

LAMBDA EXPRESSION :-

```
#include<iostream>

int multiply(int a, int b);

int main(){

    std::cout<<multiply(4,5)<<std::endl;

    std::cout<<[](int a, int b){return a*b;}(4,5)<<std::endl;

    auto f = [](int a,int b){return a*b;};

    std::cout<<f(4,5)<<std::endl;

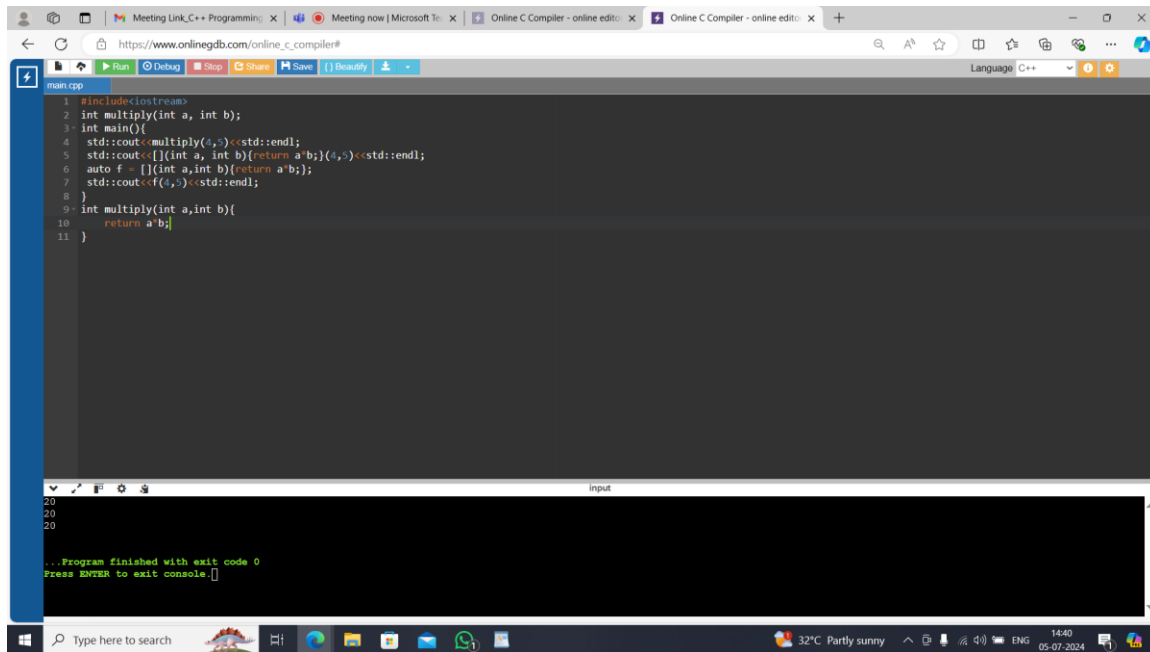
}

int multiply(int a,int b){

    return a*b;

}
```

OUTPUT :-

A screenshot of a web browser displaying an online C++ compiler. The browser's address bar shows the URL 'https://www.onlinegdb.com/online_c_compiler#'. The code editor contains the same C++ code as shown in the previous block. Below the code editor, the 'input' field is empty, and the 'output' field displays the results of the program execution: '20', '20', and '20' on three separate lines. At the bottom of the output area, it says '...Program finished with exit code 0' and 'Press ENTER to exit console.' The browser's taskbar at the bottom shows the system clock as 14:40 on 05-07-2024, with a temperature of 32°C and a weather forecast of 'Partly sunny'.

