



We have 5 open jobs ♥

Imagine yourself at Macquarie Group

What is the purpose of std::make_pair vs the constructor of std::pair?

[Ask Question](#)

What is the purpose of
`std::make_pair` ?

Why not just do
`std::pair<int, char>(0, 'a')` ?

Is there any difference between the two methods?

`c++` `stl`
`std-pair`

edited Jan 6 '17 at 10:35



Ciro Santilli 新疆改造中心 六四事件 法轮功

115k 23 447 394

asked Feb 14 '12 at 1:37

user542687

5 In C++11, you

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

See
[my
answer](#)
.
–
[PlagueHammer](#)
Mar 19
'14 at
6:54

1 In
C++17,
std::
make_
pair
is
redund
ant.
There
is an
answer
below
that
details
this. –
[Drew Dormann](#)
May 22
at
14:56

6 Answers

The
difference is
that with
std::pair
you need to
specify the
types of both
elements,
whereas
std::make_pa
ir will create
a pair with
the type of
the elements
that are
passed to it,
without you
needing to
tell it. That's

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

See this
example from
http://www.cplusplus.com/reference/std/utility/make_pair/

```
pair<int,int> one;
pair<int,int> two;

one = make_pair(1,2);
two = make_pair(3,4);
```

Aside from
the implicit
conversion
bonus of it, if
you didn't use
make_pair
you'd have to
do

```
one = pair<int,int>(1,2);
```

every time
you assigned
to one, which
would be
annoying
over time...

edited Feb 14 '12 at 1:51

answered Feb 14 '12 at 1:39



Tor Valamo

24.2k 8 62 75

Actually,
the
types
should
be
deduced
at
compile
time
without
the
need

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

'12 at
1:51

@Tor
Yeah, I
know
how to
use
both of
them, I
was
just
curious
if there
was a
reason
for
std::
make_
pair .
Appare
ntly it
is just
for
conven
ience.

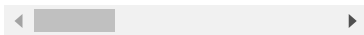
—
user5
42687
Feb 14
'12 at
1:56

@Jay
It
would
appear
so. —
[Tor Vala](#)
Feb 14
'12 at
1:58

13 I think
you
can do
one =
{10,
20}
nowad
ays but
I don't
have a
C++11
compil
er
handy
to
. . .

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

4 Also note that make_pair works with unnamed types, including structs, unions, lambdas, and other doodads. – [Mooing](#) Feb 6 '15 at 22:29



We have 5 open jobs ▼



Imagine yours

As @MSalters replied above, you can now use curly braces to do this in C++11 (just verified this with a C++11 compiler):

```
pair<int, in
```

[lited Mar 22 '14 at 21:06](#)



[Martin](#)

6,495 5 26 30

answered Feb 24 '14 at 18:28

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

There is no difference between using `make_pair` and explicitly calling the `pair` constructor with specified type arguments.

`std::make_pair` is more convenient when the types are verbose because a template method has type deduction based on its given parameters. For example,

```
std::vector<
std::vector<

// shorter
vecOfPair.put

// longer
vecOfPair.put(
emptyV));
```

answered Feb 14 '12 at 1:49

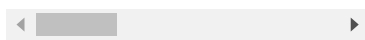


devil

1,424

12

21



It's worth noting that this is a common idiom in C++ template

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

can find more
information
and a nice
example
[here](#).

Edit As
someone
suggested in
the
comments
(since
removed) the
following is a
slightly
modified
extract from
the link in
case it
breaks.

An Object
Generator
allows
creation of
objects
without
explicitly
specifying
their types. It
is based on a
useful
property of
function
templates
which class
templates
don't have:
The type
parameters of
a function
template are
deduced
automatically
from its
actual
parameters.
std::make_pa
ir is a

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

template
depending on
the actual
parameters of
the
std::make_pair
function.

```
template <class T, class U>
std::pair<T, U>
make_pair(T t, U u)
{
    return std::pair<T, U>(t, u);
}
```

edited Apr 3 '14 at 9:37

answered Mar 15 '13 at 14:35



mkm

450 4 17

I think
your
code is
missin
g
referen
ces &

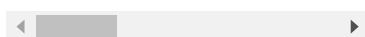
—

[Mooing](#)
Feb 6
'15 at
22:31

@duck
Actuall
y &&
since
C++11.

