ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Департамент программной инженерии

Алгоритмы и структуры данных

Семинар №2. 2021-2022 учебный год

Нестеров Роман Александрович, *ДПИ ФКН и НУЛ ПОИС*

Бессмертный Александр Игоревич, *ДПИ ФКН*

Higher School of Economics

Faculty of Computer science

Education is what survives when what has been learnt is forgotten

На прошлом семинаре

- Фундаментальные типы данных
- Программа – структура, компиляция, организация в памяти
- Многофайловые проекты – заголовочные файлы, пространства имен
- Ввод-вывод

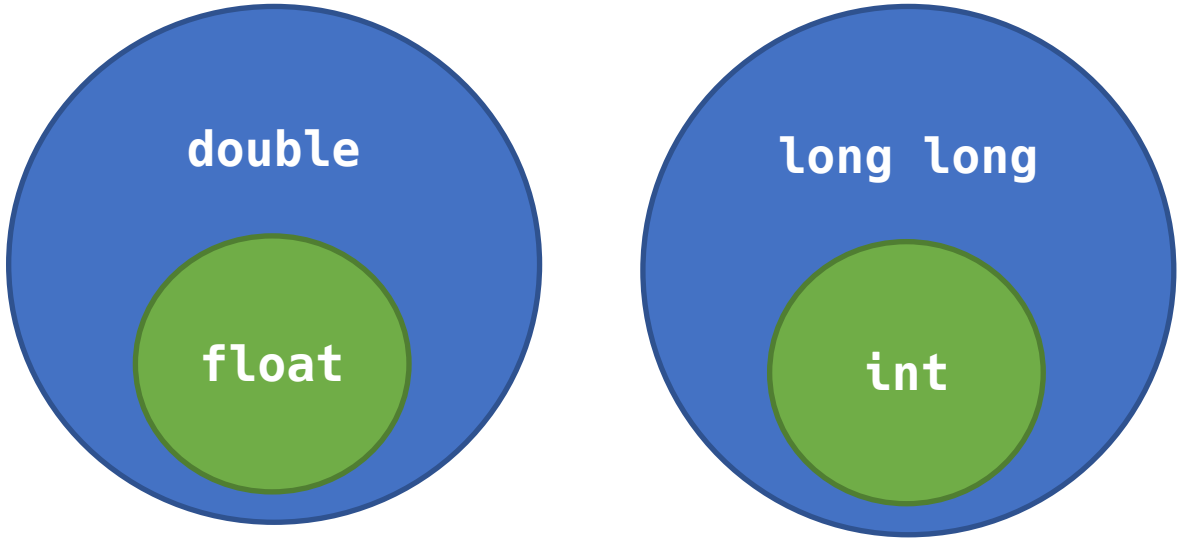
План

- Преобразование и вывод типов
- Перечисления и объединения
- Указатели и работа с динамической памятью
- Ссылки
- Функции – передача параметров, перегрузка, параметры по умолчанию

Преобразования и вывод типов

Преобразование типов

```
#include <iostream>

float f = 5;
double d = 6.63 /  4;

void doIt(long x) {
    std::cout << “I am here”;
    return;
}

int main() {
    doIt(4);
    return 0;
}
```



Неявное преобразование типов

Числовое расширение

```
long l = 5;

double d1 = 0.45f;
```

Всегда безопасно

Неявное преобразование типов

Числовое расширение

```
long l = 5;

double d1 = 0.45f;
```

Всегда безопасно

Числовая конверсия

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

int main() {
    double d2 = 567;
    short s1 = 145;
    int i1 = 15000;
    char c = i1;

    float f = 0.123456789;
    cout << setprecision(9) << f;
    return 0;
}
```

```
0.123456791
```

Неявное преобразование типов

long double

double

float

unsigned long long

long long

unsigned long

long

unsigned int

int

“Решающий” тип в выражении

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

int main() {
    double a = 5.0;
    short b = 1;
    cout << typeid(a+b).name() << endl;

    cout << 75u-120;
}
```

```
double
???
```

Явное преобразование типов

- C-style
- static_cast
- reinterpret_cast
- const_cast
- dynamic_cast

Явное преобразование типов

- **C-style**
- static_cast
- reinterpret_cast
- const_cast
- dynamic_cast

Непроверяемая конвертация типов

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

int main() {
    int v1 = 11;
    int v2 = 3;
    double x = (double)v1) / v2;

    cout << x;
    return 0;
}
```

```
3.66667
```

Явное преобразование типов

- C-style
- **static_cast**
- reinterpret_cast
- const_cast
- dynamic_cast

Проверяемая конвертация фундаментальных типов

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

int main() {
    char c = 120;
    cout << static_cast<int>(c);

    int v1 = 15;
    int v2 = 6;
    float x = static_cast<float>(v1) / v2;

    int i = 45;
    i = static_cast<int>(i / 3.77);
    return 0;
}
```

