Языки программирования
Лекция 12

ПМИ Семестр 2

Демяненко Я.М.

2024

Обобщенность алгоритмов

Определение алгоритмов в терминах итераторов, а не непосредственно структур данных, делает возможной независимость алгоритмов от последних.

Так **алгоритмы** становятся **особенно** "**обобщенными**", способными работать с большим количеством разных структур данных.

Ни одна другая библиотека не способна на такое расширение, как STL.

Алгоритмы STL

- Алгоритмы представляют собой внешние функции
- Большинство алгоритмов определено в заголовочных файлах <algorithm> и <functional>

algorithm

algorithm — заголовочный файл в стандартной библиотеке языка программирования C++, включающий набор **функций** для выполнения **алгоритмических операций** над **контейнерами** и над другими последовательностями

functional

functional — заголовочный файл в стандартной библиотеке языка программирования C++, предоставляющий набор шаблонов классов для работы с **функциональными объектами**, а также набор **вспомогательных классов** для их **использования в алгоритмах** стандартной библиотеки

Основные группы алгоритмов

- Алгоритмы можно разбить на следующие группы:
  - Алгоритмы, не модифицирующие последовательность
  - Алгоритмы, модифицирующие последовательность
  - Алгоритмы сортировки

Алгоритмы, не модифицирующие последовательность

all_of	Test condition on all elements in range
any_of	Test if any element in range fulfills condition
none_of	Test if no elements fulfill condition
for_each	Apply function to range
find	Find value in range
find_if	Find element in range
find_if_not	Find element in range (negative condition)
find_end	Find last subsequence in range
find_first_of	Find element from set in range
adjacent_find	Find equal adjacent elements in range
count	Count appearances of value in range
count_if	Return number of elements in range satisfying condition
mismatch	Return first position where two ranges differ
equal	Test whether the elements in two ranges are equal
is_permutation	Test whether range is permutation of another
search	Search range for subsequence
search_n	Search range for elements

for_each()

Apply function to range **Применить функцию к диапазону**

```
void MyFunc(MyType x);
vector<MyType> v;

vector<MyType>::iterator p;
for (p=v.begin(); p!= v.end(); ++p)
   MyFunc(*p);

//или
```

Использование

```
for_each (begin(v), end(v), MyFunc);
```

Возможная реализация

```
template<typename InIt, typename Fun>
Fun for_each(InIt b, InIt e, Fun f) {
   while(b != e)    {
      f(*b);
      ++b;
   }
   return f;
}
```

Использование

```
Fun for_each(InIt first, InIt last, Fun f);

InIt = list<int>::iterator;
Fun = void(*)(int);

// Печать элемента контейнера
void print(int i)  {
    cout << i << " ";
}

list<int> l{1, 3, 5};
for_each(begin(l), end(l), print);

 // Тот же код можно написать с использованием лямбда выражений

 for_each(begin(l), end(l), [](int i){cout << i << " ";});
```

find()

Find value in range Найти значение в диапазоне

```
template<typename InIt, typename T>
InIt find(InIt first, InIt last, const T &value);
```

Возможная реализация

```
template<typename InIt, typename T>
InIt find(InIt b, InIt e, T t) {
   while(b != e)
      if(*b == t)
         return b;
      ++b;
   return b;
}
```

Использование

```
int a[]{3, 1, 5, 7};
auto it = find(begin(a), end(a), 5);
if (it!= end(a))
   cout << "нашли" << *it;
else
   cout << "нет";
```

Использование

```
list<int> l;
…
list<int>::iterator p=find(begin(l), end(l), 42);
auto p = find(begin(l), end(l), 42);
```

find_if()

Find element in range Найти элемент в диапазоне

```
template<typename InIt, typename Pred>
InIt find_if(InIt first, InIt last, Pred pred);
```

Возможная реализация

```
template<typename InIt, typename
Pred>
InIt find_if(InIt b, InIt e, Pred p) {
   while(b != e)
      if(p(*b))
         return b;
      ++b;
   return b;
}
```

find_if()

Использование

```
int a[]{3, 1, 5, 7}
auto it = find_if(begin(a), end(a), [](int x)->bool{return x%2 == 0})
if (if it!= end(a))
   cout << "нашли" << *it;
else
   cout << "Нет";
```

Использование

```
list<int> l;
bool less_than_7(int v) { return v<7; }
list<int>::iterator p = find_if(begin(l), end(l),less_than_7);
```

find_end()

