

13.04.2024

Приведения типов в стиле C++

Спикер: Ибрагимов Булат Ленарович

Fast Track в Телеком, 2024

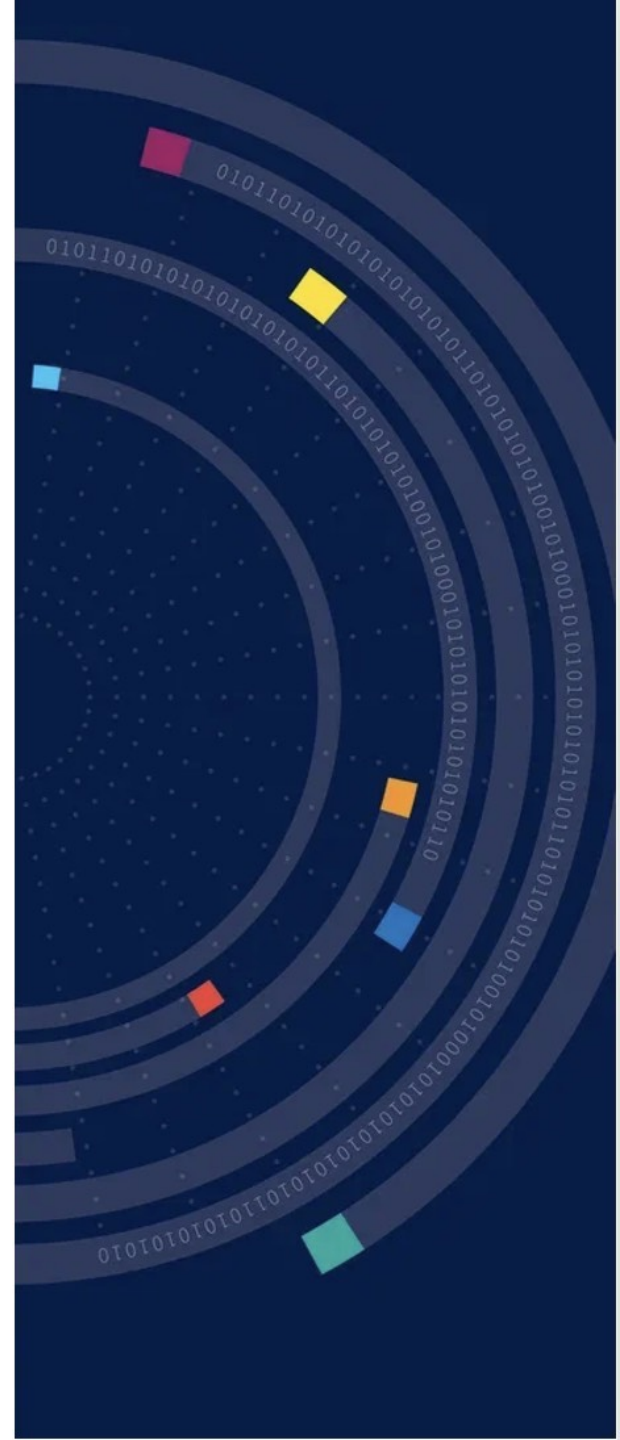




ИБРАГИМОВ БУЛАТ ЛЕНАРОВИЧ

- Преподаватель в МФТИ. Проводит курсы по С++ и алгоритмам, структурам данных
- Научный сотрудник Института Искусственного Интеллекта (AIRI)
- Работал разработчиком-исследователем в Яндекс и Сбербанк

Приведения типов в стиле C++



Приведение типов в стиле C

Чтобы результат выражения был приведен к требуемому типу (это бывает необходимо, например, для вызова требуемой перегрузки функции или шаблон) можно воспользоваться приведением типов, доставшимся нам в наследство из языка C:

- `(<требуемый тип>)<выражение>` - классическое приведение в стиле C
- `<требуемый тип>(<выражение>)` - функциональное приведение (только в C++)

```
void f(int);  
void f(double);  
  
template <class T>  
void g(T);  
// ...  
double x = 0.0;  
f((int)x);    // void f(int)  
g(int(x));    // void g<int>(int)
```

Приведение типов в стиле C

C-style приведение - самый агрессивный вид приведения, который существует в языке C++. Помимо "безопасных" приведений оно может снимать константность (в том числе со ссылок и элементов под указателями!) и менять интерпретацию битов по определенному месту в памяти.

```
const int cx = 11;
const int* pcx = &cx;
const int& rcx = cx;

int* px = (int*)pcx;    // ok (!!!)
int& rx = (int&)rcx;    // ok (!!!)
*px = 25;               // UB
rx = 34;                // UB

std::cout << *((float*)pcx); // pcx будет разыменован как float!
```

`static_cast`



static_cast

`static_cast` осуществляет допустимые в С++ явные преобразования типов (с учетом константности)

Синтаксис: `static_cast< <тип> >(<выражение>)`

- `<тип>` - целевой тип
- `<выражение>` - значение, которое нужно преобразовать

Корректность преобразований проверяется на этапе компиляции!

