**std::transform() in C++ STL (Perform an operation on all elements)**

// Using transform() in STL to add two arrays

#include <iostream>

#include <algorithm>

using namespace std;

int main()

{

int arr1[] = {1, 2, 3};

int arr2[] = {4, 5, 6};

int n = sizeof(arr1)/sizeof(arr1[0]);

int res[n];

// Single line code to add arr1[] and arr2[] and

// store result in res[]

transform(arr1, arr1+n, arr2, res, plus<int>()); // plus is predicate fn

for (int i=0; i<n; i++)

cout << res[i] << " ";

}

### **transform() in C++ is used in two forms:**

**1. Unary Operation** : Applies a unary operator on input to convert into output

transform(Iterator inputBegin, Iterator inputEnd,

Iterator OutputBegin, unary\_operation)

// C++ program to demonstrate working of transform with unary operator.

#include <bits/stdc++.h>

using namespace std;

int increment(int x) { return (x+1); }

int main()

{

int arr[] = {1, 2, 3, 4, 5};

int n = sizeof(arr)/sizeof(arr[0]);

// Apply increment to all elements of arr[] and store the modified elements b ack in arr[]

transform(arr, arr+n, arr, increment);

for (int i=0; i<n; i++)

cout << arr[i] << " ";

return 0;

}

**// C++ program to demonstrate working of functors.**

**#include <bits/stdc++.h>**

**using namespace std;**

**// A Functor**

**class increment**

**{**

**private:**

**int num;**

**public:**

**increment(int n) : num(n) { }**

**// This operator overloading enables calling**

**// operator function () on objects of increment**

**int operator () (int arr\_num) const {**

**return num + arr\_num;**

**}**

**};**

**// Driver code**

**int main()**

**{**

**int arr[] = {1, 2, 3, 4, 5};**

**int n = sizeof(arr)/sizeof(arr[0]);**

**int to\_add = 5;**

**transform(arr, arr+n, arr, increment(to\_add));**

**for (int i=0; i<n; i++)**

**cout << arr[i] << " ";**

**}**

Predefiend functors in c++

<https://www.scaler.com/topics/cpp-functor/>

Here's an example that transforms each string in a vector to its uppercase form:

#include <iostream>

#include <algorithm>

#include <vector>

#include <string>

#include <cctype>

int main() {

std::vector<std::string> v1 = {"abc", "def", "ghi"};

std::vector<std::string> v2(v1.size());

std::transform(v1.begin(), v1.end(), v2.begin(), [](const std::string& s) {

std::string result;

for (char c : s) {

result += std::toupper(c);

}

return result;

});

for (const std::string& str : v2) {

std::cout << str << " ";

}

return 0;

}

#include <iostream>

#include <algorithm>

#include <vector>

int main() {

// Initializing two vectors

std::vector<int> v1 = {1, 2, 3, 4, 5};

std::vector<int> v2(v1.size());

// Using std::transform to perform element-wise addition

std::transform(v1.begin(), v1.end(), v2.begin(), [](int x) { return x + 1; });

// Printing the result

for (int num : v2) {

std::cout << num << " ";

}

return 0;

}

In C++, a predicate function is a function that takes one or more arguments and returns a boolean value. Predicates are commonly used in algorithms and functions that require a condition to be checked. For example, predicates are often used with standard algorithms like `std::find\_if`, `std::remove\_if`, and `std::sort` to specify criteria for searching, removing, or sorting elements.

Here's an example of a predicate function in C++:

```cpp

#include <iostream>

#include <vector>

#include <algorithm>

// Predicate function to check if a number is even

bool isEven(int number) {

return number % 2 == 0;

}

int main() {

// Creating a vector of integers

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

// Using std::find\_if with the isEven predicate

auto it = std::find\_if(numbers.begin(), numbers.end(), isEven);

// Checking if an even number was found

if (it != numbers.end()) {

std::cout << "Found the first even number: " << \*it << std::endl;

} else {

std::cout << "No even numbers found." << std::endl;

}

return 0;

}

```

In this example, `isEven` is a predicate function that checks if a given number is even. It is then used as an argument to `std::find\_if` to search for the first even number in a vector.

Predicates can take various forms based on the context in which they are used. They can be unary or binary, depending on the algorithm's requirements. Unary predicates take a single argument, and binary predicates take two arguments. The key characteristic is that they return a boolean value indicating whether a certain condition is met.

Here's an example of a binary predicate used with `std::sort`:

```cpp

#include <iostream>

#include <vector>

#include <algorithm>

// Binary predicate function to compare integers based on their absolute values

bool compareAbs(int a, int b) {

return std::abs(a) < std::abs(b);

}

int main() {

std::vector<int> numbers = {-3, 1, -5, 2, -8, 4};

// Using std::sort with the compareAbs predicate

std::sort(numbers.begin(), numbers.end(), compareAbs);

// Printing the sorted vector

for (int num : numbers) {

std::cout << num << " ";

}

return 0;

}

```

In this example, `compareAbs` is a binary predicate that compares integers based on their absolute values. It is used with `std::sort` to sort a vector of integers in ascending order of absolute values.