Find last subsequence in range Найти последнюю подпоследовательность в диапазоне

```
template<typename FwdIt1, typename FwdIt2>
FwdIt1 find_end(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2);
```

Пример

```
int array[12]={7,3,3,7,6,8,0,3,7,6,5,5};
int subarr[3]={3,7,6};
int* found_it;
found_it=find_end(array, array+12, subarr, subarr+3);
```

Пример

если первая последовательность – Miss**ss**ippi,
а вторая – ss,
то find_end() возвращает итератор, указывающий на первую s во втором вхождении ss.

find_first_of()

Find element from set in range Найти элемент из набора в диапазоне

```
template<typename FwdIt1, typename FwdIt2>
FwdIt1 find_first_of(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2);
```

Пример. Поиск первой гласной
Первая последовательность synesthesia
Вторая последовательность aeiou

Пример

```
int x[]={1,3,4};
int y[]={0,2,3,4,5};

int* p = find_first_of(x,x+3,y,y+5);   // p=&x[1]

int* q = find_first_of(p+1,x+3,y,y+5);          // q=&x[2]
```

count() и count_if()

Count appearances of value in range **Подсчет появления значения в диапазоне**

```
template<typename InIt, typename Type>
iterator_traits::distance_type  count(InIt first, InIt last, const Type& value);
```

Return number of elements in range satisfying condition
Возвращает количество элементов в диапазоне, удовлетворяющих условию

```
template<typename InIt, typename Pred>
iterator_traits::distance_type   count_if(InIt first, InIt last, Pred pred);
```

Пример

```
int n = count(p,p+size,'e');
```

Пример

```
bool equals_a(char* v) { return *v='a'; }

int n=count_if(p,p+size,equals_a);
```

all_of

Test condition on all elements in range

all_of(b, e, pred);

Возвращает true если все элементы удовлетворяют заданному предикату

equal()

Test whether the elements in two ranges are equal

```
template<typename InIt1, typename InIt2>
bool equal(InIt1 first, InIt1 last, InIt2 first2);

bool equal(b, e, b1) {
   while(b != e)    {
      if (*b != *b1)
         return false;
      ++b;
      ++b1;
   }
   return true;
}
```

Пример

```
bool b = equal(begin(li), end(li), begin(vd));
```

search

Search range for subsequence

Ищет подпоследовательность в последовательности
search(b, e, b1, e1);

template<typename FwdIt1, typename FwdIt2>
FwdIt search(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2);

Пример
в слове Mississippi подпоследовательность iss встречается дважды,
и search() возвращает итератор, указывающий на начало первого вхождения

Пример
```
string str="How old are you?";
string s="old";
char* p=search(begin(str), end(str), begin(s), end(s));
```

Алгоритмы, модифицирующие последовательность

copy	Копирует последовательность, начиная с первого элемента
copy_if	Копирует определенные элементы диапазона
copy_backward	Копирует последовательность, начиная с последнего элемента
swap	Меняет местами два элемента
swap_ranges	Меняет местами элементы двух последовательностей
iter_swap	Меняет местами два элемента, на которые указывают итераторы
transform	Выполняет операцию над каждым элементом в последовательности
replace	Заменяет элементы с указанным значением
replace_if	Заменяет элементы при выполнении предиката
replace_copy	Копирует последовательность, заменяя элементы с указанным значением
replace_copy_if	Копирует последовательность, заменяя элементы при выполнении предиката
fill	Заменяет все элементы данным значением
fill_n	Заменяет первые n элементов указанным значением
generate	Заменяет все элементы результатом операции
generate_n	Заменяет первые n элементов результатом операции

Алгоритмы, модифицирующие последовательность

remove	Удаляет элементы с данным значением
remove_if	Удаляет элементы при выполнении предиката
remove_copy	Копирует последовательность, удаляя элементы с указанным значением
remove_copy_if	Копирует последовательность, удаляя элементы, удовлетворяющие предикату
unique	Удаляет равные соседние элементы
unique_copy	Копирует последовательность, удаляя равные соседние элементы
reverse	Меняет порядок следования элементов на обратный
reverse_copy	Копирует последовательность в обратном порядке
rotate	Перемещает элементы циклически
rotate_copy	Копирует элементы циклической последовательности
random_shuffle	Перемещает элементы согласно случайному равномерному распределению («тасует» последовательность)
partition	Перемещает вперёд элементы, удовлетворяющие предикату
stable_partition	Перемещает вперёд элементы, удовлетворяющие предикату, сохраняя их относительный порядок следования

copy()