Вывод типов и auto

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

int sum(int a, int b) { return a + b; }

int main() {
    double x = 3.0;

    auto y = 5.6;
    auto z = 5 + 9;
    auto t = sum(6, 8);

    cout << typeid(y).name() << endl;
    cout << typeid(z).name() << endl;
    cout << typeid(t).name();
    return 0;
}
```

```
double
int
int
```

Вывод типов и `auto`

Вывод типа возвращаемого значения функции (не рекомендуется)

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

auto subtract(int a, int b) { return a - b; }

int main() {
    auto x = subtract(a, b);

    cout << typeid(x).name();
    return 0;
}
```

Q: Можно ли определить функцию `void f1(auto a, auto b) { … }`?

Перечисления и объединения

Перечисления enum

Ввод перечислителей

```
#include <iostream>

enum Colors {
    RED,
    GREEN
};

int main() {
    Colors color;
    std::cin >> color;

    int iCol;
    std::cin >> iCol;
    color = static_cast<Colors>(iCol);
    return 0;
}
```

Вывод перечислителей

```
#include <iostream>

enum Colors {
    RED,
    GREEN
};

int main() {
    Colors color = GREEN;

    switch (color) {
        case RED:
            std:cout << “red”; break;
        case GREEN:
            std::cout << “green”; break;
        default:
            std::cout << “unknown”; break;
    }
}
return 0;
}
```

Перечисления enum

Состояния при работе с файлом

```
enum Result
{
    SUCCESS,
    ERROR_OPENING,
    ERROR_READING
};
…
Result readFileContents()
{
    if (!openFile())
        return ERROR_OPENING;
    if (!readFile())
        return ERROR_READING;

    return SUCCESS;
}
```

- Документация кода
- Улучшение читаемости кода

Перечисления `enum`

- Перечисления относятся к целочисленным типам, т.е. обычно занимают столько же памяти, как и `int`
- Часто используемые перечисления можно вынести в заголовочный файл и подключать при необходимости

Перечисления с областью видимости `enum class`

Сравнение разных перечислителей

```
enum Fruits
{
    APPLE,
    KIWI
};

enum Colors
{
    RED,
    GREEN
};
```

```
Fruits fruit = KIWI;
Colors color = GREEN;

if (fruit == color)
    cout << "fruit and color are equal\n";
else
    cout << "fruit and color are not equal\n";
```

Перечисления с областью видимости `enum class`

Перечисления с ограниченной областью видимости

```
enum class Fruits
{
    APPLE,
    KIWI
};

enum class Colors
{
    RED,
    GREEN
};
```

```
Fruits fruit = KIWI;
Colors color = GREEN;

if (Fruits::fruit == Colors::color)
    cout << "fruit and color are equal\n";
else
    cout << "fruit and color are not equal\n";
```

Перечисления с областью видимости `enum class`

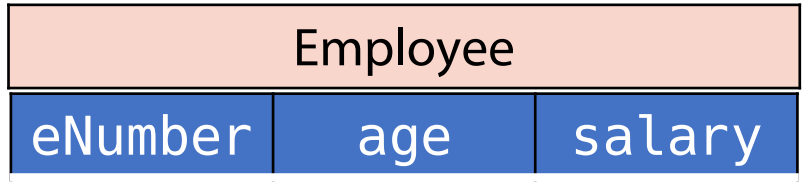
Перечисления с ограниченной областью видимости

```
enum class Fruits
{
    APPLE,
    KIWI
};

enum class Colors
{
    RED,
    GREEN
};
```

```
Fruits fruit = Fruits::KIWI;
Colors color = Colors::GREEN;

if (Fruits::fruit == Colors::color)
    cout << "fruit and color are equal\n";
else
    cout << "fruit and color are not equal\n";
```

Объединения union

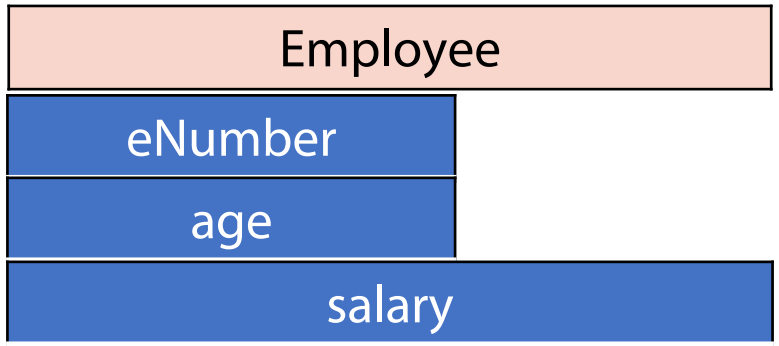
```
struct Employee {
    int eNumber;
    int age;
    double salary;
}
```



