```
class Solution {
public:

    int reverse(int a) {
        int b = 0;
        while (a > 0) {
            b = b * 10 + a % 10;
            a /= 10;
        }
        return b;
    }



    int minMirrorPairDistance(vector<int>& nums) {
        vector<int>A=nums;
         map<int, int> pre;
        int n = A.size(), res = n;
        for (int i = 0; i < n; ++i) {
            if (pre.count(A[i])) {
                res = min(res, i - pre[A[i]]);
            }
            pre[reverse(A[i])] = i;
        }

        return (res < n) ? res : -1;
    }
};
```

can u change that fn with lambda fn make reverse as lambda fn

Here you go — **minimal change**, just replace the `reverse()` function with a **lambda** inside `minMirrorPairDistance`.
Everything else untouched.

✅ Updated Code (with lambda reverse)

cpp Copy code

```cpp
class Solution {
public:
    int minMirrorPairDistance(vector<int>& nums) {

        // Lambda reverse fn
        auto rev = [&](int a) {
            int b = 0;
            while (a > 0) {
                b = b * 10 + a % 10;
                a /= 10;
            }
            return b;
        };

        vector<int> A = nums;
        map<int, int> pre;
        int n = A.size(), res = n;

        for (int i = 0; i < n; ++i) {
            if (pre.count(A[i])) {
                res = min(res, i - pre[A[i]]);
            }
            pre[rev(A[i])] = i;   // using lambda here
        }

        return (res < n) ? res : -1;
    }
};
```

✔️ What changed?

- Removed the old `reverse()` member function.
- Added a **lambda**:

cpp Copy code

```cpp
auto rev = [&](int a) { ... };
```

- Then used `rev(A[i])` where required.

If you want, I can also show how to make the lambda `static` or move it outside the class.

can you just tell me about lambda function in cpp in details
-syntax,conditions,constraint,all terminology involved
-scenario in cp/dsa where to use most used cases
-and take some example and perfom some operation in details

Here is **the cleanest, CP-focused, no-nonsense explanation** of lambda functions in C++.
Everything you need: **syntax** → **terminology** → **rules** → **when to use** → **CP examples**.

⭐ 1. What is a Lambda Function?

A **lambda** is an **inline, anonymous function** (a function without a name).

In CP, it is mainly used for:

- Custom sorting (`sort` with comparator)
- Passing small utility functions (reverse digits, gcd, check fn...)
- Capturing variables from outer scope
- Using inside STL algorithms (`for_each`, `count_if`, etc.)

⭐ 2. Syntax (Full Form)

cpp Copy code

```cpp
[capture_list] (parameters) -> return_type {
    // body
};
```

Example:

cpp Copy code

```cpp
auto add = [&](int a, int b) -> int {
    return a + b;
};
```

⭐ 3. Breakdown of Terminology

✔️ (1) Capture List `[]`

Specifies what variables from outside the lambda are allowed inside it.

Most used in CP:

Capture	Meaning
`[]`	captures nothing
`[&]`	captures **everything by reference**
`[=]`	captures **everything by value**
`[&x]`	captures only x by reference
`[x]`	captures only x by value

CP uses **[&]** the most.

✔️ (2) Parameters `( )`

Exactly like normal functions.

✔️ (3) Return Type `-> type`

Often optional.
Compiler deduces automatically if the lambda has a single `return`.

```cpp
auto f = [&](int x){ return x*x; };
```

✔️ (4) Body `{ }`

Just code.

⭐ 4. Constraints / Rules You Must Know

1) Lambda cannot be recursive unless using `std::function`

Wrong:

```cpp
auto f = [&](int n){ return (n==0?1:f(n-1)); };  // ERROR
```

Correct:

```cpp
function<int(int)> f = [&](int n){
    return (n==0 ? 1 : f(n-1));
};
```

---

2) Lambda cannot modify value-captured variables

```cpp
int x = 10;
auto f = [=](){ x++; }; // error (x captured by value)
```

---

3) Lambda can modify ref-captured variables

```cpp
int x = 10;
auto f = [&](){ x++; }; // OK
```

---

4) Lambdas have internal type (compiler-generated)

That's why we store them using `auto`.

