Программирование на языке C++
Лекция 7

Преобразование в стиле C++

Александр Смаль

Преобразование в стиле C

В C этот оператор преобразует встроенные типы и указатели.

```
int a = 2;
int b = 3;

// int → double
double size = ((double)a) / b * 100;

// double → int
void * data = malloc(sizeof(double) * int(size));

// void * → double *
double * array = (double *)data;

// double * → char *
char * bytes = (char *)array;
```

Преобразования в C++: `static_cast`

Служит для преобразований связанных типов:

Преобразования в C++: `static_cast`

Служит для преобразований связанных типов:

- Стандартные преобразования.
  - Преобразования числовых типов.

```
double s = static_cast<double>(2) / 3 * 100;
s = static_cast<int>(d);
```

  - Указатель/ссылка на производный класс в указатель/ссылку на базовый класс.
  - `T*` в `void*`.

Преобразования в C++: `static_cast`

Служит для преобразований связанных типов:

- Стандартные преобразования.
  - Преобразования числовых типов.

```
double s = static_cast<double>(2) / 3 * 100;
s = static_cast<int>(d);
```

  - Указатель/ссылка на производный класс в указатель/ссылку на базовый класс.
  - `T*` в `void*`.
- Явное (пользовательское) приведение типа:

```
Person p = static_cast<Person>("Ivan");
```

Преобразования в C++: `static_cast`

Служит для преобразований связанных типов:

- Стандартные преобразования.
  - Преобразования числовых типов.

```
double s = static_cast<double>(2) / 3 * 100;
s = static_cast<int>(d);
```

  - Указатель/ссылка на производный класс в указатель/ссылку на базовый класс.
  - `T*` в `void*`.
- Явное (пользовательское) приведение типа:

```
Person p = static_cast<Person>("Ivan");
```

- Обратные варианты стандартных преобразований:
  - Указатель/ссылка на базовый класс в указатель/ссылку на производный класс (преобразование вниз, downcast),
  - `void*` в любой `T*`.

Преобразования в C++: `static_cast`

Служит для преобразований связанных типов:

- Стандартные преобразования.
  - Преобразования числовых типов.

```
double s = static_cast<double>(2) / 3 * 100;
s = static_cast<int>(d);
```

  - Указатель/ссылка на производный класс в указатель/ссылку на базовый класс.
  - `T*` в `void*`.
- Явное (пользовательское) приведение типа:

```
Person p = static_cast<Person>("Ivan");
```

- Обратные варианты стандартных преобразований:
  - Указатель/ссылка на базовый класс в указатель/ссылку на производный класс (преобразование вниз, downcast),
  - `void*` в любой `T*`.
- Преобразование к `void`.

Преобразования в C++: `const_cast`

Служит для снятия/добавления константности.

Преобразования в C++: const_cast

Служит для снятия/добавления константности.

```
void foo(double const& d) {
    const_cast<double &>(d) = 10;
}
```

Использование const_cast — признак плохого дизайна.

Преобразования в C++: const_cast

Служит для снятия/добавления константности.

```
    void foo(double const& d) {
        const_cast<double &>(d) = 10;
    }
```

Использование const_cast — признак плохого дизайна.

Кроме редких исключений:

```
T & operator[](size_t i) {
    return const_cast<T &>(
            const_cast<Vector const &>(*this)[i]);
}

T const & operator[](size_t i) const {
    assert(i < size_);
    return data_[i];
}
```

Преобразования в C++: `reinterpret_cast`

Служит для преобразований указателей и ссылок на несвязанные типы.

Преобразования в C++: `reinterpret_cast`

Служит для преобразований указателей и ссылок на несвязанные типы.

```
void send(char const * data, size_t length);
char * receive(size_t * length);
```

Преобразования в C++: `reinterpret_cast`

Служит для преобразований указателей и ссылок на несвязанные типы.

```
void send(char const * data, size_t length);
char * receive(size_t * length);
```

```
double * m = static_cast<double*>
                    (malloc(sizeof(double) * 100));
... // инициализация m
char * mc = reinterpret_cast<char *>(m);
send(mc, sizeof(double) * 100);
```

Преобразования в C++: reinterpret_cast

Служит для преобразований указателей и ссылок на несвязанные типы.

```
void send(char const * data, size_t length);
char * receive(size_t * length);
```

```
double * m = static_cast<double*>
                    (malloc(sizeof(double) * 100));
... // инициализация m
char * mc = reinterpret_cast<char *>(m);
send(mc, sizeof(double) * 100);
```

```
size_t length = 0;
double * m = reinterpret_cast<double*>
                                (receive(&length));
length = length / sizeof(double);
```

Преобразования в C++: reinterpret_cast

Служит для преобразований указателей и ссылок на несвязанные типы.

```
void send(char const * data, size_t length);
char * receive(size_t * length);
```

```
double * m = static_cast<double*>
                    (malloc(sizeof(double) * 100));
... // инициализация m
char * mc = reinterpret_cast<char *>(m);
send(mc, sizeof(double) * 100);
```

```
size_t length = 0;
double * m = reinterpret_cast<double*>
                                (receive(&length));
length = length / sizeof(double);
```

Поможет преобразовать указатель в число.

```
size_t ms = reinterpret_cast<size_t>(m);
```

Границы применимости преобразования в стиле C

Границы применимости преобразования в стиле C

- Преобразования в стиле C может заменить любое из рассмотренных преобразований:
  - `static_cast`,
  - `reinterpret_cast`,
  - `const_cast`.

Границы применимости преобразования в стиле C

- Преобразования в стиле C может заменить любое из рассмотренных преобразований:
  - `static_cast`,
  - `reinterpret_cast`,
  - `const_cast`.
- Преобразования в стиле C можно использовать для
  - преобразование встроенных типов,
  - преобразование указателей на явные типы.

Границы применимости преобразования в стиле C

- Преобразования в стиле C может заменить любое из рассмотренных преобразований:
  - `static_cast`,
  - `reinterpret_cast`,
  - `const_cast`.
- Преобразования в стиле C можно использовать для
  - преобразование встроенных типов,
  - преобразование указателей на явные типы.
- Преобразования в стиле C не стоит использовать:
  - с пользовательскими типами и указателями на них,
  - в шаблонах.

Когда преобразование в стиле C приводит к ошибке

```
// abc.h
struct A { int a; };

struct B {};

struct C : A, B {};
```

Когда преобразование в стиле C приводит к ошибке

```
// abc.h
struct A { int a; };

struct B {};

struct C : A, B {};
```

```
#include "abc.h"

C * foo(B * b) {
    return (C *)b;
}
```

Когда преобразование в стиле C приводит к ошибке

```
// abc.h
struct A { int a; };

struct B {};

struct C : A, B {};
```

```
#include "abc.h"

C * foo(B * b) {
    return (C *)b;
}
```

```
struct A; struct B; struct C;

C * foo(B * b) {
    return (C *)b;
}
```

Когда преобразование в стиле C приводит к ошибке

```
// abc.h
struct A { int a; };

struct B {};

struct C : A, B {};
```

```
#include "abc.h"

C * foo(B * b) {
    return (C *)b;
}
```

Если в этой точке известны определения классов, то происходит преобразование `static_cast`.

```
struct A; struct B; struct C;

C * foo(B * b) {
    return (C *)b;
}
```

Если известны только объявления, то происходит преобразование `reinterpret_cast`.