Итого памяти:

2*4 + 8 = 16 байт

```
union Employee {
    int eNumber;
    int age;
    double salary;
}
```

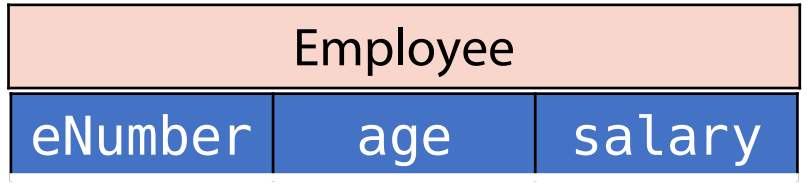


Итого памяти:

???

Объединения union

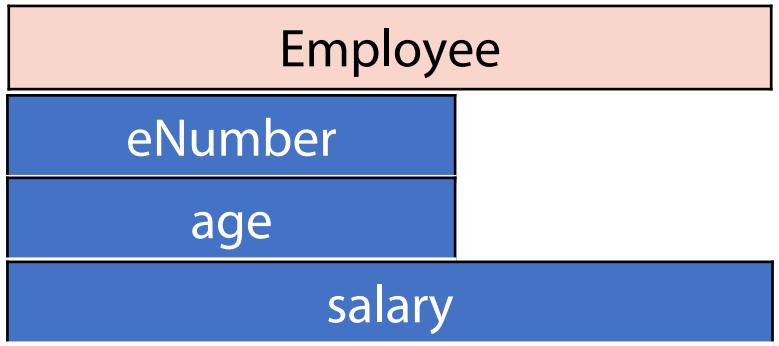
```
struct Employee {
    int eNumber;
    int age;
    double salary;
}
```



Итого памяти:

2*4 + 8 = 16 байт

```
union Employee {
    int eNumber;
    int age;
    double salary;
}
```

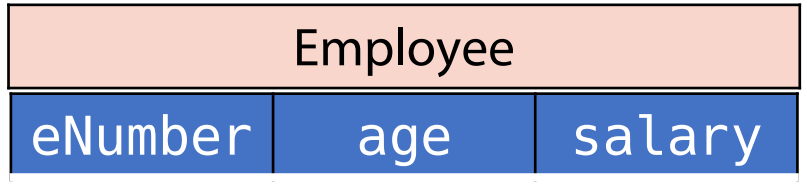


Итого памяти:

8 байт

Объединения union

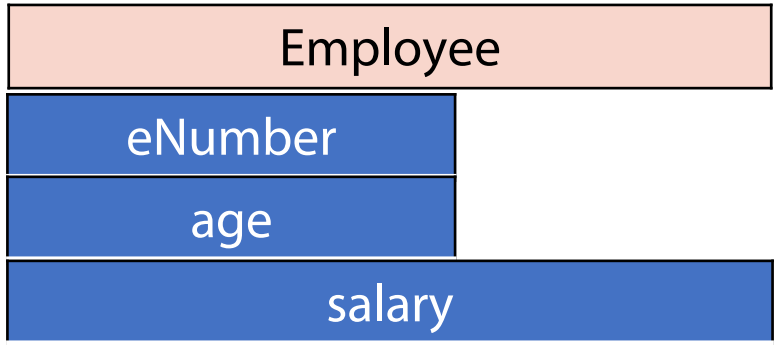
```
struct Employee {
    int eNumber;
    int age;
    double salary;
}
```



Итого памяти:

2*4 + 8 = 16 байт

```
union Employee {
    int eNumber;
    int age;
    double salary;
}
```



Итого памяти:

8 байт

Объединения union

```
union Employee {
    int eNumber;
    int age;
    double salary;
};
…
Employee emp;
emp.eNumber = 1234;
cout << emp.age << ‘ ’ << emp.salary;

emp.salary = 9876.56;
cout << emp.eNumber << ‘ ’ << emp.salary;
```

```
1234 -9.25596e+61
-1374389535 9876.56
```

