Advanced Programming [146125] - PATRIGNANI

Domanda 1

Will the following code compile?

```
    let x = String::from(“hello”);
    let y = x;
    println!("{}, world!", y);
    println!("{}, world!", x);
```

Domanda 1 Risposta

yes

no, x does not implement Display

no, x is re-borrowed

no, x is moved

Domanda 2

What does this evaluate to?

```
{
    let mut s1 = String::from(“Hello!“);
    {
        let mut s2 = &s1;
        s2.push_str("World!");
        println!("{}", s2)
    }
}
```

Domanda 2 Risposta

print: "Hello! World!"

print: "Hello!World!"

error

print: "Hello!"

Domanda 3

What does this evaluate to?

```
{
        let mut a = 50;
        let b = &mut a;
        a = 20;
        println!(“a: {}, a:{}“,a,a);
}
```

Domanda 3 Risposta

error

print: "a: 50, a: 50"

print: "a: 20, a: 20"

print: "a: 20, a: 50"

Domanda 4

What is printed?

```
fn foo(s: &mut String) -> usize {
    s.push_str("Bob");
    s.len()
}
fn main() {
    let mut s1 = String::from(”Alice”);
    println!("{}",foo(&mut s1));
}
```

Domanda 4 Risposta

8

0

error

5

Domanda 5

Will the following code compile?

```
let mut v: Vec<f32> = vec![1.0, 2.5, 3.7];
for i in v.iter_mut() {
    // `powf` is the power function for floats
    *i = i.powf(2.0);
}
```

Domanda 5 Risposta

No, `i` is declared as immutable

No, `&f32` doesn't implement the method `powf`

No, cannot immutably borrow `v` (at line 4) while mutably borrowed (at line 2)

Yes

Domanda 6

What does this code print?

```
pub fn main(){
  let mut array = vec![10, 20, 30];
   function(&array);
  println!(”{}”, array[0]);
}
fn function(array: &mut Vec<u32>) {
    array[0] = 100;
}
```

Domanda 6 Risposta

error

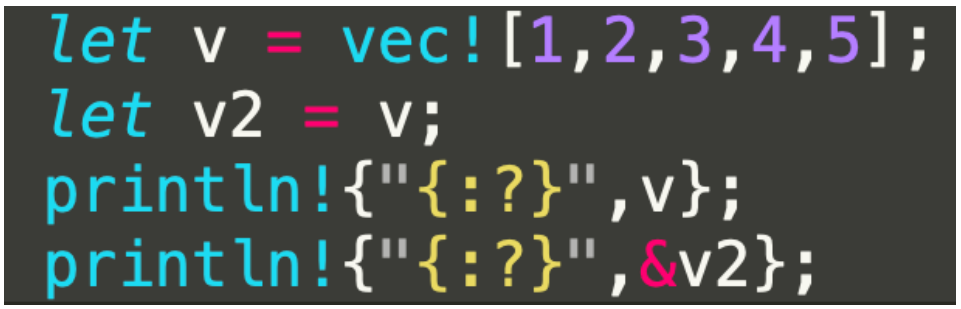
10, 20, 30

10

100

Domanda 7

Will the following code compile?



Domanda 7 Risposta

Yes

No, Vec dose not implement Display

No, v2 can't be borrowed at line 4

No, v is moved at line 2, and can't be used at line 3

Domanda 8

Will the following code compile?

```rust
fn foo(s: String)->String{
    s
}
pub fn main(){
    let mut s = String::from("hello"); // this is line 4
    let s = foo(s);
    println!{"{:?}",s};
}
```

Domanda 8 Risposta

No, s is moved inside the function , and can't be used at line 6

Yes

No, String dose not implement Debug

No, s can't be borrowed at line 5

Domanda 9

write enum A with variants A2 A1, the former takes two chars, the latter three i32's

write enum B with variants B1 B2, the former takes 2 i32's, the latter a String

write a function `bfroma` that takes an A and then returns a B where B1 has i32's that are the i32 casting of their A2 counterparts, and B2 has the sting that is the concatenation of the three floats in A1, separated by a -

For example:

