имен

## Определения

- Определение пользовательских типов, констант, `#define`

## Глобальные объявления

- Объявление переменных, классов, структур, которые доступны до окончания работы программы

## Пользовательские функции

## Точка входа компилятора

- Функция `main( ) { ... }`

# Структура программы на C++

## Вычисление факториала

```
/* Программа итерационного вычисления факториала */
```

```
#include <iostream>  
using namespace std;
```

```
#define msg "FACTORIAL\n"  
typedef int k;
```

```
k num = 0, fact = 1, value = 0;
```

```
k factorial(k& num) {  
    for (k i = 1; i <= num; i++) {  
        fact *= i;  
    }  
    return fact;  
}
```

```
int main() {  
    k Num = 5;  
    value = factorial(Num);  
    cout << msg;  
    cout << Num << "! = " << value << endl;  
    return 0;  
}
```

# Структура программы на C++

## Вычисление факториала

```
/* Программа итерационного вычисления факториала */
```

```
#include <iostream>
using namespace std;
```

```
#define msg "FACTORIAL\n"
typedef int k;
```

```
k num = 0, fact = 1, value = 0;
```

```
k factorial(k& num) {
    for (k i = 1; i <= num; i++) {
        fact *= i;
    }
    return fact;
}
```

```
int main() {
    k Num = 5;
    value = factorial(Num);
    cout << msg;
    cout << Num << "! = " << value << endl;
    return 0;
}
```

← Цель программы

# Структура программы на C++

## Вычисление факториала

```
/* Программа итерационного вычисления факториала */
```

```
#include <iostream>  
using namespace std;
```

```
#define msg "FACTORIAL\n"  
typedef int k;
```

```
k num = 0, fact = 1, value = 0;
```

```
k factorial(k& num) {  
    for (k i = 1; i <= num; i++) {  
        fact *= i;  
    }  
    return fact;  
}
```

```
int main() {  
    k Num = 5;  
    value = factorial(Num);  
    cout << msg;  
    cout << Num << "! = " << value << endl;  
    return 0;  
}
```

Подключение стандартных средств поддержки ввода-вывода и соответствующего пространства имен





# Структура программы на C++

## Вычисление факториала

```
/* Программа итерационного вычисления факториала */
```

```
#include <iostream>
using namespace std;
```

```
#define msg "FACTORIAL\n"
typedef int k;
```

```
k num = 0, fact = 1, value = 0;
```

```
k factorial(k& num) {
    for (k i = 1; i <= num; i++) {
        fact *= i;
    }
    return fact;
}
```

```
int main() {
    k Num = 5;
    value = factorial(Num);
    cout << msg;
    cout << Num << "! = " << value << endl;
    return 0;
}
```

← Создание макроса `msg` и  
псевдонима для стандартного  
типа `int`

# Структура программы на C++

## Вычисление факториала

```
/* Программа итерационного вычисления факториала */
```

```
#include <iostream>
using namespace std;
```

```
#define msg "FACTORIAL\n"
typedef int k;
```

```
k num = 0, fact = 1, value = 0;
```

```
k factorial(k& num) {
    for (k i = 1; i <= num; i++) {
        fact *= i;
    }
    return fact;
}
```

```
int main() {
    k Num = 5;
    value = factorial(Num);
    cout << msg;
    cout << Num << "! = " << value << endl;
    return 0;
}
```

← Объявление глобальных переменных num, fact, value типа k

# Структура программы на C++

## Вычисление факториала

```
/* Программа итерационного вычисления факториала */
```

```
#include <iostream>
using namespace std;
```

```
#define msg "FACTORIAL\n"
typedef int k;
```

```
k num = 0, fact = 1, value = 0;
```

```
k factorial(k& num) {
    for (k i = 1; i <= num; i++) {
        fact *= i;
    }
    return fact;
}
```

```
int main() {
    k Num = 5;
    value = factorial(Num);
    cout << msg;
    cout << Num << "! = " << value << endl;
    return 0;
}
```

← Определение функции,  
которая вычисляет факториал  
и возвращает его значение в  
глобальную переменную  
fact

# Структура программы на C++

## Вычисление факториала

```
/* Программа итерационного вычисления факториала */
```

```
#include <iostream>  
using namespace std;
```

```
#define msg "FACTORIAL\n"  
typedef int k;
```

```
k num = 0, fact = 1, value = 0;
```

```
k factorial(k& num) {  
    for (k i = 1; i <= num; i++) {  
        fact *= i;  
    }  
    return fact;  
}
```

```
int main() {  
    k Num = 5;  
    value = factorial(Num);  
    cout << msg;  
    cout << Num << "! = " << value << endl;  
    return 0;  
}
```