—

[Justme](#)
Jan 22
at
15:53



**Class
template
arguments**

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

Before
C++17 you
could not
write
something
like:

```
std::pair p(
```

since that
would infer
template
types from
the
constructor
arguments.

C++17
makes that
syntax
possible, and
therefore
make_pair
redundant.

Before
C++17,
std::make_pa
ir allowed
us to write
less verbose
code:

```
MyLongClassN  
MyLongClassN  
auto p = std
```

instead of the
more
verbose:

```
std::pair<MyL
```

which repeats
the types,
and can be
very long.

Type
inference
works in that

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

make_pair is
essentially
equivalent to:

```
template<class T1, class T2>
std::pair<T1, T2>
make_pair(const T1& a, const T2& b)
{
    return std::pair<T1, T2>(a, b);
}
```

The same
concept
applies to
insertion VS
insert_iterator .

See also:

- [Why not infer template parameter from constructor?](#)
- https://en.cppreference.com/w/cpp/string/basic/string_view

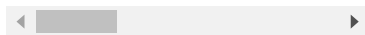
edited May 22 at 15:03

answered Jan 7 '17 at 12:26



Ciro Santilli 新疆改造中
心 六四事件 法轮功

115k 23 447 394



make_pair
creates an
extra copy
over the

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

simple
syntax.
This shows
the difference
(example by
Rampal
Chaudhary):

```
class Sample
{
    static int _objectNumber;

    int _objectNumber;
public:
    Sample()
    {
        _objectNumber++;
    }

    Sample(int _objectNumber)
    {
        _objectNumber = _objectNumber;
    }

    ~Sample()
    {
        _objectNumber--;
    }
};

int Sample::_objectNumber = 0;

int main(int argc, char* argv[])
{
    Sample s1(1);
    std::map<int, Sample> map;

    map.insert(std::make_pair(1, s1));
    //map.insert(std::pair<int, Sample>(1, s1));
    return 0;
}
```

edited Nov 28 '13 at 14:34

answered Nov 28 '13 at 13:40



EmpZoooli

41 2

3 I am
pretty
sure
that

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

if the
optimiz
ation
setting
s of the
compil
er are
high
enoug
h. –

[Björn P](#)
Feb 19
'14 at
13:00

Why
would
you
ever
want to
rely on
compil
er
optimiz
ations
for
correct
ness?

– [sjbx](#)
Dec 2
'16 at
8:46

I get
the
same
results
with
both
version
s, and
with
std::
move
just
inside
inser
t
and/or
around
what
would
be a
referen
ce to
sampl
e . It is
only

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

```

std::
map<i
nt, Sa
mple
const
&>
that I
reduce
the
numbe
r of
constru
cted
objects
, and
only
when I
delete
the
copy
constru
ctor
that I
elimina
te all
copies
(obviou
sly).
After
making
both of
those
change
s, my
result
include
s one
call to
the
default
constru
ctor
and
two
calls to
the
destruc
tor for
the
same
object.
I think I
must
be
missin
g
someth
ing.

```

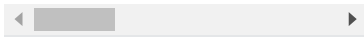
This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

'17 at
23:13

FWIW
I agree
that
optimiz
ation
and
correct
ness
should
be
comple
tely
indepe
ndent,
as this
is
exactly
the
kind of
code
you
write
as a
sanity
check
after
differe
nt
optimiz
ation
levels
produc
e
inconsi
stent
results.
In
genera
l I
would
recom
mend
empla
ce
instead
of
inser
t if
you're
just
constru
cting a
value
to
insert

This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.

instanc
es.) It's
not my
area of
experti
se, if I
can
even
say I
have
one,
but the
copy/m
ove
semant
ics
introdu
ced by
C++11
have
helped
me a
lot. –
[John P](#)
Aug 30
'17 at
23:31



This site uses cookies to deliver our services and to show you relevant ads and job listings. By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. Your use of Stack Overflow's Products and Services, including the Stack Overflow Network, is subject to these policies and terms.