Билет 4.2. Разница между понятиями operator precedence и order of evaluation. Понятие unspecified behaviour, примеры. Несколько примеров выражений, вычисление которых является unspecified behaviour (с объяснением). Примеры выражений, вычисление которых является undefined behaviour.

Operator precedence vs. order of evaluation

На порядок вычисления значений функций, в отличие от порядка выполнения операций, стандартом C++ никаких требований не наложено, и порядок вычисления значений функций является неспецифицированным.

Unspecified behaviour, примеры

Def. Unspecified behaviour — неустановленное поведение, поведение для корректной программы и данных, зависящее от компилятора.

Пример:

```
#include <iostream>

int f() {
    std::cout << 1;
    return 2;
}

int g() {
    std::cout << 2;
    return 3;
}

int main() {
    f() + g();
    return 0;
}
```

Выведется 12, ясно, что нужно сложить f() + g(), но стандарт не предписывает, что именно нужно сначала вычислить: f() или g().

Аналогичные примеры: f() + g() * g(), f(g(), h()).

Один из примеров выражений, приводящих к UB

```
int i = 55;
i = ++i + i++;
```

Билет 4.3. Приведения типов: *C − style cast*, *static_cast*, *const_cast* и *reinterpret_cast*. В каких ситуациях (кроме наследования) каждый из этих операторов приводит к ошибке компиляции, в каких - к неопределенному поведению, а в каких он уместен?

static cast: принимает решение на этапе компиляции и может, в частности,

- Вызвать конструктор или определённый пользователем оператор преобразования типа — в частности, помеченный как explicit
- Преобразовать тип указателя или ссылки в случае наследования и ряде других
- Использовать стандартное преобразование типа.

Это стандартное преобразование между типами, которое вы ожидаете. В частности указатели несовместимых типов, как например int и double, не переводятся друг в друга — CE, а переполнение — это UB.

```
int x = 42;
*static_cast<double*>(&x); //CE
long long y = 24;
static_cast<int>(y); //UB
```

reinterpret cast: "более топорно меняет тип выражения. Не выполняет никаких дополнительных операций в рантайме. Разрешаются любые преобразования указателей, не понижающие константность."

Эта штука тупо считает кусок памяти начинающийся с некоторого момента куском памяти другого типа, в частности если int кастануть к double и спросить что там лежит — будет UB, так как залезли не в свою память. Есть легкая и тяжелая форма каста: через указатель и через ссылку соответственно.

Легкая:

```
int x = 42;
std::cout << *reinterpret_cast<double*>(&x); //UB
```

Тяжелая:

```
double d = 3.14;
std::cout << std::hex << reinterpret_cast<int&>(d); //output  eb51851f побитоое расположение в памяти
```

Тут мы урезали кусок памяти с дабла до инта. Тяжелый каст — первое запрщенное заклинание курса, так как он слишком опасен.

```
double d = 3.14;
reinterpret_cast<int>(d); //CE
```

```
const int &const_iref = i;
//int &iref = reinterpret_cast<int&>(const_iref); //compiler error - can't get rid of const
//Must use const_cast instead: int &iref = const_cast<int&>(const_iref);
```

const_cast: говорим компилятору, что на самом деле объект, который мы хотим поменять неконстантый, и если он действительно такой, то компилятор пускает

```
int x = 5;
const int& y = x;
int& z = const_cast<int&>(y);
z = 10;
std::cout << y; //cout « 10
```

а если нет, то это UB

```
const int cx = 1;
int& x = const_cast<int&>(cx);
x = 2;
std::cout << cx; //cout « 1; UB
```

Вроде как такой эффект достигается тем, что константы лежат где-то в специальной части памяти или же тут же подставляются, но мы, вроде, этого не обсуждали на лекциях.

```
double d = 3.14;
const_cast<double>(d); //CE
```

```
[[maybe_unused]]
void (type::* pmf)(int) const = &type::f; // pointer to member function
// const_cast<void(type::*)(int)>(pmf); // compile error: const_cast does
                                        // not work on function pointer
```

C-style cast: жуткий каст, который пытается использовать вспе подряд, пока не сработает С лекции:"Стандарт говорит: сначала пытается сделать const cast, если не сработает, то пытается сделать static, если опять нет, то пытается как static + const, затем пытается reinterpret, затем reinterpret + const. Поэтому если даже он сработает — вы не будете знать как именно"

```
int x = 0;
double d = (double)x;
long long y;
(int)y; //UB, так как отработает как static_cast
struct A{};
struct B{};
B b;
(A)b; //CE, так как ни один из кастов не возможен
```