Копирует последовательность, начиная с первого элемента

Возможная реализация

```
template<typename ItIn, typename ItOut>
void copy(ItIn b, ItIn e, ItOut b1) {
    while (b != e)  {
        *b1 = *b;
        ++b;
        ++b1;
    }
}
```

```
vector<int> v{1, 5, 3};
list<int> l;
copy(begin(v), end(v), begin(l));

// Итератор будет иметь нулевое значение, поэтому  при попытке обратиться к нему произойдет ошибка
```

copy_if

Copy certain elements of range

Возможная реализация

```
template<class InputIt, class OutputIt, class UnaryPredicate>
OutputIt copy_if(InputIt first, InputIt last, OutputIt d_first, UnaryPredicate pred) {
    while (first != last) {
        if (pred(*first))
            *d_first++ = *first;
        first++;
    }
    return d_first;
}
```

Использование

```
std::copy_if(to_vector.begin(), to_vector.end(), std::ostream_iterator<int>(std::cout, " "),
                                                 [](int x) { return (x % 2) != 0; });
```

reverse_copy

Копирует последовательность в обратном порядке

Возможная реализация

```
template<class BidirIt, class OutputIt>
OutputIt reverse_copy(BidirIt first, BidirIt last, OutputIt d_first) {
   while (first != last) {
      *(d_first++) = *(--last);
   }
   return d_first;
```

Использование

```
std::vector<int> v({1,2,3});
print(v);
std::vector<int> destination(3);
std::reverse_copy(std::begin(v), std::end(v), std::begin(destination));
print(destination);
std::reverse_copy(std::rbegin(v), std::rend(v), std::begin(destination));
print(destination);
```

Output: 1 2 3 3 2 1 1 2 3

copy_backward

Копирует последовательность, начиная с последнего элемента

Возможная реализация

```
template< class BidirIt1, class BidirIt2 >
BidirIt2 copy_backward(BidirIt1 first, BidirIt1 last, BidirIt2 d_last) {
    while (first != last) {
        *(--d_last) = *(--last);
    }
    return d_last;
}
```

Использование

```
vector<int> from_vector;
    for (int i = 0; i < 10; i++) {
        from_vector.push_back(i);
    }
vector<int> to_vector(15);
copy_backward(from_vector.begin(), from_vector.end(), to_vector.end());
for (auto i: to_vector) { std::cout << i << " "; }
```

Output: 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9

Ещё один for_each

Нам нужен for_each, который будет умножать элементы контейнера на произвольное число

```
void twice (int & x) {
   x *= 2;
}

for_each(begin(v), end(v), twice)

// с помощью лямбд
for_each(begin(v), end(v), [](int& x){x *= 2;})
```

Захват переменных из внешнего контекста

int a = 5;
Изменим код for_each следующим образом:
for_each(begin(v), end(v), [](int& x){x *= a;}) //не компилируется

Чтобы код компилировался, необходимо указать в списке захвата, что переменная a пришла в функцию извне

int a = 5;
for_each(begin(v), end(v), **[a]**(int& x){x *= a;})

Так же в списке захвата можно использовать **[=]** — все переменные, которые встретятся в теле лямбда-функции, захватываются **по значению**.

for_each(begin(v), end(v), **[=]**(int& x){x *= a;})

Примечание. Лучше, если описание int a будет находится близко к вызову for_each

Типы возвращаемых значений

В простейших случаях тип возвращаемого значения лямбда-функции выводится компилятором:

```
void func4(std::vector<double>& v) {
    std::transform(begin(v), end(v), begin(v), [](double d) { return d < 0.00001 ? 0 : d; } );
}
```

Возвращаемый тип не может быть выведен компилятором:

```
void func4(std::vector<double>& v) {
    std::transform(v.begin(), v.end(), v.begin(), [](double d) {
                                                    if (d < 0.0001) { return 0; }
                                                    else { return d; }
});
}
```

какой тип будет в случае возвращения нуля - int или double?

Решение проблемы

Чтобы решить эту проблему, можно вручную указать возвращаемый тип лямбда-функции, используя синтаксис "-> тип":

```
void func4(std::vector<double>& v) {

    std::transform(v.begin(), v.end(), v.begin(), []**(double d) -> double** {
            if (d < 0.0001) {
                return 0;
            }
            else {
                 return d;
            }
    });
}
```

Захват переменных

Однако, лямбда-функция это не только анонимная функция, вызываемая в месте своего использования.