CAPTURE BY VALUE :-

```
#include <iostream>

void lambda_value_capture() {
```

```

int value = 1;

auto copy_value = [value] {

    return value;

};

value = 100;

auto stored_value = copy_value();

std::cout << "stored_value = " << stored_value << std::endl;

}

int main() {

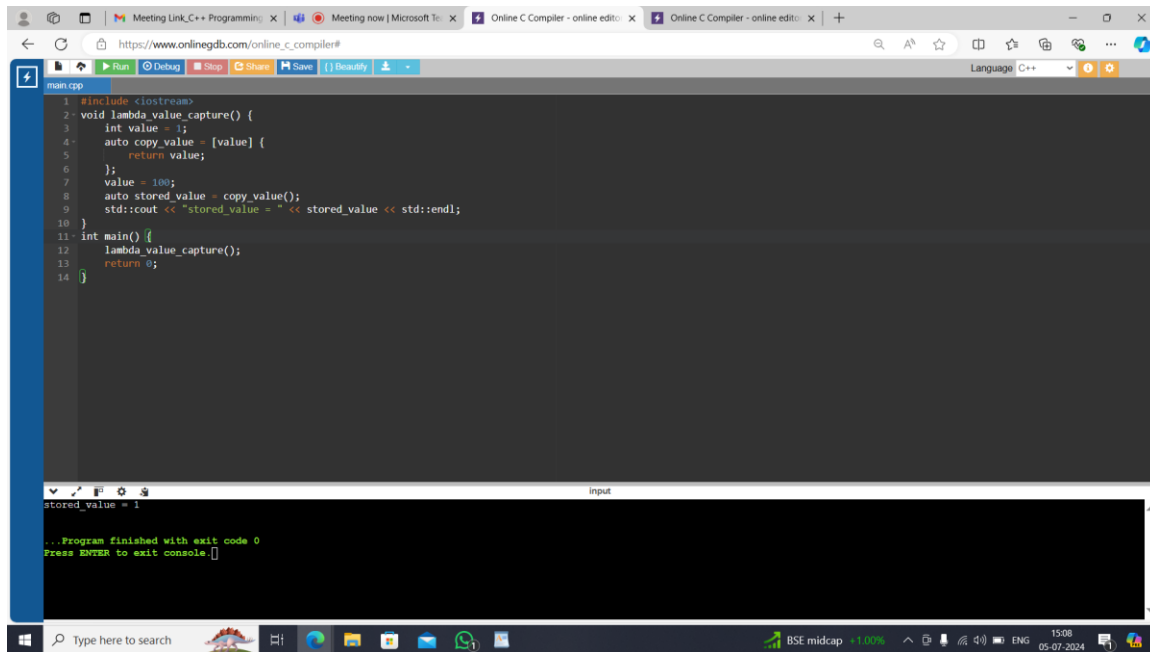
    lambda_value_capture();

    return 0;

}

```

OUTPUT :-



The screenshot shows a web browser window with the URL https://www.onlinegdb.com/online_c_compiler#. The code editor contains the following C++ code:

```

1 #include <iostream>
2 void lambda_value_capture() {
3     int value = 1;
4     auto copy_value = [value] {
5         return value;
6     };
7     value = 100;
8     auto stored_value = copy_value();
9     std::cout << "stored_value = " << stored_value << std::endl;
10 }
11 int main() {
12     lambda_value_capture();
13     return 0;
14 }

```

The output window shows the result of the program execution:

```

stored_value = 1

...Program finished with exit code 0
Press ENTER to exit console.

```

REFERENCE CAPTURE :-

```

#include <iostream>

void lambda_reference_capture() {

```

```

int value = 1;

auto copy_value = [&value] {

    return value;

};

value = 100;

auto stored_value = copy_value();

std::cout << "stored_value = " << stored_value << std::endl;

}

int main() {

    lambda_reference_capture();

    return 0;

}

```

OUTPUT :-

The screenshot shows a web browser window with the URL https://www.onlinegdb.com/online_c_compiler#. The code editor contains the following C++ code:

```

1 #include <iostream>
2 void lambda_reference_capture() {
3     int value = 1;
4     auto copy_value = [&value] {
5         return value;
6     };
7     value = 100;
8     auto stored_value = copy_value();
9     std::cout << "stored_value = " << stored_value << std::endl;
10 }
11 int main() {
12     lambda_reference_capture();
13     return 0;
14 }

```

The output console shows:

```

stored_value = 100

...Program finished with exit code 0
Press ENTER to exit console.