⭐ 5. When to Use Lambda in CP/DSA (Most Common Scenarios)

⚡ 1. Custom Sorting (MOST common)

cpp Copy code

```cpp
sort(v.begin(), v.end(), [&](int a, int b){
    return a > b;
});
```

⚡ 2. Short utility functions (e.g., reverse digit)

Good for small 3-4 line operations:

cpp Copy code

```cpp
auto rev = [&](int x){
    int r=0;
    while(x){ r=r*10 + x%10; x/=10; }
    return r;
};
```

⚡ 3. Using inside STL algorithms

cpp Copy code

```cpp
int cnt = count_if(v.begin(), v.end(), [&](int x){
    return x % 2 == 0;
});
```

⚡ 4. DFS / BFS recursion with `std::function`

cpp Copy code

```cpp
function<void(int)> dfs = [&](int u){
    vis[u] = 1;
    for(int x: adj[u])
```

```
        if(!vis[x]) dfs(x);
};
```

⚡ 5. Custom priority queue comparator

cpp Copy code

```cpp
auto cmp = [&](int a, int b){
    return a > b; // min-heap
};

priority_queue<int, vector<int>, decltype(cmp)> pq(cmp);
```

⭐ 6. Most Useful Mini Examples with Explanation

✅ Example 1: Reverse digits using lambda (your case)

cpp Copy code

```cpp
auto reverse = [&](int a){
    int b = 0;
    while(a){
        b = b*10 + a%10;
        a/=10;
    }
    return b;
};
cout << reverse(123) << "\n";  // 321
```

✅ Example 2: Lambda with sort + custom condition

Sort by (1) frequency descending, (2) value ascending.

cpp Copy code

```cpp
vector<pair<int,int>> v = {{5,2},{7,2},{1,3},{3,1}};
```

```cpp
sort(v.begin(), v.end(), [&](auto &a, auto &b){
    if (a.second != b.second)
        return a.second > b.second;
    return a.first < b.first;
});
```

✅ Example 3: Using lambda inside count_if

Count numbers divisible by 3:

cpp

Copy code

```cpp
int ans = count_if(v.begin(), v.end(), [&](int x){
    return x % 3 == 0;
});
```

✅ Example 4: Lambda modifying outer variable

cpp

Copy code

```cpp
int sum = 0;

for_each(v.begin(), v.end(), [&](int x){
    sum += x;
});

cout << sum;
```

⭐ 7. CP Trick: Capture Everything by Reference `[&]`

Because CP code is short and inside functions, we do:

cpp

Copy code

```cpp
auto good = [&](int x){
    return x >= 0 && x < n;
};
```

This avoids writing `[&n]`.

⭐ 8. CP Trick: Inline Lambda

Sometimes you don't even store lambda:

```cpp
sort(v.begin(), v.end(), [](int a, int b){
    return abs(a) < abs(b);
});
```

⭐ 9. Heavy Example (Very CP Style)

Find first index where prefix sum > X.

```cpp
vector<int> pref(n);
partial_sum(a.begin(), a.end(), pref.begin());

int x; cin >> x;

auto check = [&](int i){
    return pref[i] > x;
};

int idx = lower_bound(pref.begin(), pref.end(), 0, [&](int p, int dummy){
    return check(p);
}) - pref.begin();
```

Shows you how lambdas can be nested.

⭐ FINAL SUMMARY (CP/DSA Version)

✔️ Syntax:

```css
[capture] (args) -> ret { body }
```

✔️ Capture:

- `[&]` = capture all by reference (CP default)
- `[=]` = all by value
- `[x]` / `[&x]` = capture specific variables

✔️ Best Uses in CP:

- Custom comparator for sorting
- Utility functions (reverse, gcd, check fn)
- DFS/BFS with recursion
- count_if, for_each, all_of, none_of
- Custom priority queue

If you want, I can also show:

- Lambdas with **mutable**
- Lambdas inside classes
- Lambdas returning lambdas (advanced)
- Run-time polymorphism using lambdas

Just tell me **"show advanced lambda stuff"**.

Lambdas with mutable

Lambdas inside classes
both

Below are **two advanced but extremely useful topics** in C++ lambdas:
✔ **mutable lambdas**
✔ **lambdas inside classes**

Both are explained in **CP + DSA style** with clean examples.

⭐ 1. Mutable Lambdas (CP/DSA Practical Explanation)

By default, C++ lambdas **cannot modify variables captured by value**.

Example:

cpp Copy code

```cpp
int x = 10;

auto f = [=]() {
    x++;   // ❌ ERROR
};
```

Because `[=]` captures **copy of x** → cannot modify.

✔ Why `mutable` Exists?

To allow modifications **INSIDE the lambda only**, without affecting original variables.