Test	Result

`pub fn main(){`
`  let a1 = A::A1(1,2,3);`
`  let a2 = A::A2('a','b');`
`  println!("B2: {:?}, B1:{:?}", bfroma(a1), bfroma(a2));`
`}`
	`B2: B2("1-2-3"), B1:B1(97, 98)`

Feedback

Test	Expected	Got

`pub fn main(){`
`  let a1 = A::A1(1,2,3);`
`  let a2 = A::A2('a','b');`
`  println!("B2: {:?}, B1:{:?}", bfroma(a1), bfroma(a2));`
`}`
	`B2: B2("1-2-3"), B1:B1(97, 98)`	`B2: B2("1-2-3"), B1:B1(97, 98)`

`pub fn main(){`
`  let a1 = A::A1(1,6,30);`
`  let a2 = A::A2('t','z');`
`  println!("B2: {:?}, B1:{:?}",`
	`B2: B2("1-6-30"), B1:B1(116, 122)`	`B2: B2("1-6-30"), B1:B1(116, 122)`

Test	Expected	Got

```
bfroma(a1), bfroma(a2));
}
```

Domanda 10

Write an enum `E` that contains 2 variants: A and B, the first contains a String and the latter contains a bool

Write an enum `F` that contains 2 variants: F1 and F2, the former contains a String and the latter contains a i32

For E write a method `count_vowels` that if the variant is A, count the number of vowels in the String, otherwise return zero

for F write a method `calculation` that if the variant is F1, return a usize with the length of the String, otherwise return a usize with the casted value of the i32

moreover, for F write a function `new` that return a F1 with the String "hello"

For example:

Test	Result

```
fn main() {
    let e1 = E::A("hello".to_string());
    let e2 = E::B(true);
    println!("{:?} {:?}", e1, e1.count_vowels());
    println!("{:?} {:?}", e2, e2.count_vowels());
}
```

```
A("hello") 2
B(true) 0
```

```
fn main() {
    let f1 = F::new();
    let f2 = F::F2(10);
    let f3 = F::F2(20);
    println!("{:?} {:?}", f1, f1.calculation());
    println!("{:?} {:?}", f2, f2.calculation());
    println!("{:?} {:?}", f3, f3.calculation());
}
```

```
F1("hello") 5
F2(10) 10
F2(20) 20
```

Feedback

Test	Expected	Got

```
fn main() {
    let e1 = E::A("hello".to_string());
    let e2 = E::B(true);
    println!("{:?} {:?}", e1, e1.count_vowels());
    println!("{:?} {:?}", e2, e2.count_vowels());
}
```

```
A("hello") 2
B(true) 0
```

```
A("hello") 2
B(true) 0
```

```
fn main() {
    let f1 = F::new();
    let f2 = F::F2(10);
    let f3 = F::F2(20);
    println!("{:?} {:?}", f1, f1.calculation());
    println!("{:?} {:?}", f2, f2.calculation());
    println!("{:?} {:?}", f3, f3.calculation());
}
```

```
F1("hello") 5
F2(10) 10
F2(20) 20
```

```
F1("hello") 5
F2(10) 10
F2(20) 20
```

Domanda 11

write a function print_n that take as input an Option

if the option is Some(x) the function prints "x followed by a new line" for x amount of times.

if the option is None the function prints "Error"

For example:

Test	Result

`fn main() {`
`    print_n(Some(3));`
`}`
	`3`
	`3`
	`3`

`fn main() {`
`    print_n(None)`
`}`
	`Error`

Feedback

Test	Expected	Got

`fn main() {`
`    print_n(Some(3));`
`}`
	`3`
	`3`
	`3`
		`3`
		`3`
		`3`

`fn main() {`
`    print_n(None)`
`}`
	`Error`	`Error`

Domanda 12

write a `Balance` struct with a field `amt : i32` and a field `active:bool`

Add a `maybericher` method that takes another Balance b and returns true if this one is richer, false if b is richer, unless either Balance is not active

For example:

Test	Result

`pub fn main(){`
`    let b = Balance{amt:100,active:true};`
`    let b2 = Balance{amt:200,active:true};`
`    println!("maybericher {:?}", b.maybericher(b2));`
`}`
	`maybericher Some(false)`