Помимо прочего, лямбда-функция может использовать переменные, доступные в ее контексте.

То есть переменные, доступные в том же контексте, где и описана лямбда-функции, доступны внутри лямбда-функции.

Это называется **замыкание**.

Цитата из Википедии:

Замыкание (англ. closure) в программировании — функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции и не в качестве её параметров (а в окружающем коде).

Говоря другим языком, замыкание — функция, которая ссылается на свободные переменные в своём контексте.

```
std::vector<double>v;
const double epsilon;

    std::transform(v.begin(), v.end(), v.begin(), [epsilon](double d) -> double {
            if (d < epsilon) { return 0;}
            else { return d;}
    });
```

```
void func5(std::vector<double>& v, const double& epsilon) {
    std::transform(v.begin(), v.end(), v.begin(), [epsilon](double d) -> double {
        if (d < epsilon) {
            return 0;
        }
        else {
            return d;
        }
}
```

Помимо всего прочего, можно захватывать не только переменную. Можно захватывать ссылку на переменную.

Помимо имен захватываемых переменных, можно указывать «правила захвата», обозначаемые «&» и «=»:

- [&epsilon] — захват ссылки на переменную;
- [&, epsilon] — показывает, что по умолчанию захват переменных будет происходить по ссылке но для переменной epsilon захват должен производиться по значению;
- [=, &epsilon] — показывает, что по умолчанию захват переменных будет происходить по значению, но для переменной epsilon захват должен производиться по ссылке;
- [x, &y] — захват переменной x по значению, и переменной y по ссылке.

Вопрос. Что захватывается?

[&] —

[&, =x] —

[=, &sum] —

Ответы...

[&] — все переменные в теле лямбда, не являющиеся параметрами, захватываются по ссылке

[&, =x] — все переменные в теле лямбда, кроме x, не являющиеся параметрами, захватываются по ссылке, x захватывается по значению

[=, &sum] — все переменные в теле лямбда, кроме sum, не являющиеся параметрами, захватываются по значению, sum захватывается по ссылке

Сумма элементов контейнера с помощью for_each

```
int sum = 0;

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

Объекты функции (функторы)

На самом деле в C++ лямбда выражения являются синтаксическим сахаром.
В действительности лямбда функции с захваченными переменными переводятся в объекты функций.

Объектом функции называется **объект класса с перегруженной операцией operator()**.

```
class SumHelper {
public:
    int sum;
    SumHelper(): sum(0) {}
    void operator()(int x) {
        sum += x;
    }
}
```

```
auto f = for_each(begin(v), end(v), SumHelper());
cout << f.sum;
```

На самом деле SumHelper() в for_each – это не функция, а экземпляр класса, у которого перегружен operator()

Присваивание лямбда функций переменной

```
#include <functional>

auto g = [](int x){return x*x;}

// Здесь auto принимает тип std::function<int(int)>, поэтому можно написать так:

std::function<int(int)> g = [](int x){return x*x;}

cout << g(5);
```

Опять к алгоритмам, модифицирующим последовательность

Копирующие

- copy
- transform
- replace
- ...

transform

```
template< class InputIt, class OutputIt, class UnaryOperation >
OutputIt transform( InputIt first1, InputIt last1, OutputIt d_first, UnaryOperation unary_op )

template< class InputIt1, class InputIt2, class OutputIt, class BinaryOperation >
OutputIt transform( InputIt1 first1, InputIt1 last1, InputIt2 first2, OutputIt d_first, BinaryOperation binary_op )
```

Применяет заданную функцию к одному диапазону и сохраняет результат в другой диапазон, начинающийся с d_first

В первом варианте унарная операция unary_op применяется к диапазону [first1, last1).
Во втором варианте бинарная операция binary_op применяется к элементам из двух диапазонов: [first1, last1) и начинающемуся с first2.

Параметры

first1, last1	—	первый диапазон элементов для изменения
first2	—	начало второго диапазона элементов для изменения
d_first	—	начало целевого диапазона, может совпадать с first1 или first2

Возможная реализация transform

Первый вариант

```
template<class InputIt, class OutputIt, class UnaryOperation>
OutputIt transform(InputIt first1, InputIt last1, OutputIt d_first, UnaryOperation unary_op) {
   while (first1 != last1) {
      *d_first++ = unary_op(*first1++);
   }
   return d_first;
}
```

Второй вариант