- Экономия памяти
- Хранение данных разных типов по одному адресу

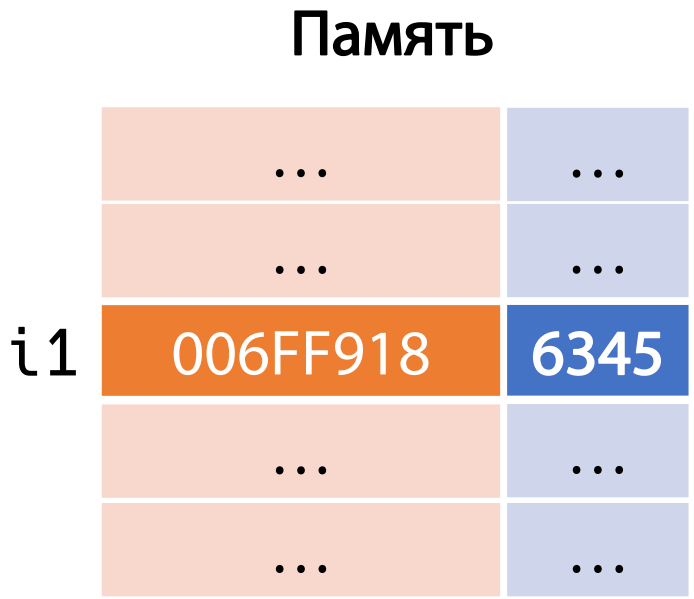
Указатели и динамическая память

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
```

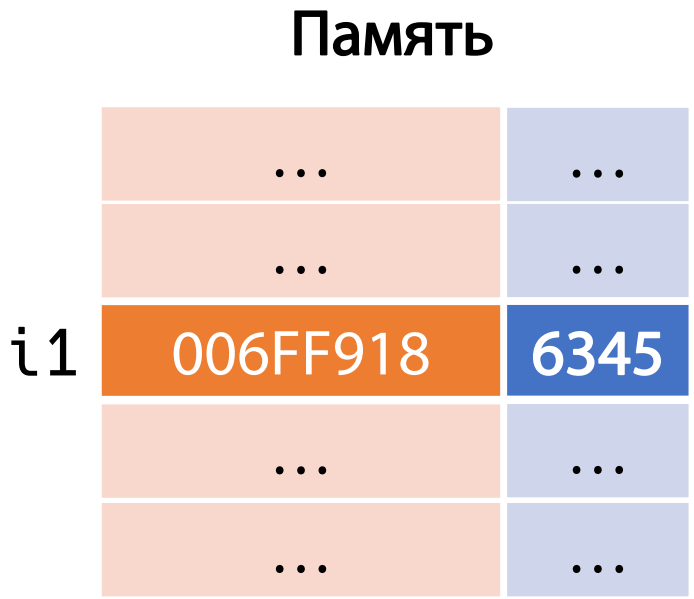


Указатели. Оператор разыменовывания *

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
```



