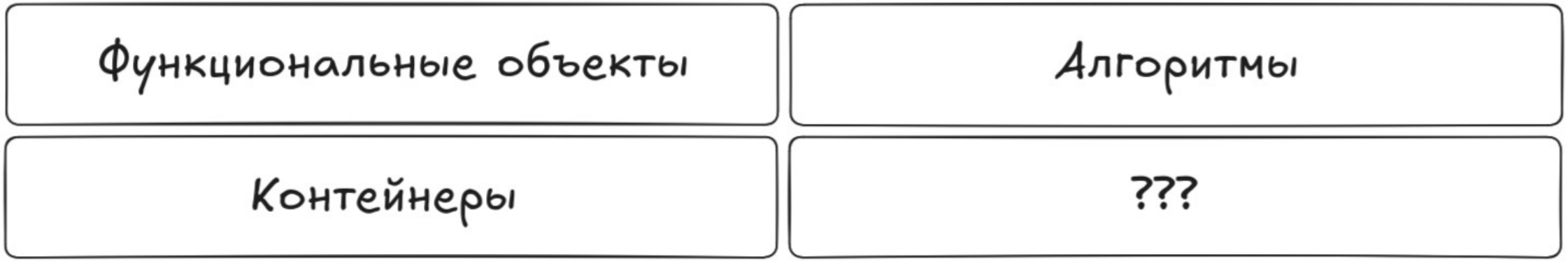
Прософт Системы

Контейнеры и итераторы в C++ (I)

СОДЕРЖАНИЕ

- Немного о STL
- Внутреннее устройство контейнеров
- Рекомендации по использованию контейнеров

STL КАК ПРИМЕР КЛАССНОГО ДИЗАЙНА

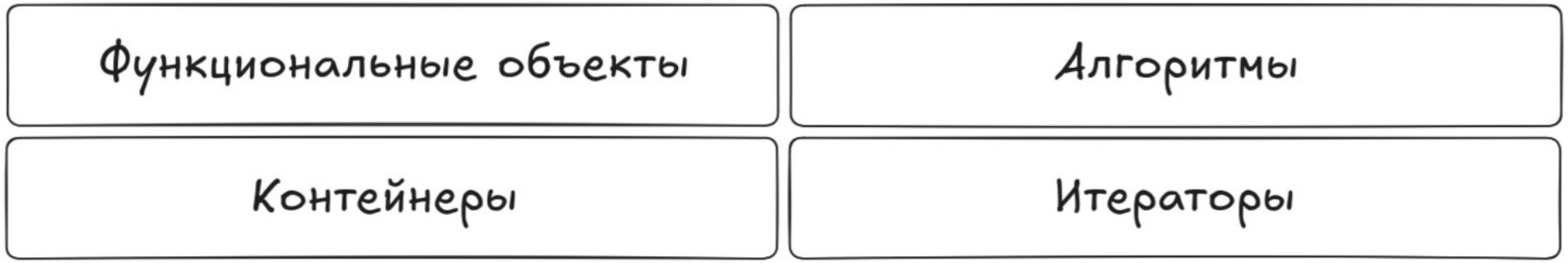


А ТЕПЕРЬ ЗАДАЧКА

Нужно написать функцию, которая будет искать элемент и возвращать его позицию в контейнере.
(только для контейнеров std::vector, std::array, std::set, std::list, std::forward_list, std::map)

Прософт Системы

ЩЕПОТКА БАЗЫ



ЩЕПОТКА БАЗЫ

- Для обобщения работы алгоритмов с контейнерами появились итераторы
- По простому итератор – это некоторый view на элемент контейнера
- Итераторы поддерживают различные арифметические операции
- У каждого итератора есть его категория, их пять штук (до C++20, сейчас 6)

ЩЕПОТКА БАЗЫ

- Разыменование (чтение) : *it
- Присваивание (запись): *it = value
- Инкремент/декремент: ++it/--it
- Доступ по []: container[]
- Доступ к элементу внутри итератора: it→Method()

ЩЕПОТКА БАЗЫ

А теперь давайте совместим приятное с полезным и будем выводить категорию итератора у соотвествующего контейнера STL

Прософт Системы

ЩЕПОТКА БАЗЫ

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
???							
???							
???							
???							
???							