Feedback

Test	Expected	Got

`pub fn main(){`
`    let b = Balance{amt:100,active:true};`
`    let b2 = Balance{amt:200,active:true};`
	`maybericher Some(false)`	`maybericher Some(false)`

Test	Expected	Got

```
    println!("maybericher {:?}", b.maybericher(b2));
}


pub fn main(){
    let b = Balance{amt:100,active:true};
    let b2 = Balance{amt:0,active:true};            maybericher Some(true)  maybericher Some(true)
    println!("maybericher {:?}", b.maybericher(b2));
}


pub fn main(){
    let b = Balance{amt:100,active:false};
    let b2 = Balance{amt:200,active:true};          maybericher None        maybericher None
    println!("maybericher {:?}", b.maybericher(b2));
}


pub fn main(){
    let b = Balance{amt:100,active:true};
    let b2 = Balance{amt:200,active:false};         maybericher None        maybericher None
    println!("maybericher {:?}", b.maybericher(b2));
}
```

Domanda 13

Write a Struct G with fields x, y of type i32 and a method named `new` that takes two i32 values and returns a G.

Also implement a method named `square` that returns a Result with x if x is the square of y or a void error otherwise.

For example:

Test	Result

```
fn main() {
    let g = G::new(4, 2);
    let result = g.square();  Ok(4)
    println!("{:?}", result);
}


fn main() {
    let g = G::new(4, 3);
    let result = g.square();  Err(())
    println!("{:?}", result);
}


fn main() {
    let g = G::new(121, 11);
    let result = g.square();  Ok(121)
    println!("{:?}", result);
}
```

Feedback

Test	Expected	Got

Test	Expected	Got

```
fn main() {
    let g = G::new(4, 2);
    let result = g.square(); Ok(4)   Ok(4)
    println!("{:?}", result);
}
```

```
fn main() {
    let g = G::new(4, 3);
    let result = g.square(); Err(()) Err(())
    println!("{:?}", result);
}
```

```
fn main() {
    let g = G::new(121, 11);
    let result = g.square(); Ok(121) Ok(121)
    println!("{:?}", result);
}
```

Domanda 14

write a struct X with 2 fields s : string and i : i32

write a struct Y with 2 fields b : bool and c : string

give each struct a constructor function `new`. The default values are "xxx", 10 for X and true, "op" for Y.

give each struct a method `getstr` for replacing the string with "", moving the string out of the struct and returning said string

// use std::mem::replace

write a function `swapstr` that takes a X and a Y and then moves s into c and c into s

make both displayable with :? the formatter as well as with a {} argument in println. With a {} argument, see the example for the result

For example:

Test	Result

```
pub fn main(){
    let mut x = X::new();
    let mut y = Y::new();
    println!("X {:?} - Y {:?}", x, y);
    let (mut x, mut y) = swapstr(x,y);
    println!("X {} - Y {}", x, y);
    let z1 = x.getstr();
    let z2 = y.getstr();
    println!("{},{},{},{}",z1,z2,x.s,y.c);
}
```

```
X X { s: "xxx", i: 10 } - Y Y { b: true, c: "op" }
X S op, I 10 - Y B true, C xxx
op,xxx,,
```

Feedback

Test	Expected	Got

```
pub fn main(){
    let mut x = X::new();
    let mut y = Y::new();
```

```
X X { s: "xxx", i: 10 } - Y Y { b: true, c: "op" }
X S op, I 10 - Y B true, C xxx
```

```
X X { s: "xxx", i: 10 } - Y Y { b: true, c: "op" }
X S op, I 10 - Y B true, C xxx
```

Test	Expected	Got

`    println!("X {:?} - Y {:?}", x, y);`
`    let (mut x, mut y) = swapstr(x,y);`
`    println!("X {} - Y {}", x, y);`
`    let z1 = x.getstr();`
`    let z2 = y.getstr();`
`    println!("{},{}, {},{}",z1,z2,x.s,y.c);`
`}`
	op,xxx,,	op,xxx,,

