Trait core::cmp::PartialOrd

Trait for values that can be compared for a sort-order.

The comparison must satisfy, for all `a`, `b` and `c`:

- antisymmetry: if `a < b` then `!(a > b)`, as well as `a > b` implying `!(a < b)`; and
- transitivity: `a < b` and `b < c` implies `a < c`. The same must hold for both `==` and `>`.

Note that these requirements mean that the trait itself must be implemented symmetrically and transitively: if `T: PartialOrd<U>` and `U: PartialOrd<V>` then `U: PartialOrd<T>` and `T: PartialOrd<V>`.

Derivable

This trait can be used with `#[derive]`. When `derive`d on structs, it will produce a lexicographic ordering based on the top-to-bottom declaration order of the struct's members. When `derive`d on enums, variants are ordered by their top-to-bottom declaration order.

How can I implement `PartialOrd`?

`PartialOrd` only requires implementation of the `partial_cmp` method, with the others generated from default implementations.

However it remains possible to implement the others separately for types which do not have a total order. For example, for floating point numbers, `NaN < 0 == false` and `NaN >= 0 == false` (cf. IEEE 754-2008 section 5.11).

`PartialOrd` requires your type to be `PartialEq`.

Implementations of `PartialEq`, `PartialOrd`, and `Ord` *must* agree with each other. It's easy to accidentally make them disagree by deriving some of the traits and manually implementing others.

If your type is `Ord`, you can implement `partial_cmp()` by using `cmp()`:

Run

```
use std::cmp::Ordering;

#[derive(Eq)]
struct Person {
    id: u32,
    name: String,
    height: u32,
}

impl PartialOrd for Person {
    fn partial_cmp(&self, other: &Person) -> Option<Ordering> {
        Some(self.cmp(other))
    }
}

impl Ord for Person {
    fn cmp(&self, other: &Person) -> Ordering {
        self.height.cmp(&other.height)
    }
}

impl PartialEq for Person {
    fn eq(&self, other: &Person) -> bool {
        self.height == other.height
    }
}
```

You may also find it useful to use `partial_cmp()` on your type's fields. Here is an example of `Person` types who have a floating-point `height` field that is the only field to be used for sorting:

Run

```
use std::cmp::Ordering;

struct Person {
    id: u32,
```


```
    name: String,
    height: f64,
}

impl PartialOrd for Person {
    fn partial_cmp(&self, other: &Person) -> Option<Ordering> {
        self.height.partial_cmp(&other.height)
    }
}

impl PartialEq for Person {
    fn eq(&self, other: &Person) -> bool {
        self.height == other.height
    }
}
```

Examples

```
let x : u32 = 0;
let y : u32 = 1;

assert_eq!(x < y, true);
assert_eq!(x.lt(&y), true);
```

Run

Required Methods

fn partial_cmp(&self, other: &Rhs) -> Option<Ordering>

This method returns an ordering between `self` and `other` values if one exists.

Examples

```
use std::cmp::Ordering;

let result = 1.0.partial_cmp(&2.0);
assert_eq!(result, Some(Ordering::Less));

let result = 1.0.partial_cmp(&1.0);
assert_eq!(result, Some(Ordering::Equal));

let result = 2.0.partial_cmp(&1.0);
assert_eq!(result, Some(Ordering::Greater));
```

Run

When comparison is impossible:

```
let result = std::f64::NAN.partial_cmp(&1.0);
assert_eq!(result, None);
```

Run

Provided Methods

fn lt(&self, other: &Rhs) -> bool

This method tests less than (for `self` and `other`) and is used by the `<` operator.

Examples

```
let result = 1.0 < 2.0;
assert_eq!(result, true);

let result = 2.0 < 1.0;
assert_eq!(result, false);
```

Run

fn le(&self, other: &Rhs) -> bool

This method tests less than or equal to (for `self` and `other`) and is used by the `<=` operator.