```

CAPTURE BY BOTH (VALUE & REFERENCE) :-

```

#include <iostream>

using namespace std;

```

```

int main()
{
    int m = 0;

    int n = 0;

    [&, n] (int a) mutable {m = ++n + a;}(4);

    cout << m << endl << n << endl;

}

```

OUTPUT :-

The screenshot shows a web browser window with the URL https://www.onlinegdb.com/online_c_compiler#. The code editor contains the following C++ code:

```

1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int m = 0;
6     int n = 0;
7     [&, n] (int a) mutable {m = ++n + a;}(4);
8     cout << m << endl << n << endl;
9 }

```

The output window shows the following text:

```

4
0
...Program finished with exit code 0
Press ENTER to exit console.

```

USE CASE :-

```

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

void assign(int& v)

{

    static int n=1; v= n++;
}

```

```

}

void print(int v){

    cout<<v<<" ";

}

int main(){

    vector<int>vec(10);

    for_each(vec.begin(),vec.end(),print);

    cout<<endl;

    for_each(vec.begin(),vec.end(),assign);

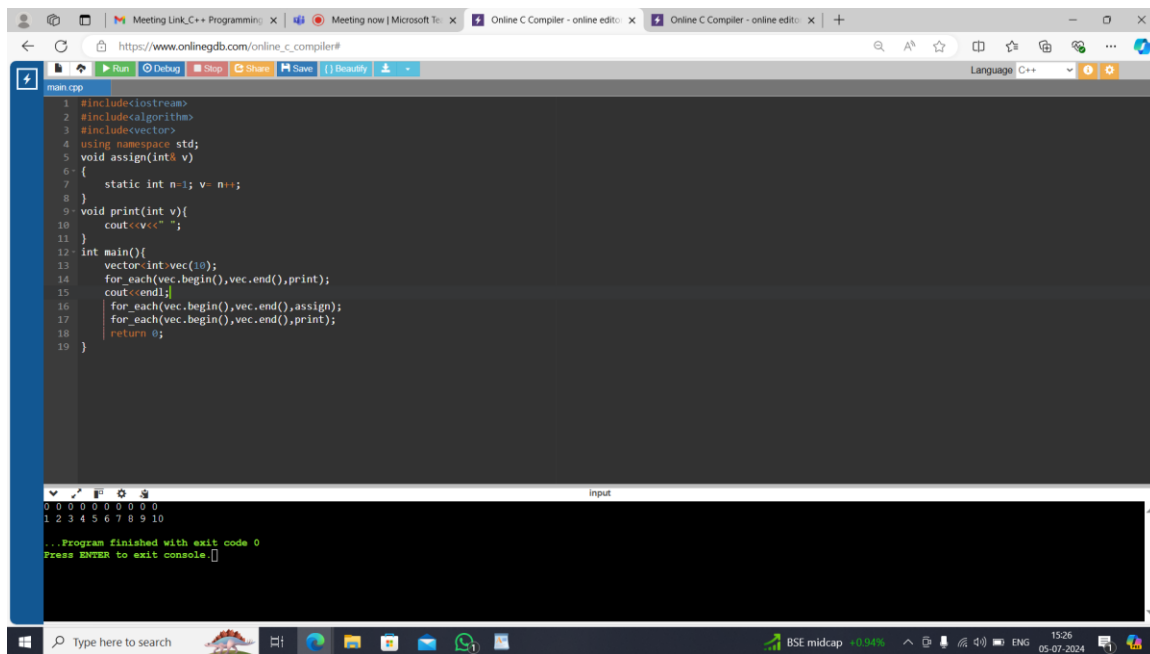
    for_each(vec.begin(),vec.end(),print);

    return 0;

}

```

OUTPUT :-



The screenshot shows a web browser window with the URL https://www.onlinegdb.com/online_c_compiler#. The code editor contains the following C++ code:

```

1 #include<iostream>
2 #include<algorithm>
3 #include<vector>
4 using namespace std;
5 void assign(int& v)
6 {
7     static int n=1; v= n++;
8 }
9 void print(int v){
10     cout<<v<<" ";
11 }
12 int main(){
13     vector<int>vec(10);
14     for_each(vec.begin(),vec.end(),print);
15     cout<<endl;
16     for_each(vec.begin(),vec.end(),assign);
17     for_each(vec.begin(),vec.end(),print);
18     return 0;
19 }

```

The output window shows the following results:

```

0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 10
...Program finished with exit code 0
Press ENTER to exit console.