`pub fn main(){`
`    let mut x = X{ s: "xxx".to_string(), i: 0 };`
`    let mut y = Y{ b: true, c : "zzz".to_string()};`
`    println!("X {:?} - Y {:?}", x, y);`
`    let (mut x, mut y) = swapstr(x,y);`
`    println!("X {} - Y {}", x, y);`
`    let z1 = x.getstr();`
`    let z2 = y.getstr();`
`    println!("{},{}, {},{}",z1,z2,x.s,y.c);`
`}`
	X X { s: "xxx", i: 0 } - Y Y { b: true, c: "zzz" }
	X S zzz, I 0 - Y B true, C xxx
	zzz,xxx,,
		X X { s: "xxx", i: 0 } - Y Y { b: true, c: "zzz" }
		X S zzz, I 0 - Y B true, C xxx
		zzz,xxx,,

`pub fn main(){`
`    let x = X{ s: "zzz".to_string(), i: 0 };`
`    let y = Y{ b: true, c : "yyy".to_string()};`
`    println!("X {:?} - Y {:?}", x, y);`
`    let (x,y) = swapstr(x,y);`
`    println!("X {} - Y {}", x, y);`
`}`
	X X { s: "zzz", i: 0 } - Y Y { b: true, c: "yyy" }
	X S yyy, I 0 - Y B true, C zzz
		X X { s: "zzz", i: 0 } - Y Y { b: true, c: "yyy" }
		X S yyy, I 0 - Y B true, C zzz

Domanda 15

Write the struct `L` with the fields s: String and n: i32

Write the struct `M` with the fields s: String and n: f64

give the method `new` for each struct with no parameters that returns a struct with s: "hello" and n: 0

give the method `new_with_params` for each struct that takes a String and a i32/f64 and returns a struct with the given parameters

write the fn `swap_string` that swap the string field of a L and a M

// use std::mem::replace

make the structs printable using println! and the :? formatter

For example:

Test	Result

Test	Result

`fn main() {`
`  let l = L::new();`
`  let m = M::new();`
`  println!("{:?} {:?}",l,m);`
`}`
	`L { s: "hello", n: 0 } M { s: "hello", n: 0.0 }`

`fn main() {`
`  let l = L::new_with_params("world".to_string(), 10);`
`  let m = M::new_with_params("world".to_string(), 10.0);`
`  println!("{:?} {:?}",l,m);`
`}`
	`L { s: "world", n: 10 } M { s: "world", n: 10.0 }`

`fn main() {`
`  let mut l = L::new_with_params("world".to_string(), 10);`
`  let mut m = M::new_with_params("hello".to_string(), 10.0);`
`  swap_string(&mut l, &mut m);`
`  println!("{:?} {:?}",l,m);`
`}`
	`L { s: "hello", n: 10 } M { s: "world", n: 10.0 }`

Feedback

Test	Expected	Got

`fn main() {`
`  let l = L::new();`
`  let m = M::new();`
`  println!("{:?} {:?}",l,m);`
`}`
	`L { s: "hello", n: 0 } M { s: "hello", n: 0.0 }`	`L { s: "hello", n: 0 } M { s: "hello", n: 0.0 }`

`fn main() {`
`  let l = L::new_with_params("world".to_string(), 10);`
`  let m = M::new_with_params("world".to_string(), 10.0);`
`  println!("{:?} {:?}",l,m);`
`}`
	`L { s: "world", n: 10 } M { s: "world", n: 10.0 }`	`L { s: "world", n: 10 } M { s: "world", n: 10.0 }`

`fn main() {`
`  let mut l = L::new_with_params("world".to_string(), 10);`
`  let mut m = M::new_with_params("hello".to_string(), 10.0);`
`  swap_string(&mut l, &mut m);`
`  println!("{:?} {:?}",l,m);`
`}`
	`L { s: "hello", n: 10 } M { s: "world", n: 10.0 }`	`L { s: "hello", n: 10 } M { s: "world", n: 10.0 }`

Domanda 16

write a function `neighbour` that takes a vector of String and an index i and returns a Result with the concatenation of the element at index i and the element at index i+1 or a void error if i is the last index.

