function template

std::forward

<utility>

```
lvalue (1)  template <class T> T&& forward (typename remove_reference<T>::type& arg) noexcept;
rvalue (2)  template <class T> T&& forward (typename remove_reference<T>::type&& arg) noexcept;
```

Forward argument

Returns an *rvalue reference* to *arg* if *arg* is not an *lvalue reference*.

If *arg* is an *lvalue reference*, the function returns *arg* without modifying its type.

This is a helper function to allow *perfect forwarding* of arguments taken as rvalue references to *deduced types*, preserving any potential *move semantics* involved.

The need for this function stems from the fact that all named values (such as function parameters) always evaluate as lvalues (even those declared as *rvalue references*), and this poses difficulties in preserving potential *move semantics* on template functions that forward arguments to other functions.

Both signatures return the same as:

```
static_cast<decltype(arg)&&>(arg)
```

By providing two signatures and using `remove_reference` on `T`, any instantiation is forced to explicitly specify the type of `T` (any implicitly deduced `T` would have no match).

Parameters

`arg`
An object.

Return value

If *arg* is an *lvalue reference*, the function returns *arg* with its type unchanged.
Otherwise, the function returns an *rvalue reference* (`T&&`) that refers to *arg* that can be used to pass an *rvalue*.

Example

```
// forward example
#include <utility>      // std::forward
#include <iostream>     // std::cout

// function with lvalue and rvalue reference overloads:
void overloaded (const int& x) {std::cout << "[lvalue]";}
void overloaded (int&& x) {std::cout << "[rvalue]";}

// function template taking rvalue reference to deduced type:
template <class T> void fn (T&& x) {
  overloaded (x);                   // always an lvalue
  overloaded (std::forward<T>(x));  // rvalue if argument is rvalue
}

int main () {
  int a;

  std::cout << "calling fn with lvalue: ";
  fn (a);
  std::cout << '\n';

  std::cout << "calling fn with rvalue: ";
  fn (0);
  std::cout << '\n';

  return 0;
}
```

Edit & Run

Output:

```
calling fn with lvalue: [lvalue][lvalue]
calling fn with rvalue: [lvalue][rvalue]
```

Data races

none

Exceptions

No-throw guarantee: this function never throws exceptions.

See also

move	Move as rvalue (function template)