←  
Функция `main()`, которая  
возвращает 0 в результате  
успешного вычисления  
факториала `Num`

# Структура программы на C++

## Вычисление факториала

```
/* Программа итерационного вычисления факториала */  
  
#include <iostream>  
using namespace std;  
  
#define msg "FACTORIAL\n"  
typedef int k;  
  
k num = 0, fact = 1, value = 0;  
  
k factorial(k& num) {  
    for (k i = 1; i <= num; i++) {  
        fact *= i;  
    }  
    return fact;  
}  
  
int main() {  
    k Num = 5;  
    value = factorial(Num);  
    cout << msg;  
    cout << Num << "! = " << value << endl;  
    return 0;  
}
```

←  
Функция `main()`, которая  
возвращает 0 в результате  
успешного вычисления  
факториала `Num`

# Стандартные библиотеки функций

Находятся в пространстве имен `std`

Заголовочные файлы	Назначение
<code>iostream, iomanip, fstream, ...</code>	Ввод-вывод, форматирование
<code>string</code>	Работа со строками
<code>math</code>	Математические операции и функции
<code>complex</code>	Функции для работы с комплексными числами
<code>cstdlib</code>	Функции общего назначения
<code>...</code>	<code>...</code>
<code>algorithm, bitset, map, queue, ...</code>	Стандартная библиотека шаблонов (STL)

# Потоки ввода и вывода <iostream>

---

- **cin** – класс, связанный со стандартным **ВВОДОМ** (клавиатура)
- **cout** – класс, связанный со стандартным **ВЫВОДОМ** (экран)
- **cerr** - класс, связанный со стандартным **ВЫВОДОМ** сообщений об **ошибках** (экран)

# Потоки ввода и вывода <iostream>

```
#include <iostream>
#include <cstdlib> // для экстренного завершения программы

int main() {
    std::cout << "Enter positive number: " << std::endl;

    int x;
    std::cin >> x;

    if (x <= 0) {
        std::cerr << "You entered a negative number!\n";
        exit(1);
    }

    std::cout << "Terminated correctly" << std::endl;
    return 0;
}
```

Enter positive number:  
-9  
You entered a negative number!

Enter positive number:  
8  
Terminated correctly



# Потоки ввода и вывода <iostream>

```
#include <iostream>
#include <cstdlib> // для экстренного завершения программы

int main() {
    std::cout << "Enter positive number: " << std::endl;

    int x;
    std::cin >> x;

    if (x <= 0) {
        std::cerr << "You entered a negative number!\n";
        exit(1);
    }

    std::cout << "Terminated correctly" << std::endl;
    return 0;
}
```

Enter positive number:  
-9  
You entered a negative number!

Enter positive number:  
8  
Terminated correctly

# Форматирование вывода

---

- **Флаги** – логические переменные, которые определяют формат вывода (включение `setf`, отключение `unsetf`)
- **Манипуляторы** – объекты, помещаемые в поток вывода и изменяющие формат вывода (автоматически включают и отключают флаги)

# Форматирование вывода

## Вывод в разных системах счисления

```
#include <iostream>

int main() {
    std::cout << std::hex << 128 << std::endl;
    std::cout << 679 << std::endl;
    std::cout << std::oct << 65536 << std::endl;
    return 0;
}
```

```
80
2a7
200000
```

## Вывод логических значений

```
#include <iostream>

int main() {
    int x = 4; int y = 6;
    std::cout << (x < y) << std::endl;
    std::cout << std::boolalpha << (y < x);
    return 0;
}
```

```
1
false
```

Манипулятор действует до тех пор, пока в поток вывода не помещен другой манипулятор

# Форматирование вывода

## Задание точности

```
#include <iostream>

int main() {
    double k = 0.1;

    std::cout << std::fixed;
    std::cout << std::setprecision(9);
    std::cout << k << std::endl;
    std::cout << std::setprecision(17);
    std::cout << k << std::endl;

    return 0;
}
```

```
0.100000000
0.100000000000000001
```

## Экспоненциальная запись

```
#include <iostream>

int main() {
    double k = 0.1 / 0.3;

    std::cout << std::scientific;
    std::cout << std::setprecision(9);
    std::cout << k << std::endl;
    std::cout << std::setprecision(17);
    std::cout << k << std::endl;

    return 0;
}
```

```
3.333333333e-01
3.33333333333333370e-01
```

# Указатели и динамическая память



# Указатели. Оператор адреса &

## Вывод адреса переменной в памяти

```
#include <iostream>
```

```
int main() {  
    int i1 = 6345;  
    std::cout << i1 << std::endl;  
    std::cout << &i1 << std::endl;  
}
```

```
6345  
006FF918
```

## Память

i1	...	...
	...	...
	006FF918	6345
	...	...
	...	...