For example:

Test	Result

`fn main() {`
`    let v = vec!["hello".to_string(), "world".to_string(), "how".to_string(),`
`"are".to_string(), "you".to_string()];`
`    let result = neighbour(&v, 0);`
`    println!("{:?}",result);`
`}`
	`Ok("helloworld")`

`fn main() {`
`    let v = vec!["hello".to_string(), "world".to_string(), "how".to_string(),`
`"are".to_string(), "you".to_string()];`
`    let result = neighbour(&v, 3);`
`    println!("{:?}",result);`
`}`
	`Ok("areyou")`

`fn main() {`
`    let v = vec!["hello".to_string(), "world".to_string(), "how".to_string(),`
`"are".to_string(), "you".to_string()];`
`    let result = neighbour(&v, 4);`
`    println!("{:?}",result);`
`}`
	`Err(())`

Feedback

Test	Expected	Got

`fn main() {`
`    let v = vec!["hello".to_string(), "world".to_string(),`
`"how".to_string(), "are".to_string(), "you".to_string()];`
`    let result = neighbour(&v, 0);`
`    println!("{:?}",result);`
`}`
	`Ok("helloworld")`	`Ok("helloworld")`

`fn main() {`
`    let v = vec!["hello".to_string(), "world".to_string(),`
`"how".to_string(), "are".to_string(), "you".to_string()];`
`    let result = neighbour(&v, 3);`
`    println!("{:?}",result);`
`}`
	`Ok("areyou")`	`Ok("areyou")`

`fn main() {`
`    let v = vec!["hello".to_string(), "world".to_string(),`
`"how".to_string(), "are".to_string(), "you".to_string()];`
`    let result = neighbour(&v, 4);`
`    println!("{:?}",result);`
`}`
	`Err(())`	`Err(())`

Domanda 17

write a function `removeelement` that takes as input a `&mut Vec<Option<i32>>` and removes the first `None` from the vector OR the first odd element, whichever comes first.

For example:

Test	Result

`fn main() {`
`  let mut v = vec![Some(1),Some(2),None,Some(3)];`
`  removeelement(&mut v);`
`  println!("{:?}",v);`
`}`
	`[Some(2), None, Some(3)]`

`fn main() {`
`  let mut v = vec![None,Some(2),None, None, Some(5)];`
`  removeelement(&mut v);`
`  println!("{:?}",v);`
`}`
	`[Some(2), None, None, Some(5)]`

Feedback

Test	Expected	Got

`fn main() {`
`  let mut v =`
`vec![Some(1),Some(2),None,Some(3)];`
`  removeelement(&mut v);`
`  println!("{:?}",v);`
`}`
	`[Some(2), None, Some(3)]`	`[Some(2), None, Some(3)]`

`fn main() {`
`  let mut v = vec![None,Some(2),None, None,`
`Some(5)];`
`  removeelement(&mut v);`
`  println!("{:?}",v);`
`}`
	`[Some(2), None, None, Some(5)]`	`[Some(2), None, None, Some(5)]`

Domanda 18

write a function `hashandhash` that takes a Hashmap<i32,String> `h1` and a Hashmap<String,i32> `h2`

and removes all elements from `h2` hashmap where the length of their keys is a key in `h1`

For example:

Test	Result

`use std::fmt::Debug;`
`fn makehashmap()->HashMap<i32,String>{`
`  let mut h = HashMap::new();`
`  h.insert(3,"what1".to_string());`
`  h.insert(4,"what2".to_string());`
`  h.insert(1,"what3".to_string());`
`  h.insert(6,"what4".to_string());`
`  h.insert(22,"what78".to_string());`
`  return h;`
`}`
`fn deterministicprinter<T,U>(h:&HashMap<T,U>) where T :`
`Debug + Ord, U : Debug + Ord{`
`  let mut v : Vec<(&T,&U)> = h.iter().collect();`
	`[(1, "what3"), (3, "what1"), (4, "what2"),`
	`(6, "what4"), (22, "what78")]`
	`[("w", 2), ("wh", 4), ("wha", 1), ("what",`
	`8), ("what1", 3)]`
	`[(1, "what3"), (3, "what1"), (4, "what2"),`
	`(6, "what4"), (22, "what78")]`
	`[("wh", 4), ("what1", 3)]`