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 *

Разыменование 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
- …

Память

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

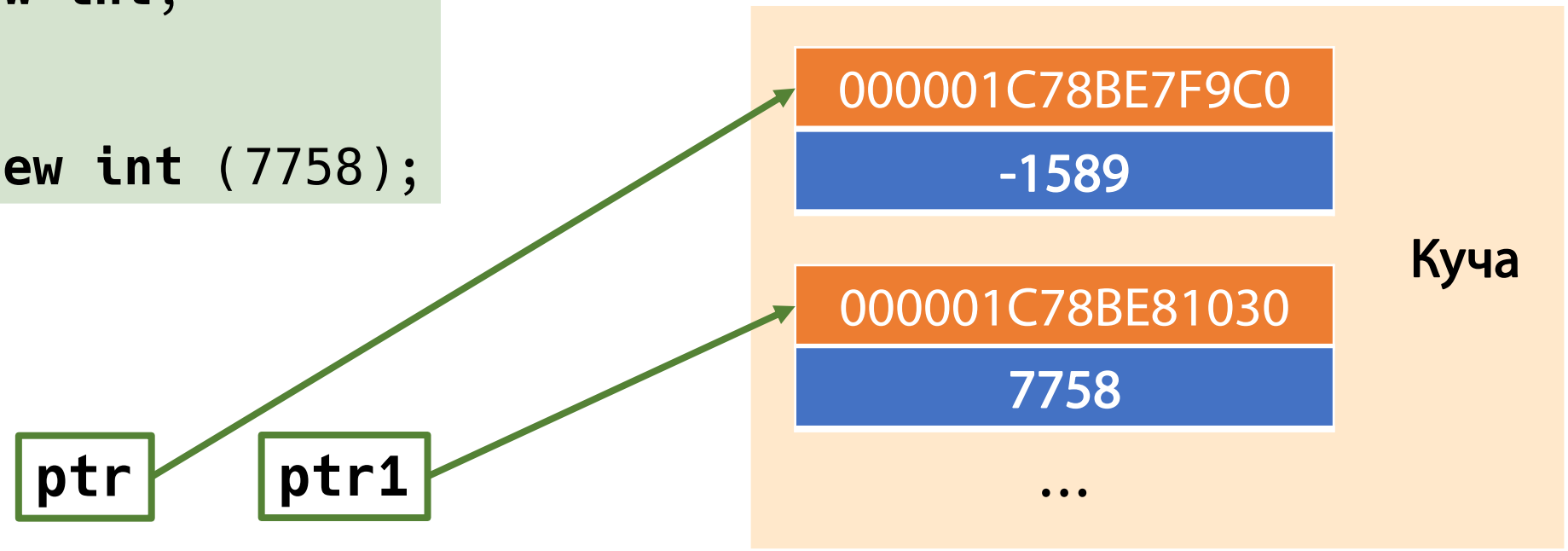
Память

- **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

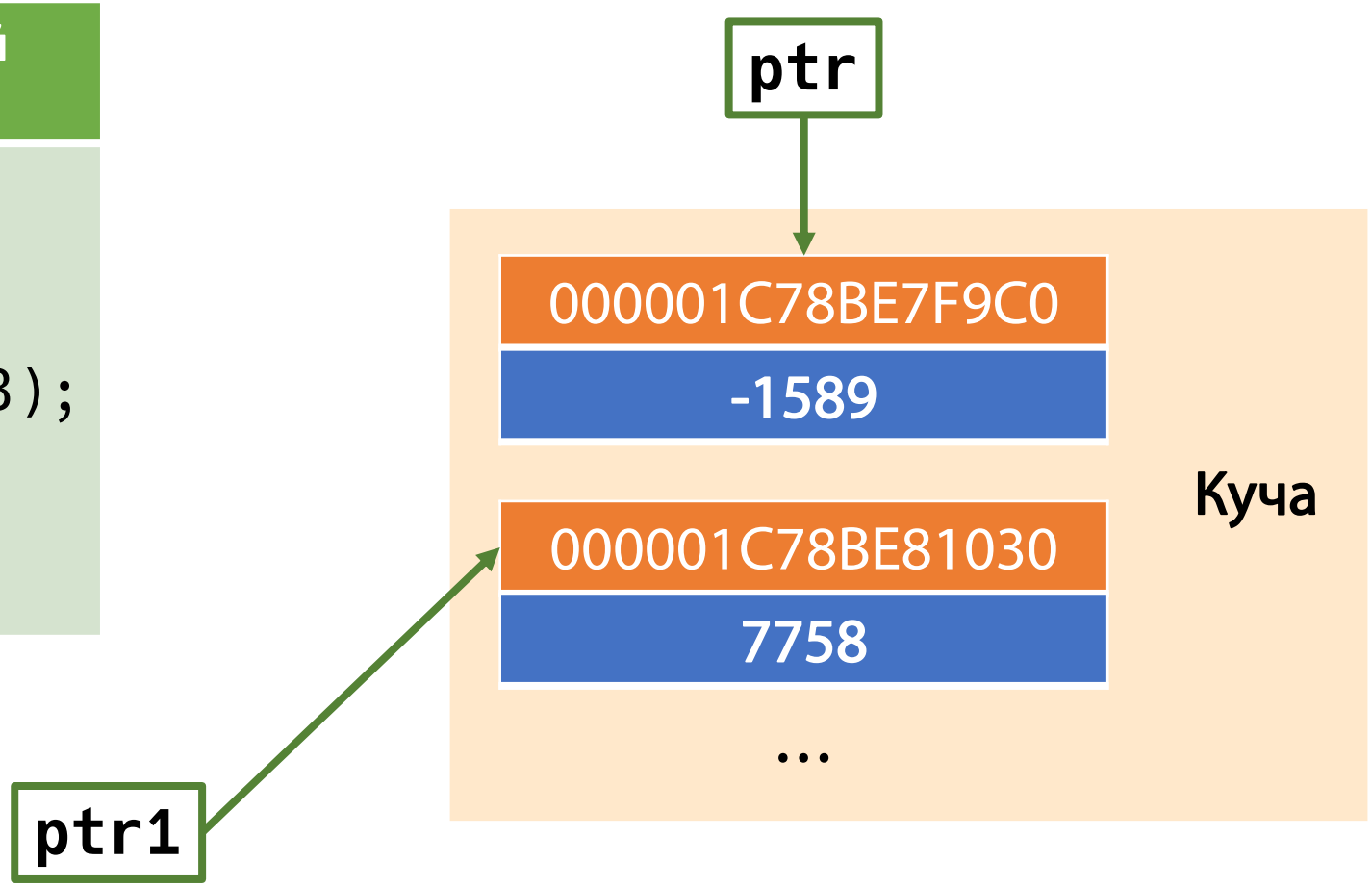
```
int *ptr = new int;
*ptr = -1589;

int *ptr1 = new int (7758);
```