Trait PartialOrd

Required Methods

partial_cmp

Provided Methods

lt

le

gt

ge

Implementors

core::cmp

Structs

Reverse

Enums

Ordering

Traits

Eq

Ord

PartialEq

PartialOrd

Functions

max

min

Examples

```
let result = 1.0 <= 2.0;
assert_eq!(result, true);

let result = 2.0 <= 2.0;
assert_eq!(result, true);
```

Run

`fn gt(&self, other: &Rhs) -> bool`

This method tests greater than (for `self` and `other`) and is used by the `>` operator.

Examples

```
let result = 1.0 > 2.0;
assert_eq!(result, false);

let result = 2.0 > 2.0;
assert_eq!(result, false);
```

Run

`fn ge(&self, other: &Rhs) -> bool`

This method tests greater than or equal to (for `self` and `other`) and is used by the `>=` operator.

Examples

```
let result = 2.0 >= 1.0;
assert_eq!(result, true);

let result = 2.0 >= 2.0;
assert_eq!(result, true);
```

Run

Implementors

`impl PartialOrd for NonZeroU8` [src]

`impl PartialOrd for NonZeroU16` [src]

`impl PartialOrd for NonZeroU32` [src]

`impl PartialOrd for NonZeroU64` [src]

`impl PartialOrd for NonZeroU128` [src]

`impl PartialOrd for NonZeroUsize` [src]

`impl PartialOrd for NonZeroI8` [src]

`impl PartialOrd for NonZeroI16` [src]

`impl PartialOrd for NonZeroI32` [src]

`impl PartialOrd for NonZeroI64` [src]

`impl PartialOrd for NonZeroI128` [src]

`impl PartialOrd for NonZeroIsize` [src]

`impl<T: PartialOrd> PartialOrd for Wrapping<T>` [src]

`impl<T: PartialOrd> PartialOrd for ManuallyDrop<T>` [src]

`impl<T: PartialOrd + Zeroable> PartialOrd for NonZero<T>` [src]

`impl<Ret> PartialOrd for fn() -> Ret` [src]

`impl<Ret> PartialOrd for extern "C" fn() -> Ret` [src]

`impl<Ret> PartialOrd for unsafe fn() -> Ret` [src]

Trait PartialOrd

Required Methods

partial_cmp

Provided Methods

lt
le
gt
ge

Implementors

core::cmp

Structs

Reverse

Enums

Ordering

Traits

Eq
Ord
PartialEq
PartialOrd

Functions

max
min

impl<Ret> PartialOrd for unsafe extern "C" fn() -> Ret [src]

impl<Ret, A> PartialOrd for fn(_: A) -> Ret [src]

impl<Ret, A> PartialOrd for extern "C" fn(_: A) -> Ret [src]

impl<Ret, A> PartialOrd for extern "C" fn(_: A, ...) -> Ret [src]

impl<Ret, A> PartialOrd for unsafe fn(_: A) -> Ret [src]

impl<Ret, A> PartialOrd for unsafe extern "C" fn(_: A) -> Ret [src]

impl<Ret, A> PartialOrd for unsafe extern "C" fn(_: A, ...) -> Ret [src]

impl<Ret, A, B> PartialOrd for fn(_: A, _: B) -> Ret [src]

impl<Ret, A, B> PartialOrd for extern "C" fn(_: A, _: B) -> Ret [src]

impl<Ret, A, B> PartialOrd for extern "C" fn(_: A, _: B, ...) -> Ret [src]

impl<Ret, A, B> PartialOrd for unsafe fn(_: A, _: B) -> Ret [src]

impl<Ret, A, B> PartialOrd for unsafe extern "C" fn(_: A, _: B) -> Ret [src]

impl<Ret, A, B> PartialOrd for unsafe extern "C" fn(_: A, _: B, ...) -> Ret [src]

impl<Ret, A, B, C> PartialOrd for fn(_: A, _: B, _: C) -> Ret [src]

impl<Ret, A, B, C> PartialOrd for extern "C" fn(_: A, _: B, _: C) -> Ret [src]