# Указатели. Оператор разыменовывания \*

Вывод значения переменной по ее адресу

```
#include <iostream>

int main() {
    int i1 = 6345;
    std::cout << &i1 << std::endl;
    std::cout << *&i1 << std::endl;
}
```

```
006FF918
6345
```

Память

i1	...	...
	...	...
	006FF918	6345
	...	...
	...	...

# Указатели. Объявление и определение

Указатели хранят адреса ячеек памяти

## Объявление указателей

```
int *xPtr;  
double *yPtr;  
  
double* zPtr, tPtr;
```



# Указатели. Объявление и определение

Указатели хранят адреса ячеек памяти

## Объявление указателей

```
int *xPtr;  
double *yPtr;  
  
double* zPtr, tPtr;  
cout << typeid(tPtr).name();
```

```
double
```

# Указатели. Объявление и определение

Указатели хранят адреса ячеек памяти

## Объявление указателей

```
int *xPtr;  
double *yPtr;  
  
double* zPtr, tPtr;  
cout << typeid(tPtr).name();
```

double

## Определение указателей

```
int x = -489;  
int *ptr = &x;  
  
cout << &x << ' ' << ptr;
```

0133F7C0 0133F7C0

# Указатели. Объявление и определение

Указатели хранят адреса ячеек памяти

## Определение указателей

```
int x = -489;  
int *ptr = &x;  
  
cout << &x << ' ' << ptr;  
  
0133F7C0 0133F7C0
```

## Определение указателей

```
int x = -489;  
double y = 9.569;  
  
int *xPtr = &x;  
double *yPtr = &y;  
  
xPtr = &y;  
xPtr = 7;  
yPtr = 0x0012FF80;
```

# Указатели. Объявление и определение

Указатели хранят адреса ячеек памяти

## Определение указателей

```
int x = -489;  
int *ptr = &x;  
  
cout << &x << ' ' << ptr;
```

```
0133F7C0 0133F7C0
```

## Определение указателей

```
int x = -489;  
double y = 9.569;  
  
int *xPtr = &x;  
double *yPtr = &y;
```

```
xPtr = &y;  
xPtr = 7;  
yPtr = 0x0012FF80;
```

# Указатели. Разыменовывание \*



- `ptr` хранит ссылку на переменную `i1`  
**`ptr` есть то же, что и `&i1`**
- `*ptr` возвращает значение переменной `i1`, которая находится по адресу, записанному в `ptr`  
**`*ptr` есть то же, что и `i1`**

# Указатели. Разыменовывание \*

## Получение содержимого ячейки памяти

```
int a = 5489;
int b = -5777;

int *ptr;

ptr = &a;
cout << *ptr << endl;

ptr = &b;
cout << *ptr << endl;

cout << typeid(&a).name( );
```

5489  
-5777  
int \*

# Указатели. Разыменовывание \*

## Разыменовывание неинициализированных указателей

```
#include <iostream>

void f1(int *&ptr) { }

int main() {
    int *ptr;
    f1(ptr);

    std::cout << *ptr;
    return 0;
}
```

...

# Указатели. Размер указателей

## Проверка объема памяти, выделенной под указатели

```
double *xPtr;  
int *yPtr;  
  
struct Point  
{  
    int x, y, z;  
};
```

```
Point *zPtr;
```

```
cout << sizeof(xPtr) << endl;  
cout << sizeof(yPtr) << endl;  
cout << sizeof(zPtr) << endl;
```

???



# Использование указателей

---

- **Массивы** реализованы посредством указателей
- **Динамическое выделение памяти** выполняется через указатели
- Передача большого количество данных в функцию **без копирования** передаваемых данных
- ...