Test **Result**

```
    v.sort();
    println!("{:?}",v);
}


fn main() {
    let mut h2: HashMap<String,i32> = HashMap::new();
    let mut h1 = makehashmap();
    h2.insert("w".to_string(), 2);
    h2.insert("wh".to_string(), 4);
    h2.insert("wha".to_string(), 1);
    h2.insert("what".to_string(), 8);
    h2.insert("what1".to_string(), 3);
    deterministicprinter(&h1);
    deterministicprinter(&h2);
    hashandhash(&mut h1,&mut h2);
    deterministicprinter(&h1);
    deterministicprinter(&h2);
}


use std::fmt::Debug;
fn makehashmap()->HashMap<i32,String>{
    let mut h = HashMap::new();
    h.insert(3,"what1".to_string());
    h.insert(4,"what2".to_string());
    h.insert(1,"what3".to_string());
    h.insert(6,"what4".to_string());
    h.insert(22,"what78".to_string());
    return h;
}
fn deterministicprinter<T,U>(h:&HashMap<T,U>) where T :
Debug + Ord, U : Debug + Ord{
    let mut v : Vec<(&T,&U)> = h.iter().collect();
    v.sort();                                                    [(1, "what3"), (3, "what1"), (4, "what2"),
    println!("{:?}",v);                                          (6, "what4"), (22, "what78")]
}                                                                [("wheeeee", 2), ("who", 1), ("whoo", 4),
                                                                 ("whoooo", 8), ("whooooooooooooooooooo",
                                                                 2)]
                                                                 [(1, "what3"), (3, "what1"), (4, "what2"),
                                                                 (6, "what4"), (22, "what78")]
fn main() {                                                      [("wheeeee", 2)]
    let mut h2: HashMap<String,i32> = HashMap::new();
    let mut h1 = makehashmap();
    h2.insert("whooooooooooooooooooo".to_string(), 2);
    h2.insert("whoo".to_string(), 4);
    h2.insert("who".to_string(), 1);
    h2.insert("whoooo".to_string(), 8);
    h2.insert("wheeeee".to_string(), 2);
    deterministicprinter(&h1);
    deterministicprinter(&h2);
    hashandhash(&mut h1,&mut h2);
    deterministicprinter(&h1);
    deterministicprinter(&h2);
}
```

Feedback

Test	Expected	Got

`use std::fmt::Debug;`
`fn makehashmap()->HashMap<i32,String>{`
`  let mut h = HashMap::new();`
`  h.insert(3,"what1".to_string());`
`  h.insert(4,"what2".to_string());`
`  h.insert(1,"what3".to_string());`
`  h.insert(6,"what4".to_string());`
`  h.insert(22,"what78".to_string());`
`  return h;`
`}`
`fn deterministicprinter<T,U>(h:&HashMap<T,U>)`
`where T : Debug + Ord, U : Debug + Ord{`
`  let mut v : Vec<(&T,&U)> = h.iter().collect();`
`  v.sort();`
`  println!("{:?}",v);`
`}`
 	
`fn main() {`
`  let mut h2: HashMap<String,i32> = HashMap::new();`
`  let mut h1 = makehashmap();`
`  h2.insert("w".to_string(), 2);`
`  h2.insert("wh".to_string(), 4);`
`  h2.insert("wha".to_string(), 1);`
`  h2.insert("what".to_string(), 8);`
`  h2.insert("what1".to_string(), 3);`
`  deterministicprinter(&h1);`
`  deterministicprinter(&h2);`
`  hashandhash(&mut h1,&mut h2);`
`  deterministicprinter(&h1);`
`  deterministicprinter(&h2);`
`}`
	`[(1, "what3"), (3, "what1"), (4, "what2"), (6, "what4"), (22, "what78")]`
	`[("w", 2), ("wh", 4), ("wha", 1), ("what", 8), ("what1", 3)]`
	`[(1, "what3"), (3, "what1"), (4, "what2"), (6, "what4"), (22, "what78")]`
	`[("wh", 4), ("what1", 3)]`
		`[(1, "what3"), (3, "what1"), (4, "what2"), (6, "what4"), (22, "what78")]`
		`[("w", 2), ("wh", 4), ("wha", 1), ("what", 8), ("what1", 3)]`
		`[(1, "what3"), (3, "what1"), (4, "what2"), (6, "what4"), (22, "what78")]`
		`[("wh", 4), ("what1", 3)]`