Освобождение динамической памяти. Оператор `delete`

Освобождение ранее выделенной динамической памяти

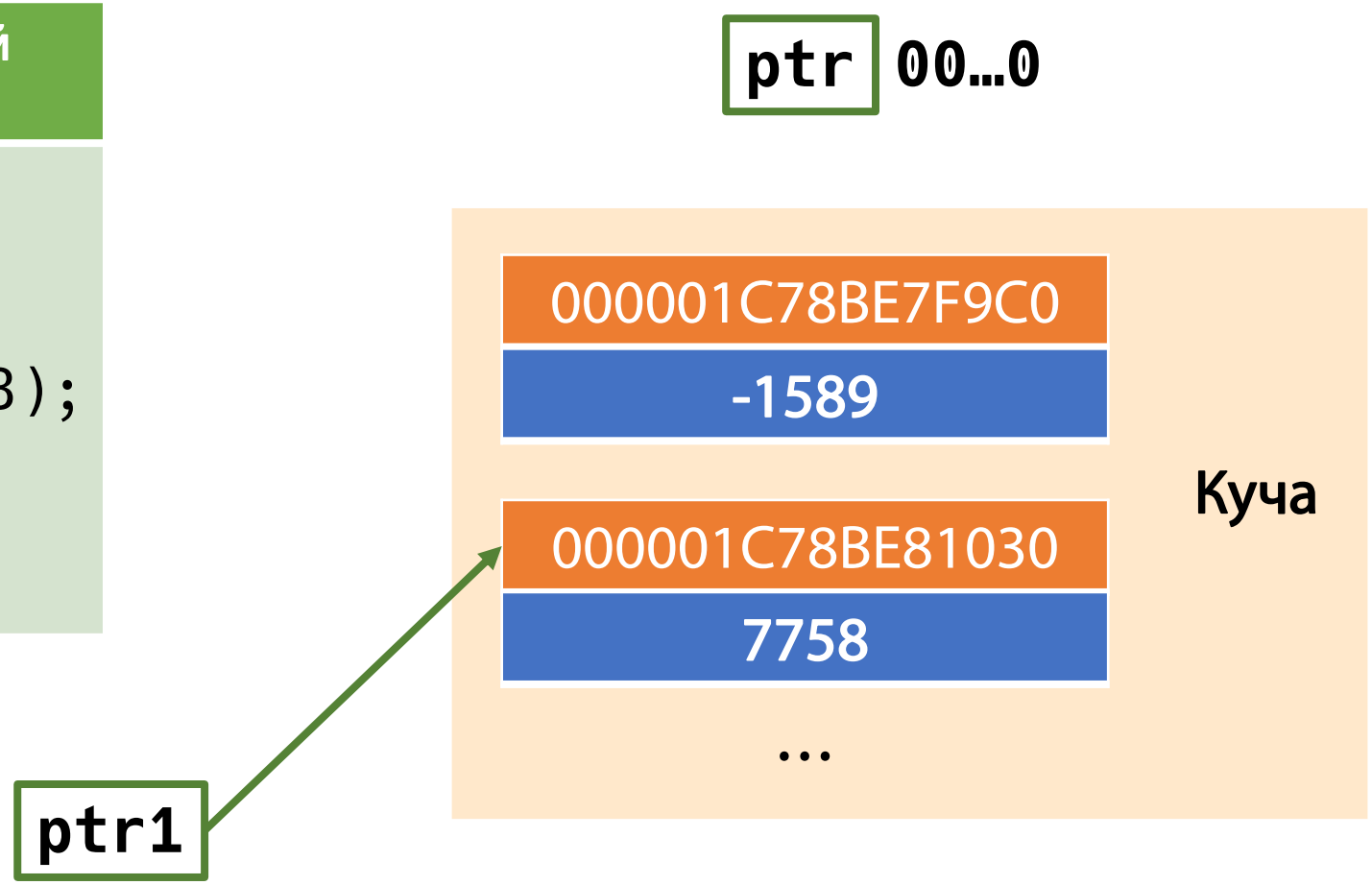
```
int *ptr = new int;
*ptr = -1589;

int *ptr1 = new int (7758);

delete ptr;
ptr = nullptr;
```