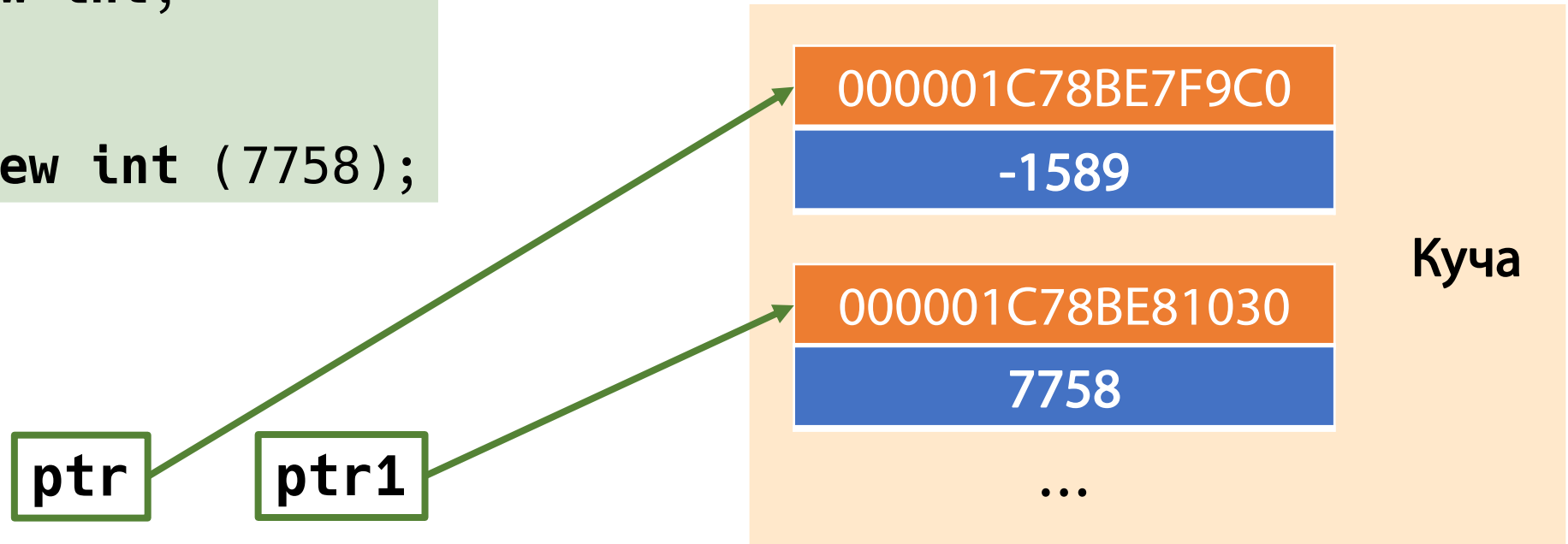
- Статическое выделение памяти – статические и глобальные переменные
- Автоматическое выделение памяти – локальные переменные и параметры функций
- Динамическое выделение памяти – по мере необходимости

- Статическое выделение памяти – статические и глобальные переменные
- Автоматическое выделение памяти – локальные переменные и параметры функций
- ➔ • Динамическое выделение памяти – по мере необходимости