```

The Windows taskbar at the bottom shows the system clock as 15:26 on 05-07-2024.

Practice Problem Statement:

Scenario: You're working on a data analysis project where you need to filter a list of integers based on whether they are even or odd. You want to use a lambda expression to achieve this filtering.

Task:

Define a function named `filter_even_odds` that takes two arguments:

`const std::vector<int>& numbers`: The vector containing the integer values.

`bool is_even`: A flag indicating whether to filter even (true) or odd (false) numbers.

Inside the function, use a lambda expression to iterate through the numbers vector.

Within the lambda, check if the current number is even using the modulo operator (%).

If the even/odd condition matches the `is_even` flag, add the number to a new filtered vector.

Return the filtered vector from the `filter_even_odds` function.

```
#include <iostream>

#include <vector>

#include <algorithm>

std::vector<int> filter_even_odds(const std::vector<int>& numbers, bool is_even) {

    std::vector<int> filtered;

    auto filter_lambda = [&](int number) { // Lambda function to filter even or odd
numbers based on is_even flag
        if ((number % 2 == 0 && is_even) || (number % 2 != 0 && !is_even)) {
            return true;
        }
        return false;
    };

    std::copy_if(numbers.begin(), numbers.end(), std::back_inserter(filtered), filter_lambda);
    // Using std::copy_if with lambda function to copy filtered elements to new vector

    return filtered;
}

int main() {

    std::vector<int> numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    std::vector<int> even_numbers = filter_even_odds(numbers, true); // Filter even
numbers
```

```

std::cout << "Even numbers: ";
for (int num : even_numbers) {
    std::cout << num << " ";
}
std::cout << std::endl;

std::vector<int> odd_numbers = filter_even_odds(numbers, false);           // Filter odd
numbers

std::cout << "Odd numbers: ";
for (int num : odd_numbers) {
    std::cout << num << " ";
}
std::cout << std::endl;
return 0;
}

```

OUTPUT :-

2. Finding Maximum Value:

Scenario: You have a list of objects and want to find the object with the highest value based on a specific criterion.

Task:

Define a function named `find_max` that takes two arguments:

`const std::vector<T>& objects:` The vector containing the objects (can be any type T).

`std::function<bool(const T& a, const T& b)> compare:` A function object (e.g., a lambda) that defines the comparison logic for finding the maximum.

Inside the function, use a `std::accumulate` with a lambda expression to iterate through the objects vector.

Within the inner lambda, compare the current element with the current maximum using the provided compare function.

If the current element is greater (based on the comparison logic), return it as the new maximum.

```
#include <iostream>
```

```
#include <vector>
```

```
#include <functional>
```

```
#include <algorithm>
```

```
template<typename T>
```

```
const T& find_max(const std::vector<T>& objects, std::function<bool(const T&, const T&)> compare) {
```

```
    // Check if vector is empty, though ideally, it should not be in your case
```

```
    if (objects.empty()) {
```

```
        throw std::invalid_argument("Empty vector passed to find_max");
```

```
    }
```

```
    // Use std::accumulate with a lambda to find the maximum element
```

```
    return std::accumulate(objects.begin() + 1, objects.end(), objects[0],
```

```
        [&](const T& current_max, const T& obj) {
```

```
            if (compare(obj, current_max)) {
```

```
                return obj;
```

```
            } else {
```

```
                return current_max;
```

```
            }
```

```
        });
```

```
}
```

```
int main() {
```

```
    std::vector<int> numbers = { 3, 9, 1, 4, 7, 5 };
```

```
    // Lambda to compare integers (find maximum)
```



```
auto max_int = find_max(numbers, [](const int& a, const int& b) {  
    return a > b; // Return true if a is greater than b  
});  
  
std::cout << "Max integer: " << max_int << std::endl;  
  
// Lambda to compare strings (find maximum based on length)  
std::vector<std::string> words = { "apple", "banana", "orange", "pear" };  
auto max_string = find_max(words, [](const std::string& a, const std::string& b) {  
    return a.length() > b.length(); // Return true if length of a is greater than length of b  
});  
std::cout << "Longest word: " << max_string << std::endl;  
return 0;  
}
```