Syntax:

cpp Copy code

```cpp
[capture] () mutable {
    // modify captured-by-value variables
}
```

✔ Example: Normal case (no mutable)

cpp Copy code

```cpp
int x = 5;

auto f = [=]() {
    // cannot modify x
    // x++;   // ❌ error
};
```

✔ Example: Using `mutable`

cpp Copy code

```cpp
int x = 5;

auto f = [=]() mutable {
    x++;  // ✔ allowed
    cout << "Inside lambda x=" << x << "\n";
};

f();
cout << "Outside x=" << x << "\n";
```

Output:

python Copy code

```python
Inside lambda x=6
Outside x=5
```

⚠️ Important:

- Inside lambda → modify copy
- Outside → original unchanged

✔ When useful in CP?

When you want to modify temporary variables inside lambda but do NOT want to affect original ones.

Example: create a stateful comparator

cpp Copy code

```cpp
int cnt = 0;

sort(v.begin(), v.end(), [=](int a, int b) mutable {
    cnt++;  // ✔ allowed inside lambda, but outer cnt stays same
    return a < b;
});
```

✔ Another CP Example: Randomized comparator

cpp Copy code

```cpp
sort(v.begin(), v.end(), [rnd=rand()] (int a, int b) mutable {
    rnd = (rnd * 1103515245 + 12345) % (1<<31);
    return (rnd % 2) ? a < b : a > b;
});
```

`rnd` is modified inside lambda but outer variable doesn't change.

⭐ 2. Lambdas Inside Classes

There are 3 main ways lambdas are used inside classes:

✔ CASE 1: Lambda as a class variable

cpp Copy code

```cpp
class Solution {
public:
    function<int(int)> rev = [&](int a){
        int b=0;
        while(a){ b=b*10 + a%10; a/=10; }
        return b;
    };
};
```

⚠️ You must use **std::function** because lambda inside class captures `this`.

✔ CASE 2: Lambda inside a member function (MOST USED)

cpp Copy code

```cpp
class Solution {
public:
    int compute(vector<int>& v) {
        auto sum = [&](int i){
            return v[i] + (i>0 ? v[i-1] : 0);
        };

        return sum(v.size()-1);
    }
};
```

✔ No `std::function` needed

✔ Lambda has access to class variables (via `[&]` capture)

✔ CASE 3: Lambda accessing class member variables

cpp Copy code

```cpp
class Solution {
public:
    int n = 100;

    int work() {
        auto get = [&](int x){
            return x + n;   // ✔ can use class member variable
        };

        return get(20);  // 120
    }
};
```

Because `[&]` captures `this` implicitly.

✔ Case 4: Lambda stored in class + mutating captured members

cpp Copy code

```cpp
class Counter {
public:
    int val = 0;

    auto inc() {
        return [&](int x){
            val += x;   // modifies class member
            return val;
        };
    }
};
```

Usage:

cpp Copy code

```
Counter c;
auto f = c.inc();
cout << f(5);   // 5
cout << f(3);   // 8
```

⭐ Practical CP Example (VERY USEFUL)

Using lambda inside class to implement DFS

cpp Copy code

```
class Solution {
public:
    vector<vector<int>> adj;
    vector<int> vis;

    void solve(int n) {
        adj.resize(n);
        vis.assign(n, 0);

        // build graph ...

        function<void(int)> dfs = [&](int u){
            vis[u] = 1;
            for(int v : adj[u]) {
                if(!vis[v]) dfs(v);
            }
        };

        dfs(0);
    }
};
```

Why `std::function`?

- Because lambda is recursive → needs its own type (unknown)

⭐ CP Example: Comparator inside class

cpp Copy code

```cpp
class Solution {
public:
    void sortPairs(vector<pair<int,int>>& v) {

        sort(v.begin(), v.end(),
            [&](auto &a, auto &b){
                if(a.second != b.second) return a.second > b.second;
                return a.first < b.first;
            }
        );
    }
};
```

⭐ Summary Table

Feature	Purpose	CP Use Case
mutable	modify value-captured vars inside lambda	randomized comparator, temporary state
lambda inside class method	access class vars	DFS, helper fns
lambda stored in variable	reuse function	reverse digit func
recursive lambda (std::function)	recursion inside lambda	DFS/BFS/tree DP

If you want, I can also show:

✅ **Lambda returning lambda**

✅ **Lambda templates** (generic lambdas)

✅ **High-performance lambdas in DSA (no std::function)**

✅ **Monoids, segment tree with lambdas**