Освобождение динамической памяти. Оператор `delete`

Освобождение ранее выделенной динамической памяти

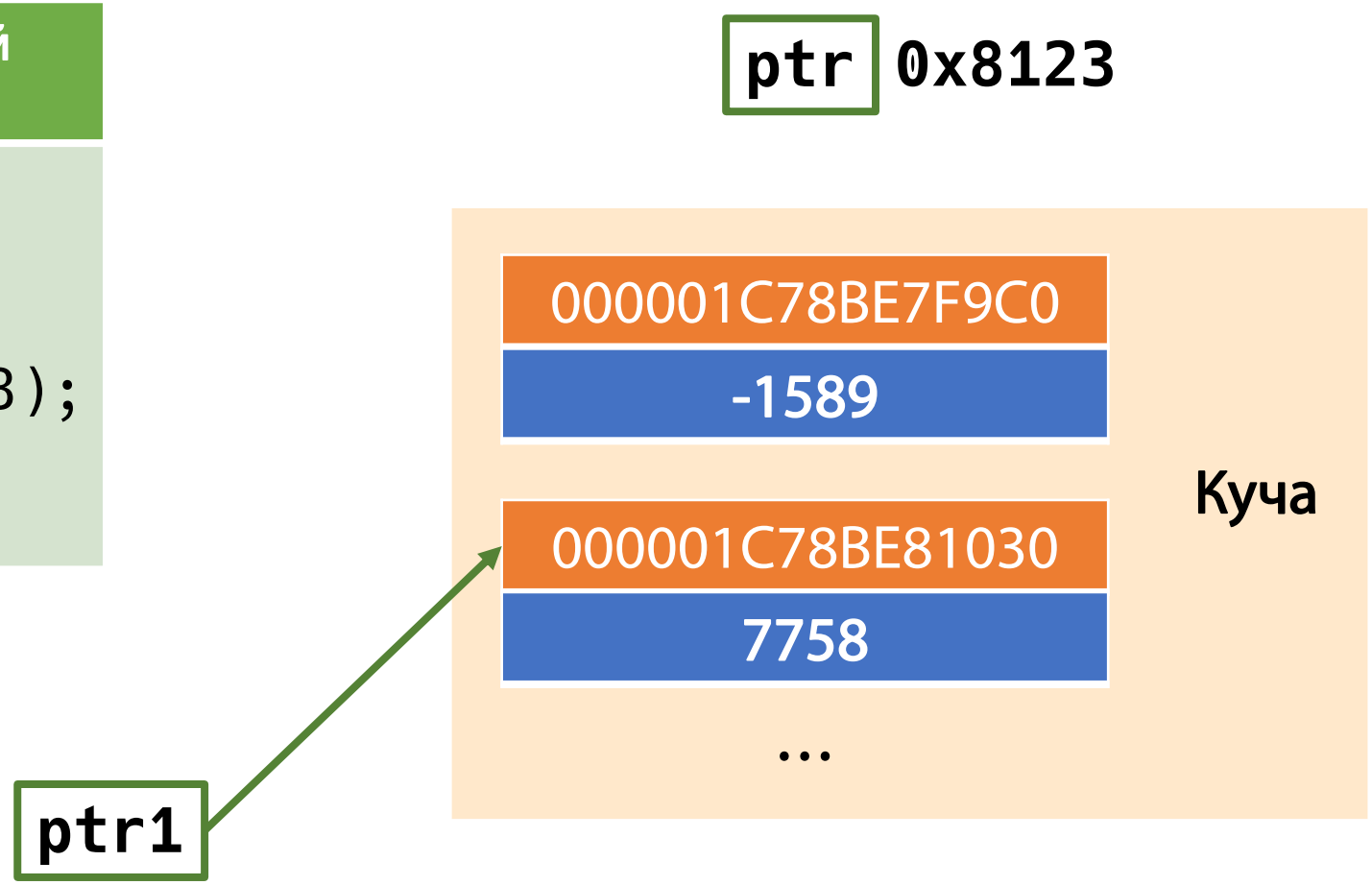
```
int *ptr = new int;
*ptr = -1589;

int *ptr1 = new int (7758);

delete ptr;
ptr = nullptr;
```



Висячий указатель. Оператор `delete`

Освобождение ранее выделенной динамической памяти

```
int *ptr = new int;
*ptr = -1589;

int *ptr1 = new int (7758);

delete ptr;
```



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

Ссылки в C++

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?