# Динамическое выделение памяти. Оператор new

Динамическое выделение памяти для переменных

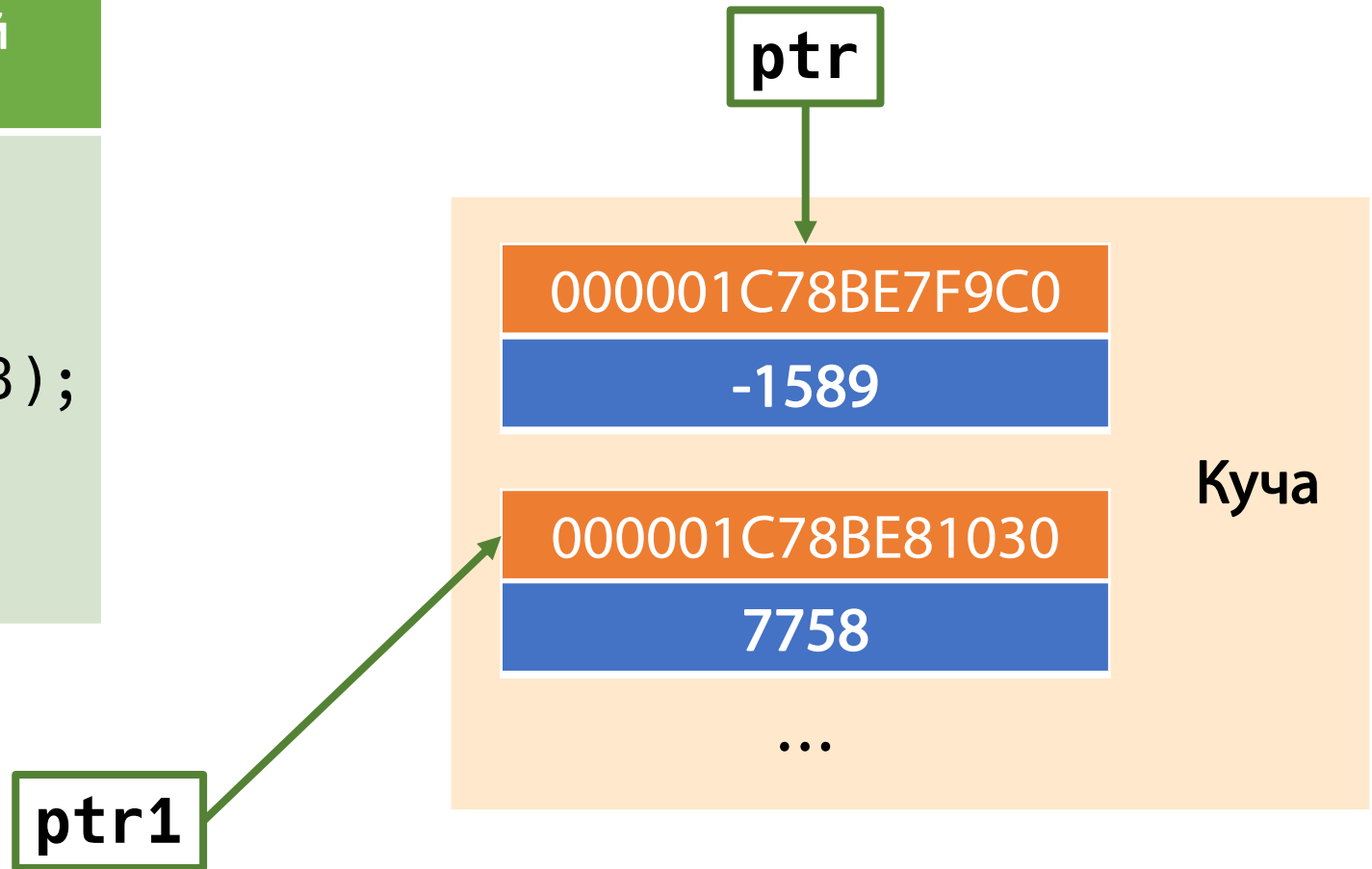
```
int *ptr = new int;  
*ptr = -1589;  
  
int *ptr1 = new int (7758);
```



# Освобождение динамической памяти. Оператор delete

Освобождение ранее выделенной  
динамической памяти

```
int *ptr = new int;  
*ptr = -1589;  
  
int *ptr1 = new int (7758);  
  
delete ptr;  
ptr = nullptr;
```



# Освобождение динамической памяти. Оператор delete

Освобождение ранее выделенной  
динамической памяти

```
int *ptr = new int;  
*ptr = -1589;  
  
int *ptr1 = new int (7758);  
  
delete ptr;  
ptr = nullptr;
```