impl<Ret, A, B, C> PartialOrd for extern "C" fn(_: A, _: B, _: C, ...) -> Ret [src]

impl<Ret, A, B, C> PartialOrd for unsafe fn(_: A, _: B, _: C) -> Ret [src]

impl<Ret, A, B, C> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C) -> Ret [src]

impl<Ret, A, B, C> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, ...) -> Ret [src]

impl<Ret, A, B, C, D> PartialOrd for fn(_: A, _: B, _: C, _: D) -> Ret [src]

impl<Ret, A, B, C, D> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D) -> Ret [src]

impl<Ret, A, B, C, D> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, ...) -> Ret [src]

impl<Ret, A, B, C, D> PartialOrd for unsafe fn(_: A, _: B, _: C, _: D) -> Ret [src]

impl<Ret, A, B, C, D> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D) -> Ret [src]

impl<Ret, A, B, C, D> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, ...) -> Ret [src]

impl<Ret, A, B, C, D, E> PartialOrd for fn(_: A, _: B, _: C, _: D, _: E) -> Ret [src]

impl<Ret, A, B, C, D, E> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E) -> Ret [src]

impl<Ret, A, B, C, D, E> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, ...) -> Ret [src]

impl<Ret, A, B, C, D, E> PartialOrd for unsafe fn(_: A, _: B, _: C, _: D, _: E) -> Ret [src]

impl<Ret, A, B, C, D, E> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E) -> Ret [src]

impl<Ret, A, B, C, D, E> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F> PartialOrd for fn(_: A, _: B, _: C, _: D, _: E, _: F) -> Ret [src]

impl<Ret, A, B, C, D, E, F> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F) -> Ret

impl<Ret, A, B, C, D, E, F> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F> PartialOrd for unsafe fn(_: A, _: B, _: C, _: D, _: E, _: F) -> Ret [src]

impl<Ret, A, B, C, D, E, F> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F) -> Ret [src]

impl<Ret, A, B, C, D, E, F> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G> PartialOrd for fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G> PartialOrd for unsafe fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H> PartialOrd for fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H> PartialOrd for unsafe fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I> PartialOrd for fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I> PartialOrd for unsafe fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J> PartialOrd for fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J) -> Ret [src]

Trait PartialOrd

Required Methods

partial_cmp

Provided Methods

lt
le
gt
ge

Implementors

core::cmp

Structs

Reverse

Enums

Ordering

Traits

Eq
Ord
PartialEq
PartialOrd

Functions

max
min

impl<Ret, A, B, C, D, E, F, G, H, I, J> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J> PartialOrd for unsafe fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K> PartialOrd for fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K> PartialOrd for unsafe fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K, L> PartialOrd for fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K, _: L) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K, L> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K, _: L) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K, L> PartialOrd for extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K, _: L, ...) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K, L> PartialOrd for unsafe fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K, _: L) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K, L> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K, _: L) -> Ret [src]

impl<Ret, A, B, C, D, E, F, G, H, I, J, K, L> PartialOrd for unsafe extern "C" fn(_: A, _: B, _: C, _: D, _: E, _: F, _: G, _: H, _: I, _: J, _: K, _: L, ...) -> Ret [src]

impl<T: ?Sized> PartialOrd for *const T [src]

impl<T: ?Sized> PartialOrd for *mut T [src]

impl<T: ?Sized> PartialOrd for NonNull<T> [src]

impl<T: ?Sized> PartialOrd for PhantomData<T> [src]

impl<Y: PartialOrd, R: PartialOrd> PartialOrd for GeneratorState<Y, R> [src]

impl PartialOrd for () [src]

impl PartialOrd for bool [src]

impl PartialOrd for f32 [src]

impl PartialOrd for f64 [src]

impl PartialOrd for char [src]

impl PartialOrd for usize [src]

impl PartialOrd for u8 [src]

impl PartialOrd for u16 [src]