```
template<class InputIt1, class InputIt2, class OutputIt, class BinaryOperation>
OutputIt transform(InputIt first1, InputIt last1, InputIt first2, OutputIt d_first, BinaryOperation binary_op)
{
   while (first1 != last1) {
      *d_first++ = binary_op(*first1++, *first2++);
   }
   return d_first;
}
```

Пример

```
#include <string>
#include <algorithm>
#include <iostream>

int main() {
    std::string s("hello");
    std::transform(begin(s), end(s), s.begin(), std::toupper);
    std::cout << s;
}
```

Ещё примеры

```
transform(v.begin(), v.end(), v.begin(), [](int x){return x + 1;});

vector<int> v1{1, 2, 3};

int a[] {2, -2, 8};

list<int> l;

transform(begin(v1), end(v1), a, back_inserter(l),  std::max<int>)
```

std::remove, std::remove_if

```
template< class ForwardIt, class T >
ForwardIt remove( ForwardIt first, ForwardIt last, const T& value );                (until C++20)

template< class ForwardIt, class T >
constexpr ForwardIt remove( ForwardIt first, ForwardIt last, const T& value );      (since C++20)
```

```
template< class ForwardIt, class UnaryPredicate >
ForwardIt remove_if( ForwardIt first, ForwardIt last, UnaryPredicate p );           (until C++20)

template< class ForwardIt, class UnaryPredicate >
constexpr ForwardIt remove_if( ForwardIt first, ForwardIt last, UnaryPredicate p ); (since C++20)
```

Возможная реализация remove

```
template< class ForwardIt, class T >
ForwardIt remove(ForwardIt first, ForwardIt last, const T& value) {
    first = std::find(first, last, value);
    if (first != last)
        for(ForwardIt i = first; ++i != last; )
            if (!(*i == value))
                *first++ = std::move(*i);
    return first;
}
```

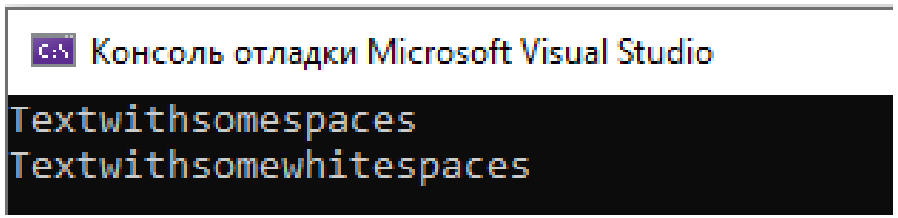
Возможная реализация remove_if

```
template<class ForwardIt, class UnaryPredicate>
ForwardIt remove_if(ForwardIt first, ForwardIt last, UnaryPredicate p) {
    first = std::find_if(first, last, p);
    if (first != last)
        for(ForwardIt i = first; ++i != last; )
            if (!p(*i))
                *first++ = std::move(*i);
    return first;
}
```

```
int main() {

    std::string str1 = "Text with some   spaces";
    str1.erase(std::remove(str1.begin(), str1.end(), ' '), str1.end());
    std::cout << str1 << '\n’;


    std::string str2 = "Text\n with\tsome \t  whitespaces\n\n";
    str2.erase(std::remove_if(str2.begin(), str2.end(), [](unsigned char x){return std::isspace(x);}), str2.end());
    std::cout << str2 << '\n';
}
```



constexpr

constexpr — спецификатор типа, введённый в стандарт программирования языка C++11 для обозначения константных выражений, которые могут быть вычислены во время компиляции кода.

Изначально стандартом языка C++11 на его применение был наложен ряд существенных условий и ограничений, однако большинство из них было снято после выхода стандарта C++14.

Разница между const и constexpr

Данная программа компилируется

```
#include <iostream>

struct A {
   constexpr static double x = 10.0;
};


int main() {
   A a;
   return 0;
}
```

А данная программа нет.

```
#include <iostream>

struct A {
   const static double x = 10.0;
};


int main() {
   A a;
   return 0;
}
```

Когда спецификатор constexpr используется для функций

Существенное значение также имеет место, когда этот спецификатор, constexpr, используется для функций.

Только функции-члены класса могут иметь квалификатор const, который имеет отношение к объекту, для которого вызывается данная функция-член класса.

Обычные функции не могут быть константными.

Спецификатор constexpr введен для того, чтобы заставить компилятор на этапе компиляции создавать объекты и использовать их как константы времени компиляции.