extra4. Выражения свертки (fold expressions), их синтаксис и принцип работы. Реализация функции print с помощью fold-expressions. Реализация структуры is_homogeneous с помощью fold expressions. Автоматический вывод типа для шаблонов классов в C++17, пример с вектором. Явное задание правил вывода (deduction guides) для шаблонов классов в C++17, примеры.

Fold expressions

Для всех элементов пакета можно сделать операцию

```
template <typename T>
void print(const Args&... args) {
  (std::cout << ... << args) << '\n';
}
```

Можно писать следующее `( ... < args );` где вместо $<$ любой бинарный оператор. Это Fold expression, у него есть 4 типа:

`( ... < args );` левоассоциативно

`( args < ... );` правоассоциативно

`( x < ... < args );` лево и еще аргумент в начале

`( args < ... < x );` право и аргумент в конце

```
template <typename Hed, typename... Tail>
struct is_homogeneous {
  static const bool value  (std::is_same_v<Head, Tail> && ...);
};
```

Компилятор проверяет, что Head равен каждому элементу Tail

Deduction guides

```
template <typename T>
void f(T x){

}

int main(){
  int x = 0;
  int& y = x;
  f(y);
}
```

Так как я пишу `f(T x)`, то компилятор захочет принимать по значению

Как кстати проверить что `T` это `int` а не `int&`? Можно сделать `T z = 5;` или `sizeof(T)` или `std::is_reference_v` (в библиотеке `type_traits`)

```
template <typename T>
class C {
  C() = delete;
};

template <typename T>
void f(T x){
  C<T>();
}
```

А можно так, тут будет конечно CE потому что в compile-time можно узнать про T, потому что вызвется ошибка. В обычной ситуации `const` тоже отброситься, но если сделать `void f(T& x)` то `f(const int&)` сохранит `const`. Если надо вызвать от конкретного типа то так можно и писать `f<int&>(x)`

```
#include <functionals>

int x = 0;
f(std::ref(x));
```

Это такая обертка, считается что это типо ссылка, `reference_wrapper` (но ему нельзя присваивать старые значения(для этого нужно написать `x.get()`), а можно другой `reference_wrapper`). Но у него нет конструктора по умолчанию и нет операций как у указателя.. Но при этом можно сделать ветокр от него (в отличие от ссылки)

Можно с вектором еще сделать так: `std::vector v{1, 2, 3, 4, 5};`

```
template<typename T>
    struct S {
        S(T x) {
            C<T>();
        }
  };

S(const char*) -> S<std::string>;
```

Это user-defined deduction rule, если вызвать `S("abc")` то он будет воспринимать это как строку. При этом deduction guides могут быть и шаблонными