static_cast

```
int f(int);
float f(float);

int x = 0;

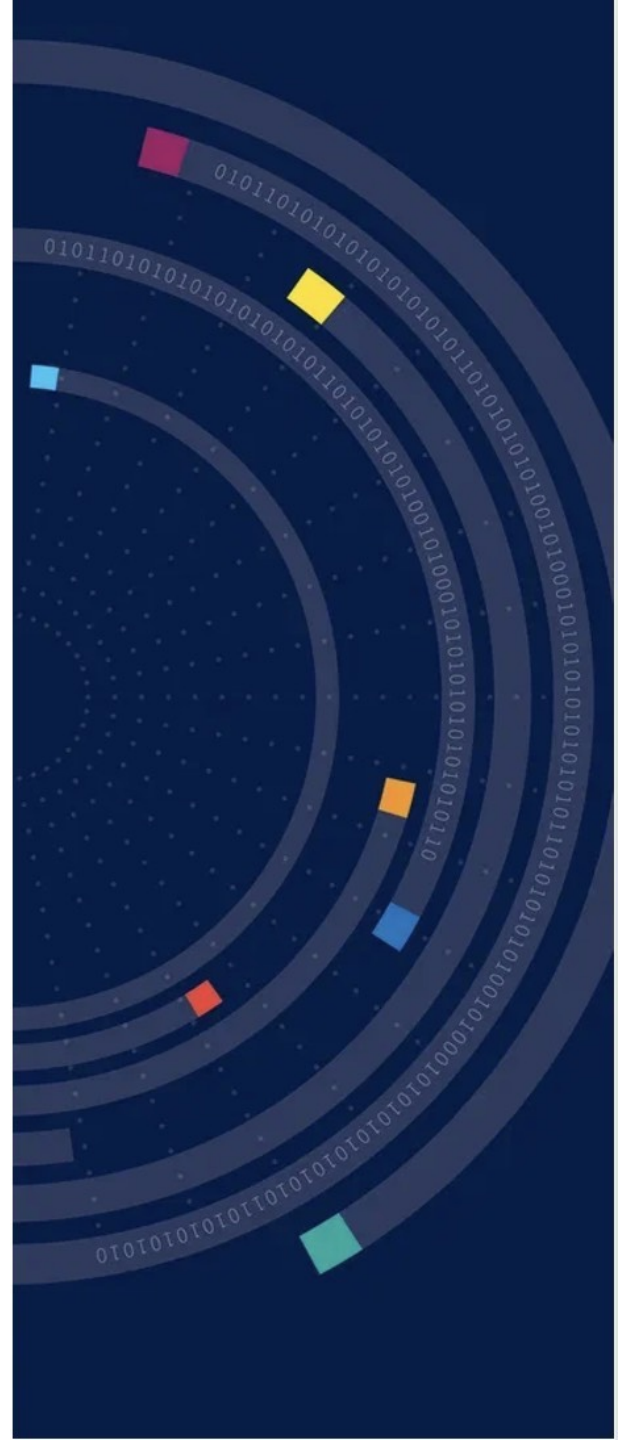
// Преобразования между числовыми типами
auto x_float = static_cast<float>(x);           // type(x_float) == float
f(static_cast<int>(x_float));                   // int f(int)

// Пользовательские преобразования
auto stack = static_cast<Stack>(5);             // явно используется к-р преобразования

// Преобразования из void* в указатель на тип и наоборот
void* ptr = std::malloc(sizeof(int));
auto int_ptr = static_cast<int*>(ptr);           // int*
std::free(static_cast<void*>(int_ptr));

// Выбор перегрузки функции
auto f_ptr = static_cast<int(*)>(f);            // указатель на int f(int)
static_cast<float(*)>(f)(x);                    // float f(float)
```


`const_cast`



const_cast

Позволяет снять/добавить квалификаторы `const` и `volatile`

Синтаксис: `const_cast< <тип> >(<выражение>)`

- `<выражение>` - возвращает значение, которое можно проинтерпретировать как указатель или ссылку
- `<тип>` - совпадает с типом `<выражение>` с точностью до `const` и `volatile`

Корректность проверяется на этапе компиляции

const_cast

```
int x = 0;
```

```
// Относительно безопасные преобразования
```

```
auto& x_cref = const_cast<const int&>(x); // type(x_cref) == const int&  
auto& x_ref = const_cast<int&>(x_cref); // type(x_ref) == int&  
auto x_cptr = const_cast<const int*>(&x); // type(x_cptr) == const int*  
auto x_ptr = const_cast<int*>(x_cptr); // type(x_ptr) == int*
```

```
const int cx = 0;  
const char* c_str = "string";
```

```
// Опасные преобразования
```

```
auto& cx_ref = const_cast<int&>(cx); // type(cx_ref) == int&  
auto cx_ptr = const_cast<int*>(&cx); // type(cx_ptr) == int*  
auto str = const_cast<char*>(c_str); // type(str) == char*
```

`reinterpret_cast`



reinterpret_cast

Позволяет изменить интерпретацию битов в памяти

Синтаксис: `reinterpret_cast< <тип> >(<выражение>)`

- `<выражение>` - должно иметь целый тип, тип указателя или ссылки
- `<тип>` - целый тип или указатель. Если `<выражение>` - ссылка (не временное значение), то `<тип>` тоже может быть ссылкой

Корректность проверяется на этапе компиляции

reinterpret_cast

```
uint64_t ux = 11;

std::cout << reinterpret_cast<float*>(ux) << '\n';    // 0xb
std::cout << *reinterpret_cast<double*>(&ux) << '\n'; // 5.43472e-323

int32_t sx = -123;
std::cout << reinterpret_cast<uint32_t*>(sx) << '\n'; // 4294967173
```



Резюме

- Пользуйтесь `static_cast` для разрешенных преобразований типов, для неразрешенных преобразований воспользуйтесь отказом от них.