**ptr** 00...0

**ptr1**

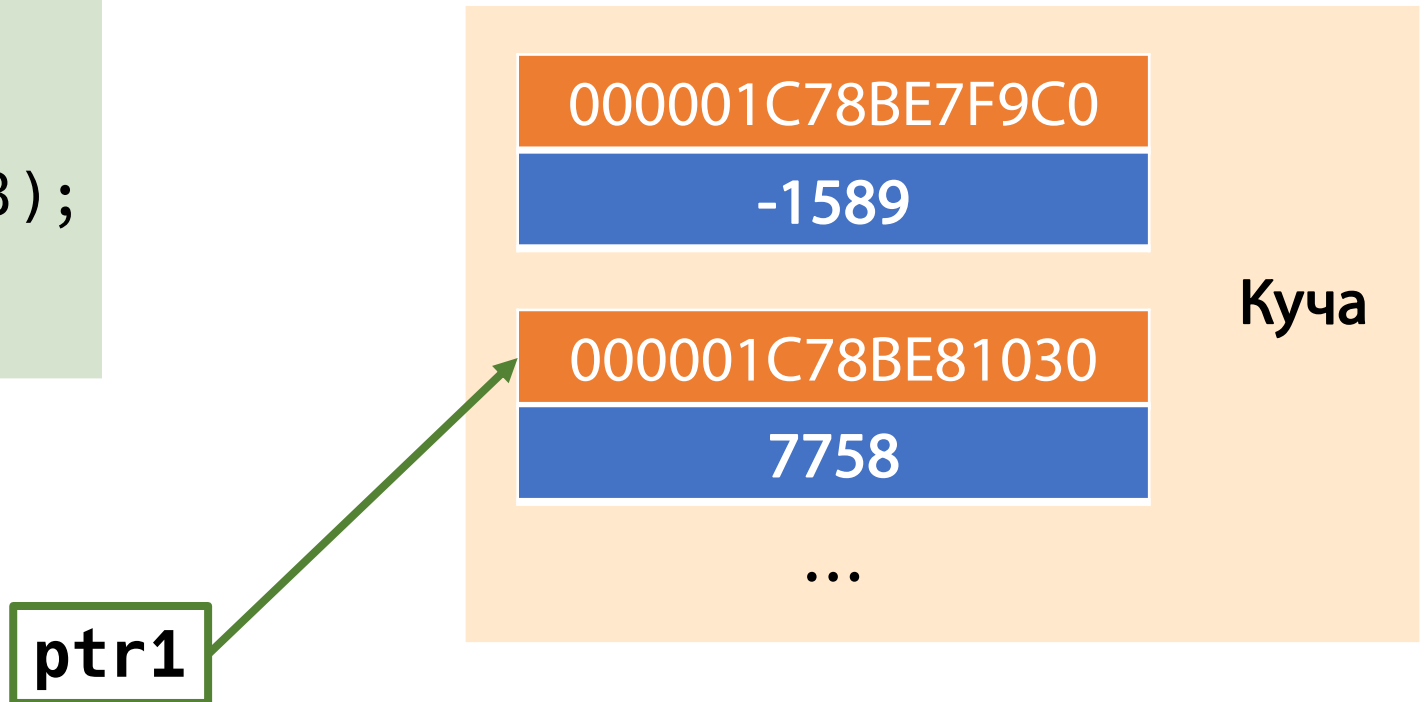


# Висячий указатель. Оператор delete

Освобождение ранее выделенной  
динамической памяти

```
int *ptr = new int;  
*ptr = -1589;  
  
int *ptr1 = new int (7758);  
  
delete ptr;
```

**ptr** 0x8123



# Утечка памяти – потеря адресов

## Выход указателя из области видимости

```
void doIt( )
{
    int *ptr = new int;
}

int main( )
{
    ...;
    doIt( );
    ...;
}
```

## Переприсваивание указателей

```
int x = -567;
int *xPtr = new int;
ptr = &x;

int *nPtr = new int;
nPtr = new int;
```



# Ссылки



# Ссылки в C++

---

- Ссылки на неконстантные значения
- Ссылки на константные значения

# Ссылки в C++

## Ссылка как псевдоним объекта

```
#include <iostream>
using namespace std;

int main() {
    int x = 1492;
    int &xRef = x;

    x = 1728;
    xRef = 1961;

    cout << x << endl;
    x--;
    cout << xRef << endl;
}
```

1961  
1960

Ссылки ведут себя в точности так же, как и значения на которые они ссылаются

**&x = xRef**

## Ссылка для упрощения доступа к данным

```
struct Date
{
    int day;
    int month;
    int year;
};

struct Employee
{
    string name;
    Date birthDay;
    double salary;
};
```

```
Employee Fred;
Fred = {"Fred Williams", {12, 3, 1989}, 3450};

Fred.birthday.month = 4;

int &ref = Fred.birthday.month;
ref = 10;
```

# Ссылки в C++

- Ссылка – указатель, который неявно разыменовывается при доступе к значению объекта
- Ссылка обязательно должна быть проинициализирована
- Ссылка не может быть изменена

```
int x = -7985;  
int y = 12;
```

```
int &xRef = x;  
xRef = y;
```

???

# Функции



# Функция, параметры, аргументы

```
#include <iostream>
using namespace std;

void print(int a1, int d, int n) {
    for (int i = 0; i < n; i++) {
        cout << a1 + i * d << ' ';
    }
}

int main() {
    print(5, -6, 25);
    return 0;
}
```

Аргументы передаются в функцию:

- по значению
- по ссылке
- по адресу

# Передача по значению

```
#include <iostream>
using namespace std;

void print(int a1, int d, int n) {
    for (int i = 0; i < n; i++) {
        cout << a1 + i * d << ' ';
    }
}

int main() {
    int x = 4;
    print(x, -6, 5 * 4);
    return 0;
}
```

Аргументы передаются в функцию:

- **по значению**
- Аргументы не изменяются функцией
- Аргументы полностью копируются в функцию



# Передача по ссылке

```
#include <iostream>
using namespace std;

void inc(int &x) {
    x = x + 1;
}

int main() {
    int loc = 14;

    cout << "loc = " << loc << '\n';
    inc(loc);
    cout << "loc = " << loc << '\n';
    return 0;
}
```

Аргументы передаются в функцию:

