std::accumulate

Определено в заголовочном файле <numeric>

```
template< class InputIt, class T >
T accumulate( InputIt first, InputIt last, T init );                    (1)

template< class InputIt, class T, class BinaryOperation >
T accumulate( InputIt first, InputIt last, T init,                      (2)
              BinaryOperation op );
```

Функция *accumulate* считает сумму значений *val* и всех элементов в диапазоне `[first, last)`. The first version uses `operator+` to sum up the elements, the second version uses the given binary function `op`.

Параметры

first, last — диапазон элементов в сумме

init — initial value of the sum

op — binary operation function object that will be applied.

The signature of the function should be equivalent to the following:

```
Ret fun(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`.
Тип `Type1` должен быть таков, что объект типа `T` может быть неявно преобразован в `Type1`. Тип `Type2` должен быть таков, что объект типа `InputIt` может быть разыменован и затем неявно преобразован в `Type2`. Тип `Ret` должен быть таков, что объекту типа `T` можно присвоить значение типа `Ret`.

Требования к типам

- `InputIt` должен соответствовать требованиям `InputIterator`.
- `T` должен соответствовать требованиям `CopyAssignable` и `CopyConstructible`.

Возвращаемое значение

The sum of the given value and elements in the given range.

Возможная реализация

Первый вариант

```
template<class InputIt, class T>
T accumulate(InputIt first, InputIt last, T value)
{
    for (; first != last; ++first) {
        value = value + *first;
    }
    return value;
}
```

Второй вариант

```
template<class InputIt, class T, class BinaryOperation>
T accumulate(InputIt first, InputIt last, T value,
             BinaryOperation op)
{
    for (; first != last; ++first) {
        value = op(value, *first);
    }
    return value;
}
```

Пример

Запустить этот код

```
#include <iostream>
#include <vector>
#include <numeric>
#include <string>

int multiply(int x, int y)
{
    return x*y;
}

std::string magic_function(std::string res, int x)
{
    return res += (x > 5) ? "b" : "s";
}

int main()
{
    std::vector<int> v{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    int sum = std::accumulate(v.begin(), v.end(), 0);
    int product = std::accumulate(v.begin(), v.end(), 1, multiply);
    std::string magic = std::accumulate(v.begin(), v.end(), std::string(),
                                        magic_function);

    std::cout << sum << '\n'
              << product << '\n'
              << magic << '\n';
}
```

Вывод:

```
55
3628800
sssssbbbbb
```

См. также

adjacent_difference	вычисляет разницу между соседними элементами в диапазоне (шаблон функции) [править] (https://ru.cppreference.com/mwiki/index.php?title=%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD:cpp/algorithm/dsc_adjacent_difference&action=edit)
inner_product	вычисляет скалярное произведение двух диапазонах элементы (шаблон функции) [править] (https://ru.cppreference.com/mwiki/index.php?title=%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD:cpp/algorithm/dsc_inner_product&action=edit)
partial_sum	вычисляет частичную сумму ряда элементов (шаблон функции) [править] (https://ru.cppreference.com/mwiki/index.php?title=%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD:cpp/algorithm/dsc_partial_sum&action=edit)

Источник — «https://ru.cppreference.com/mwiki/index.php?title=cpp/algorithm/accumulate&oldid=41186»