СТАРЫЕ ГРАБЛИ (ПОТОКИ ВВОДА-ВЫВОДА)

Под капотом std::cin/std::cout скрыто очень много интересных вещей, давайте попробуем проитерироваться по потокам ввода и вывода через итератор

СТАРЫЕ ГРАБЛИ (ПОТОКИ ВВОДА-ВЫВОДА)

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
Input							
Output							
???							
???							
???							

СТАРЫЕ ГРАБЛИ (ПОТОКИ ВВОДА-ВЫВОДА)

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
Input	-	+	+	-	+	-	
Output	+	-	+	-	-	-	
???							
???							
???							

DIY ИЛИ ГОТОВИМ ИТЕРАТОР В ДОМАШНИХ УСЛОВИЯХ

В первой задаче на контесте вы отважно пилили простой класс строки, теперь давайте добавим к этой строке итератор

FORWARD ITERATOR

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
Input	-	+	+	-	+	-	
Output	+	-	+	-	-	-	
Forward							
???							
???							

FORWARD ITERATOR

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
Input	-	+	+	-	+	-	
Output	+	-	+	-	-	-	
Forward	+	+	+	-	+	-	
???							
???							

BIDIRECTIONAL ITERATOR

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
Input	-	+	+	-	+	-	
Output	+	-	+	-	-	-	
Forward	+	+	+	-	+	-	
Bidirectional							
???							

BIDIRECTIONAL ITERATOR

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
Input	-	+	+	-	+	-	
Output	+	-	+	-	-	-	
Forward	+	+	+	-	+	-	
Bidirectional	+	+	+	+	+	-	
???							

RANDOM ACCESS ITERATOR

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
Input	-	+	+	-	+	-	
Output	+	-	+	-	-	-	
Forward	+	+	+	-	+	-	
Bidirectional	+	+	+	+	+	-	
RandomAccess							

RANDOM ACCESS ITERATOR

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
Input	-	+	+	-	+	-	
Output	+	-	+	-	-	-	
Forward	+	+	+	-	+	-	
Bidirectional	+	+	+	+	+	-	
RandomAccess	+	+	+	+	+	+	

RANDOM ACCESS ITERATOR

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	???
Input	-	+	+	-	+	-	
Output	+	-	+	-	-	-	
Forward	+	+	+	-	+	-	
Bidirectional	+	+	+	+	+	-	
RandomAccess	+	+	+	+	+	+	

CONTIGUOUS ITERATOR (C++ 20)

Итератор	Запись	Чтение	Инкремент	Декремент	Доступ к элементу	Доступ по []	Непрерывное расположение элементов в памяти
Input	-	+	+	-	+	-	-
Output	+	-	+	-	-	-	-
Forward	+	+	+	-	+	-	-
Bidirectional	+	+	+	+	+	-	-
RandomAccess	+	+	+	+	+	+	-
Contiguous	+	+	+	+	+	+	+

TRAITS

В STL есть очень мощный механизм метапрограммирования различных классов под названием Traits

ITERATOR TRAITS

Traits (характеристики) - это шаблонные структуры, которые предоставляют информацию о свойствах типов. Они служат для:

- Получения информации о типах
- Предоставления единого интерфейса для работы с разными типами
- Реализации специализаций для особых случаев

ITERATOR TRAITS


```
template<class Iterator>
struct iterator_traits {
    using difference_type   = typename Iterator::difference_type;
    using value_type        = typename Iterator::value_type;
    using pointer           = typename Iterator::pointer;
    using reference         = typename Iterator::reference;
    using iterator_category = typename Iterator::iterator_category;
};
```