- **по ссылке**
- Аргументы не копируются в функцию
- Аргументы могут изменяться функцией
- Функция может вернуть сразу несколько значений

# Передача по ссылке

## Возврат нескольких значений через аргументы функции

```
#include <iostream>
#include <cmath>
using namespace std;

void trig(double arg, double &sOut, double &cOut)
{
    const pi = 3.14;
    double rads = arg * pi / 180;
    sOut = sin(rads);
    cOut = cos(rads);
}
```

```
int main() {
    double s = 0.0;
    double c = 0.0;

    trig(30.0, s, c);

    cout << s << '\n';
    cout << c << '\n';
}
```

# Передача по константной ссылке

## Защита от изменения аргументов внутри функции

```
#include <iostream>
#include <cmath>
using namespace std;

void sinus(const double &arg, double &sin)
{
    const pi = 3.14;
    arg = arg * pi / 180;
    sin = sin(arg);
}
```

```
int main() {
    double s = 0.0;

    double x = 180.0;
    sinus(x, s);
}
```

# Передача по константной ссылке

## Защита от изменения аргументов внутри функции

```
#include <iostream>
#include <cmath>
using namespace std;

void sinus(const double &arg, double &sin)
{
    const pi = 3.14;
    arg = arg * pi / 180;
    sin = sin(arg);
}
```

```
int main() {
    double s = 0.0;

    double x = 180.0;
    sinus(x, s);
}
```

# Передача по константной ссылке

---

- Гарантия защиты от изменения передаваемых аргументов
- Константные значения могут быть переданы в функцию только посредством константных ссылок
- Константные ссылки могут принимать любые типы аргументов

# Решаем у доски...1

```
#include <iostream>
using namespace std;

int f(int a, int &b, int &c) {
    a = b + a;
    b = a - b;
    c = c + a + b;
    return a + b + c;
}

int main () {
    int a, b, c, d;
    a=5; b=6; c=7; d=8;
    d = f(a, b, c);
    cout<<a<<b<<c<<d<<endl;
}
```

Что будет выведено в результате работы программы?

## Решаем у доски...2

```
#include <iostream>
using namespace std;

int f(int a, int &b, int &c) {
    a = b + a;
    b = a - b;
    c = c + a + b;
    return a + b + c;
}

int main () {
    int a, b, c, d;
    a=5; b=6; c=7; d=8;
    d = f(c, a, c);
    cout<<a<<b<<c<<d<<endl;
}
```

Что будет выведено в результате работы программы?

# Объявление и инициализация массива






# Структуры vs. массивы

```
struct Employee {  
    int age;  
    double salary;  
    string name  
}
```

```
struct TestRes {  
    int resSt1;  
    int resSt2;  
    int resSt3;  
    ...  
    int resSt265;  
}
```



Использование структуры для хранения большого количества однотипных данных **крайне неудобно**

## Структуры vs. массивы



**Массив** – совокупный тип данных, который позволяет получать доступ ко всем переменным одного типа через один и тот же идентификатор.

```
int resStudents[265];
```

Элементы массива индексируются с **0**.

# Размер фиксированного массива



```
int array[265];

const int size = 265;

int size1;
cin >> size1;
int array1[size1];

int x = 265;
const int size2 = x;
int array2[size2];
```

# Размер фиксированного массива

```
int array[265];  
  
const int size = 265;
```

```
int size1;  
cin >> size1;  
int array1[size1];
```

```
int x = 265;  
const int size2 = x;  
int array2[size2];
```

Ошибка компиляции!

Ошибка компиляции!

# Инициализация массива

```
int array1[19] = {-45, 36, -989};
```

```
int array2[64] = { };
```

```
int array3[] = {1, 1, 0, 1, 1};
```

# Инициализация массива

```
int array1[19] = {-45, 36, -989};  
  
int array2[64] = { };  
  
int array3[] = {1, 1, 0, 1, 1};
```

```
const n = 50;  
int array4[n] = { };  
  
for (size_t i = 0; i < n; i++) {  
    array[i] = ...;  
}
```

# Массивы в памяти и указатели

---

## Выделение памяти для массива

```
const int n = 150;  
int array[n] = { };  
  
cout << sizeof(array);  
600 (150 * sizeof(int))
```



## Выделение памяти для массива

```
const int n = 150;  
int array[n] = { };  
  
cout << sizeof(array);  
600 (150 * sizeof(int))
```

```
int array[n] = {1, 1, 0, 1, 1, 0, 1};  
cout << sizeof(array) / sizeof(array[0]);  
7
```

