template <typename T>
func(T&& x) → forwarding Ref
Universal

template <typename T>
class Abdo{
func(T&& var)} → not forwarding Ref

auto&& x=10 → forwarding Ref

Reference Collapsing

T& &
T& &&
T&& & ⟩ T&

T&& && → T&&

Reference Collapsing May happen at:

1. func(T&& val)
2. using LREf = MyClass&
   LREf &
3. decltype(x)& val=5
   int& &=int&

Instead of static_cast<T&&>(val
std::move is implemented, better name move_cast

std::array<int,500> a → move does not help here
everything is stack allocated.

Copy Elision is better/efficient than move.

When PRvalue is converted to objelt it is called temporery materiolization.

auto s=string{"Abdu"}
PRVal → getting materialized

string str(100000,'A')
StrVec svec;
svec.push_back(str)
str → is in a moved from state, should be in valid state, generally not used
But there are cases. .size() must return 0, invariants must hold for std containers.
Moved from State is not same with defaul constructed state,
however for stl it generally holds, as it is more efficient

Special Member Functions

↳ Constructor, Destructor, Copy/Move Constructor/Assignment Operator.

↳ They can be not declared or user Declared or implicitly declared

If any of copy functions or destructor is user declared then move members are not declared

MyClass(const MyClass&)=delete ⇒ User declared but deleted

MyClass& operator=(const MyClass&)=delete → cant be copied, if moves are not user declared then they are not written, so cant be moved as well

If move members are declared then copy members are deleted

user declared → defined
→ default
→ delete

We should never delete move members.

↳ if cant be moved, fallbacked to copy anyway.

Most of the time defaul constructor must exist, otherwise it limits usage (with containers etc)

in hpp ~MyClass;

in cpp MyClass::~MyClass() = default;

> Valid and defaulted, needed for using pimpl idiom with unique-ptr

If a class const or reference variable then default constructor is deleted.

implicitly declared → defaulted
→ deleted

Compiler decides for noexcept depending on member variables default constructor
constructor's

void func(int x) noexcept; → func guarantees that it wont throw
noexcept(true);

void func(T x) noexcept(is_no_throw_copy_constructible_v<T>);

void func(T x) noexcept(noexcept(x+x));
↳ unevaluated context

Destructor is always defaulted, unless user declares it.
If default constructor is user declared, everything else is defaulted.
If destructor is user declared, move's are not declared, others defaulted → bad for Copy
If copy constructor is user declared, ↓, constructor not declared, destructor + copy(=) → defaulted
If " assignment is user declared, ↓, " destructor, copy constr. defaulted.
If move constructor is declared, copy members are deleted, constructor not declared, destructor defaulted
If move assignment is declared, ↓, constructor destructor defaulted, move const. not declared
↳ move assignment not declared.

MyClass Obj; → default init
MyClass Obj{}; → value init, first step is zero init

When PR value inits a variable there is no copy (was diffrent befor C++17)

Pr value can be materialized when init something, or when it is discarded