Trait PartialOrd

Required Methods

partial_cmp

Provided Methods

lt
le
gt
ge

Implementors

core::cmp

Structs

Reverse

Enums

Ordering

Traits

Eq
Ord
PartialEq
PartialOrd

Functions

max
min

impl PartialOrd for u32 [src]

impl PartialOrd for u64 [src]

impl PartialOrd for u128 [src]

impl PartialOrd for isize [src]

impl PartialOrd for i8 [src]

impl PartialOrd for i16 [src]

impl PartialOrd for i32 [src]

impl PartialOrd for i64 [src]

impl PartialOrd for i128 [src]

impl PartialOrd for ! [src]

impl<'a, 'b, A: ?Sized, B: ?Sized> PartialOrd<&'b B> for &'a A [src]
where
    A: PartialOrd<B>,

impl<'a, 'b, A: ?Sized, B: ?Sized> PartialOrd<&'b mut B> for &'a mut A [src]
where
    A: PartialOrd<B>,

impl<T: PartialOrd> PartialOrd for Reverse<T> [src]

impl PartialOrd for Ordering [src]

impl PartialOrd for TypeId [src]

impl<T: PartialOrd> PartialOrd for [T; 0] [src]

impl<T: PartialOrd> PartialOrd for [T; 1] [src]

impl<T: PartialOrd> PartialOrd for [T; 2] [src]

impl<T: PartialOrd> PartialOrd for [T; 3] [src]

impl<T: PartialOrd> PartialOrd for [T; 4] [src]

impl<T: PartialOrd> PartialOrd for [T; 5] [src]

impl<T: PartialOrd> PartialOrd for [T; 6] [src]

impl<T: PartialOrd> PartialOrd for [T; 7] [src]

impl<T: PartialOrd> PartialOrd for [T; 8] [src]

impl<T: PartialOrd> PartialOrd for [T; 9] [src]

impl<T: PartialOrd> PartialOrd for [T; 10] [src]

impl<T: PartialOrd> PartialOrd for [T; 11] [src]

impl<T: PartialOrd> PartialOrd for [T; 12] [src]

impl<T: PartialOrd> PartialOrd for [T; 13] [src]

impl<T: PartialOrd> PartialOrd for [T; 14] [src]

impl<T: PartialOrd> PartialOrd for [T; 15] [src]

impl<T: PartialOrd> PartialOrd for [T; 16] [src]

impl<T: PartialOrd> PartialOrd for [T; 17] [src]

impl<T: PartialOrd> PartialOrd for [T; 18] [src]

impl<T: PartialOrd> PartialOrd for [T; 19] [src]

Trait PartialOrd

Required Methods

partial_cmp

Provided Methods

lt
le
gt
ge

Implementors

core::cmp

Structs

Reverse

Enums

Ordering

Traits

Eq
Ord
PartialEq
PartialOrd

Functions

max
min

impl<T: PartialOrd> PartialOrd for [T; 20]

impl<T: PartialOrd> PartialOrd for [T; 21] [src]

impl<T: PartialOrd> PartialOrd for [T; 22] [src]

impl<T: PartialOrd> PartialOrd for [T; 23] [src]

impl<T: PartialOrd> PartialOrd for [T; 24] [src]

impl<T: PartialOrd> PartialOrd for [T; 25] [src]

impl<T: PartialOrd> PartialOrd for [T; 26] [src]

impl<T: PartialOrd> PartialOrd for [T; 27] [src]

impl<T: PartialOrd> PartialOrd for [T; 28] [src]

impl<T: PartialOrd> PartialOrd for [T; 29] [src]

impl<T: PartialOrd> PartialOrd for [T; 30] [src]

impl<T: PartialOrd> PartialOrd for [T; 31] [src]

impl<T: PartialOrd> PartialOrd for [T; 32] [src]

impl<T: PartialOrd + Copy> PartialOrd for Cell<T> [src]

impl<T: ?Sized + PartialOrd> PartialOrd for RefCell<T> [src]

impl<T: PartialOrd> PartialOrd for Option<T> [src]