## Расположение массива в памяти

<ИМЯ\_массива> хранит адрес первого элемента массива.

```
const int n = 150;
int array[n] = {-1453, 225, 54, -3698, -7 };

cout << array << ' ' << &array[0] << '\n';
cout << *array;
```

00BAF750 00BAF750  
-1453

## Расположение массива в памяти

<ИМЯ\_массива> хранит адрес первого элемента массива.

```
const int n = 150;
int array[n] = {-1453, 225, 54, -3698, -7 };

cout << array << ' ' << &array[0] << '\n';
cout << *array;
```

00BAF750 00BAF750  
-1453

<ИМЯ\_массива> **не является указателем!**

## Расположение массива в памяти

Элементы массива хранятся в смежных ячейках памяти.

```
const int n = 3;  
int array[n] = {-1453, 225, 54};  
  
for (int i = 0; i < n; i++) {  
    cout << &array[i] << ' ';  
}
```

00B3F714 00B3F718 00B3F71C

	Память	
	...	...
array[0]	00B3F714	-1453
array[1]	00B3F718	225
array[2]	00B3F71C	54
	...	...

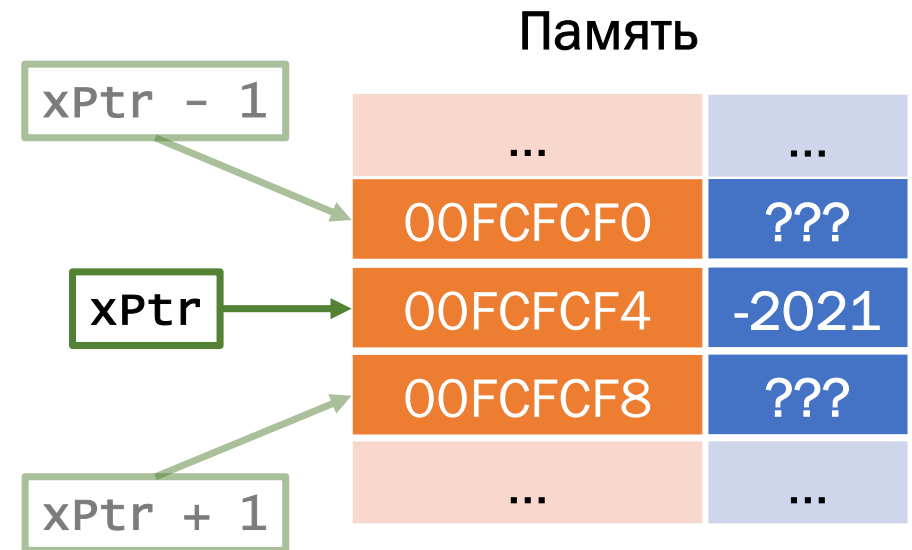
## Расположение массива в памяти

Указатели **допускают** выполнение арифметических операций над собой.

```
int x = -2021;  
int *xPtr = &x;
```

```
cout << xPtr << '\n';  
cout << xPtr + 1 << '\n';  
cout << xPtr - 1 << '\n';
```

```
00FCFCF4 00FCFCF8 00FCFCF0
```



# Расположение массива в памяти

## Итерация по массиву с помощью указателя

```
int numUnits = 0;
int array[] = { 1, 1, 0, 1, 1, 0, 1 };

int *last = array1 + sizeof(array) / sizeof(array[0]);

for (int *ptr = array1; ptr < last ; ++ptr)
{
    if (*ptr == 1) {
        numUnits = numUnits + 1;
    }
}
```

# Передача массивов в функции



## От массива остается только указатель...

```
#include <iostream>

void func(int array[]) {
    std::cout << sizeof(array);
}

int main() {
    int array[] = {1, 1, 0, 1, 1};
    func(array);
    std::cout << ' ' << sizeof(array);
}
```

4 20



## От массива остается только указатель...

### Передача размера массива в функцию

```
#include <iostream>

void func(int array[], int n) {
    for (int i = 0; i < n; i++) {
        array[i] = array[i] * 3;
    }
}

int main() {
    int array[] = {1, 1, 0, 1, 1};
    func(array, sizeof(array) / sizeof(array[0]));
    std::cout << array[3];
}

3
```

## От массива остается только указатель...

### Передача размера массива в функцию

```
#include <iostream>

void func(int *array, int n) {
    for (int i = 0; i < n; i++) {
        array[i] = array[i] * 3;
    }
}

int main() {
    int array[] = {1, 1, 0, 1, 1};
    func(array, sizeof(array) / sizeof(array[0]));
    std::cout << array[3];
}

3
```