`use std::fmt::Debug;`
`fn makehashmap()->HashMap<i32,String>{`
`  let mut h = HashMap::new();`
`  h.insert(3,"what1".to_string());`
`  h.insert(4,"what2".to_string());`
`  h.insert(1,"what3".to_string());`
`  h.insert(6,"what4".to_string());`
`  h.insert(22,"what78".to_string());`
`  return h;`
`}`
`fn deterministicprinter<T,U>(h:&HashMap<T,U>)`
`where T : Debug + Ord, U : Debug + Ord{`
`  let mut v : Vec<(&T,&U)> = h.iter().collect();`
`  v.sort();`
`  println!("{:?}",v);`
`}`
	`[(1, "what3"), (3, "what1"), (4, "what2"), (6, "what4"), (22, "what78")]`
	`[("wheeeee", 2), ("who", 1), ("whoo", 4), ("whoooo", 8), ("whooooooooooooooooooo", 2)]`
	`[(1, "what3"), (3, "what1"), (4, "what2"), (6, "what4"), (22, "what78")]`
	`[("wheeeee", 2)]`
		`[(1, "what3"), (3, "what1"), (4, "what2"), (6, "what4"), (22, "what78")]`
		`[("wheeeee", 2), ("who", 1), ("whoo", 4), ("whoooo", 8), ("whooooooooooooooooooo", 2)]`
		`[(1, "what3"), (3, "what1"), (4, "what2"), (6, "what4"), (22, "what78")]`
		`[("wheeeee", 2)]`

Test	Expected	Got

```
fn main() {
    let mut h2: HashMap<String,i32> =
HashMap::new();
    let mut h1 = makehashmap();

h2.insert("whooooooooooooooooooo".to_string(),
2);
    h2.insert("whoo".to_string(), 4);
    h2.insert("who".to_string(), 1);
    h2.insert("whoooo".to_string(), 8);
    h2.insert("wheeeee".to_string(), 2);
    deterministicprinter(&h1);
    deterministicprinter(&h2);
    hashandhash(&mut h1,&mut h2);
    deterministicprinter(&h1);
    deterministicprinter(&h2);
}
```

Domanda 19

write a function `unique` that takes a Hashmap<i32,String> `h` and an i32 `l` and returns the optional `h` with a new entry whose value is a `l` long string of 'X', unless there is a string `s` already present in `h` whose length is `l` (when adding a value, do it with key equals to the amount of entries in the hashmap)

For example:

Test	Result

```
use std::fmt::{Debug, Error, Formatter};
fn makehashmap()->HashMap<i32,String>{
    let mut h = HashMap::new();
    h.insert(2,"what1".to_string());
    h.insert(4,"what2".to_string());
    h.insert(1,"what3".to_string());
    h.insert(5,"what".to_string());
    return h;
}
fn deterministicprinter<T,U>(h:&HashMap<T,U>) where T : Debug + Ord,
U : Debug + Ord{
    let mut v : Vec<(&T,&U)> = h.iter().collect();
    v.sort();
    println!("{:?}",v);
}
// ----------------------------------------------------
fn main() {
    let mut h1 = makehashmap();
    deterministicprinter(&h1);
    let ret = unique(h1, 5);
    println!("{:?}",ret);
    let mut h1 = makehashmap();
    deterministicprinter(&h1);
    let ret = unique(h1, 2).unwrap();
```

```
[(1, "what3"), (2, "what1"),
(4, "what2"), (5, "what")]
None
[(1, "what3"), (2, "what1"),
(4, "what2"), (5, "what")]
[(1, "what3"), (2, "what1"),
(4, "XX"), (5, "what")]
```

```
    deterministicprinter(&ret);
}
```