impl PartialOrd for NoneError [src]

impl<T: PartialOrd, E: PartialOrd> PartialOrd for Result<T, E> [src]

impl<T: PartialOrd> PartialOrd for [T] [src]

impl PartialOrd for str [src]

impl PartialOrd for Error [src]

impl PartialOrd for Duration [src]

impl<A> PartialOrd for (A,) [src]
where
    A: PartialOrd + PartialEq + ?Sized,

impl<A: PartialOrd + PartialEq, B> PartialOrd for (A, B) [src]
where
    B: PartialOrd + PartialEq + ?Sized,

impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C> PartialOrd for (A, B, C) [src]
where
    C: PartialOrd + PartialEq + ?Sized,

impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C: PartialOrd + PartialEq, D> PartialOrd for (A, B, C, D) [src]
where
    D: PartialOrd + PartialEq + ?Sized,

impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C: PartialOrd + PartialEq, D: PartialOrd + PartialEq, E> PartialOrd for (A, B, C, D, E) [src]
where
    E: PartialOrd + PartialEq + ?Sized,

impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C: PartialOrd + PartialEq, D: PartialOrd + PartialEq, E: PartialOrd + PartialEq, F> PartialOrd for (A, B, C, D, E, F) [src]
where
    F: PartialOrd + PartialEq + ?Sized,

impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C: PartialOrd + PartialEq, D: PartialOrd + PartialEq, E: PartialOrd + PartialEq, F: PartialOrd + [src]

Trait PartialOrd

Required Methods

partial_cmp

Provided Methods

lt

le

gt

ge

Implementors

core::cmp

Structs

Reverse

Enums

Ordering

Traits

Eq

Ord

PartialEq

PartialOrd

Functions

max

min

```
PartialEq, G> PartialOrd for (A, B, C, D, E, F, G)
where
    G: PartialOrd + PartialEq + ?Sized,
```

```
impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C: PartialOrd +
PartialEq, D: PartialOrd + PartialEq, E: PartialOrd + PartialEq, F: PartialOrd +
PartialEq, G: PartialOrd + PartialEq, H> PartialOrd for (A, B, C, D, E, F, G, H)
where
    H: PartialOrd + PartialEq + ?Sized,
```
[src]

```
impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C: PartialOrd +
PartialEq, D: PartialOrd + PartialEq, E: PartialOrd + PartialEq, F: PartialOrd +
PartialEq, G: PartialOrd + PartialEq, H: PartialOrd + PartialEq, I> PartialOrd
for (A, B, C, D, E, F, G, H, I)
where
    I: PartialOrd + PartialEq + ?Sized,
```
[src]

```
impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C: PartialOrd +
PartialEq, D: PartialOrd + PartialEq, E: PartialOrd + PartialEq, F: PartialOrd +
PartialEq, G: PartialOrd + PartialEq, H: PartialOrd + PartialEq, I: PartialOrd +
PartialEq, J> PartialOrd for (A, B, C, D, E, F, G, H, I, J)
where
    J: PartialOrd + PartialEq + ?Sized,
```
[src]

```
impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C: PartialOrd +
PartialEq, D: PartialOrd + PartialEq, E: PartialOrd + PartialEq, F: PartialOrd +
PartialEq, G: PartialOrd + PartialEq, H: PartialOrd + PartialEq, I: PartialOrd +
PartialEq, J: PartialOrd + PartialEq, K> PartialOrd for (A, B, C, D, E, F, G, H,
I, J, K)
where
    K: PartialOrd + PartialEq + ?Sized,
```
[src]

```
impl<A: PartialOrd + PartialEq, B: PartialOrd + PartialEq, C: PartialOrd +
PartialEq, D: PartialOrd + PartialEq, E: PartialOrd + PartialEq, F: PartialOrd +
PartialEq, G: PartialOrd + PartialEq, H: PartialOrd + PartialEq, I: PartialOrd +
PartialEq, J: PartialOrd + PartialEq, K: PartialOrd + PartialEq, L> PartialOrd
for (A, B, C, D, E, F, G, H, I, J, K, L)
where
    L: PartialOrd + PartialEq + ?Sized,
```
[src]

```
impl PartialOrd for i8x2
```
[src]

```
impl PartialOrd for u8x2
```
[src]

```
impl PartialOrd for b8x2
```
[src]

```
impl PartialOrd for i16x2
```
[src]

```
impl PartialOrd for u16x2
```
[src]

```
impl PartialOrd for i8x4
```
[src]

```
impl PartialOrd for u8x4
```
[src]

```
impl PartialOrd for b8x4
```
[src]

```
impl PartialOrd for i8x8
```
[src]

```
impl PartialOrd for u8x8
```
[src]

```
impl PartialOrd for b8x8
```
[src]

```
impl PartialOrd for i16x4
```
[src]

```
impl PartialOrd for u16x4
```
[src]

```
impl PartialOrd for i32x2
```
[src]

```
impl PartialOrd for u32x2
```
[src]

```
impl PartialOrd for f32x2
```
[src]

```
impl PartialOrd for i8x16
```
[src]