Перегрузка функций

Возможность определения нескольких функций с одинаковым именем, но с разными параметрами

```
int f(int a, int b)
{
    return a - b;
}
```

Перегрузка функций

Возможность определения нескольких функций с одинаковым именем, но с разными параметрами

```
int f(int a, int b)
{
    return a - b;
}
```

```
double fD(double a, double b)
{
    return a - b;
}
```

Перегрузка функций

Возможность определения нескольких функций с одинаковым именем, но с разными параметрами

```
int f(int a, int b)
{
    return a - b;
}
```

```
double f(double a, double b)
{
    return a - b;
}
```

Перегрузка функций

Возможность определения нескольких функций с одинаковым именем, но с разными параметрами

```
int f(int a, int b)
{
    return a - b;
}
```

```
double f(double a, double b)
{
    return a - b;
}
```

```
int f(int a, int b, int c)
{
    return a - b - c;
}
```

Перегрузка функций

Тип возврата **НЕ** учитывается при перегрузке функции

Перегрузка функций

Тип возврата **НЕ** учитывается при перегрузке функции

Некорректная попытка перегрузки функции!

```
int f(int a, int b)
{
    return a - b;
}
```

```
double f(int a, int b)
{
    return a - b;
}
```

Перегрузка функций

Вызов перегруженной функции сопряжен с поиском **соответствия** между аргументами и параметрами:

- Совпадение найдено
- Совпадение не найдено
- Найдено несколько вариантов совпадений

Перегрузка функций

Точное совпадение между аргументами и параметрами функции

```
void print(char *a) { … }
void print(int a) { … }
...
int x = 0;
print(x);
```

Перегрузка функций

Точное совпадение на найдено…

Неявные преобразования типов

```
void print(char *a) { … }
void print(int a) { … }
...
print('z');
```

- `char, short -> int`
- `float -> double`
- `enum -> int`

Перегрузка функций

Точное совпадение на найдено и неявные преобразования нет…
Стандартные преобразования типов

```
struct Employee { … };
void print(float a) { … }
void print(Employee a) { … }
...
print('z');
```

- `int -> float`, …
- `0 -> float`, `0 -> int *`
- `ptr * -> void *`
- `...`

Перегрузка функций

Точное совпадение на найдено и неявные преобразования нет…
Стандартные преобразования типов

```
struct Employee { … };
void print(float a) { … }
void print(Employee a) { … }
...
print('z');
```

- `int -> float`, …
- `0 -> float`, `0 -> int *`
- `ptr * -> void *`
- `...`

Стандартные преобразования имеют одинаковый приоритет

Перегрузка функций

Найдено **несколько совпадений**…

```
void print(float a);
void print(unsigned int a);
...
print('z');
print(3.1415);
print(0);
```

Ошибка компиляции!

Перегрузка функций

Найдено **несколько совпадений**…

```
void print(float a);
void print(unsigned int a);
...
print(static_cast<unsigned int>(0));
```

~~Ошибка компиляции!~~

Параметры функций по умолчанию

Параметр функции, который при вызове может быть опущен, так как имеет определенное (по умолчанию) значение

```
#include<iostream>

void print (int a, int b=5) {
    std::cout << "a = " << a << std::endl;
    std::cout << "b = " << b << std::endl;
}

int main() {
    printValues(1);
    printValues(6, 7);
    return 0;
}
```

```
a = 1
b = 5

a = 6
b = 7
```

Параметры функций по умолчанию

Параметр функции, который при вызове может быть опущен, так как имеет определенное (по умолчанию) значение

Все параметры по умолчанию в прототипе или в определении функции должны находиться **справа**!

```
void f(int a = 4, int b, int c = 7);

void f(int b, int a = 4, int c = 7);
```

Параметры функций по умолчанию

Сочетаются с перегрузкой функций

```
#include<iostream>
#include<string>

void print(std::string s) {
    std::cout << s;
}

void print(char c = ‘ ‘){
    std::cout << c;
}
```

```
int main( ) {
    print( );
    return 0;
}
```

Параметры функций по умолчанию

Сочетаются с перегрузкой функций

```
#include<iostream>
#include<string>

void print(int x) {
    std::cout << x;
}

void print(int x, int y = 6) {
    std::cout << x << y;
}
```

```
int main() {
    int p = 955;
    print(p);
    return 0;
}
```

Параметры функций по умолчанию

Сочетаются с перегрузкой функций

```
#include<iostream>
#include<string>

void print(int x) {
    std::cout << x;
}

void print(int x, int y = 6) {
    std::cout << x << y;
}
```

```
int main() {
    int p = 955;
    print(p);
    return 0;
}
```

Ошибка компиляции! Несколько совпадений