Trait PartialOrd

Required Methods

partial_cmp

Provided Methods

lt
le
gt
ge

Implementors

core::cmp

Structs

Reverse

Enums

Ordering

Traits

Eq
Ord
PartialEq
PartialOrd

Functions

max
min

impl PartialOrd for u8x16

impl PartialOrd for b8x16 [src]

impl PartialOrd for i16x8 [src]

impl PartialOrd for u16x8 [src]

impl PartialOrd for i32x4 [src]

impl PartialOrd for u32x4 [src]

impl PartialOrd for f32x4 [src]

impl PartialOrd for i64x2 [src]

impl PartialOrd for u64x2 [src]

impl PartialOrd for f64x2 [src]

impl PartialOrd for i8x32 [src]

impl PartialOrd for u8x32 [src]

impl PartialOrd for b8x32 [src]

impl PartialOrd for i16x16 [src]

impl PartialOrd for u16x16 [src]

impl PartialOrd for i32x8 [src]

impl PartialOrd for u32x8 [src]

impl PartialOrd for f32x8 [src]

impl PartialOrd for i64x4 [src]

impl PartialOrd for u64x4 [src]

impl PartialOrd for f64x4 [src]

impl PartialOrd for i8x64 [src]

impl PartialOrd for u8x64 [src]

impl PartialOrd for b8x64 [src]

impl PartialOrd for i16x32 [src]

impl PartialOrd for u16x32 [src]

impl PartialOrd for i32x16 [src]

impl PartialOrd for u32x16 [src]

impl PartialOrd for f32x16 [src]

impl PartialOrd for i64x8 [src]

impl PartialOrd for u64x8 [src]

impl PartialOrd for f64x8 [src]

impl PartialOrd for CpuidResult [src]

impl<T: ?Sized + PartialOrd> PartialOrd for Box<T>

impl<T: ?Sized + PartialOrd> PartialOrd for Arc<T>

impl<T: ?Sized + PartialOrd> PartialOrd for Rc<T>

impl<K: PartialOrd, V: PartialOrd> PartialOrd for BTreeMap<K, V>

Trait PartialOrd

Required Methods

partial_cmp

Provided Methods

lt

le

gt

ge

Implementors

core::cmp

Structs

Reverse

Enums

Ordering

Traits

Eq

Ord

PartialEq

PartialOrd

Functions

max

min

```
impl<T: PartialOrd> PartialOrd for BTreeSet<T>

impl<'a, B: ?Sized> PartialOrd for Cow<'a, B>
where
    B: PartialOrd + ToOwned,

impl<T: PartialOrd> PartialOrd for LinkedList<T>

impl PartialOrd for String

impl<T: PartialOrd> PartialOrd for Vec<T>

impl<A: PartialOrd> PartialOrd for VecDeque<A>

impl PartialOrd for UnicodeVersion
```