# Лекция по курсу «Алгоритмы и структуры данных» / «Технологии и методы программирования»

## СТАНДАРТНАЯ БИБЛИОТЕКА C++:

Упорядоченные ассоциативные контейнеры:
ассоциативные массивы (map, multimap), множества (set, multiset)
Неупорядоченные ассоциативные контейнеры:
н/у ассоциативные массивы (unordered_map, unordered_ multimap),
н/у множества (unordered_ set, unordered_ multiset);

Мясников Е.В.

# Упорядоченные ассоциативные контейнеры

# Упорядоченные ассоциативные контейнеры

**map<K,V,C,A>** ассоциативный массив (отображение) – последовательность пар <ключ-значение> (**K**,**V**)

**multimap<K,V,C,A>** ассоциативный массив с дублирующимися ключами (кратность 2 и выше, мультиотображение)

**set<K,C,A>** множество ключей K

**multiset<K,C,A>** множество с дублирующимися ключами (кратность 2 и выше)

Обычно реализуются как ***сбалансированные деревья поиска*** (красно-черные)

Параметры:

**K, V** – типы ключей и значений

**C** – тип сравнения

**A** – аллокатор, определяющий способ размещения и освобождения памяти

По-умолчанию **C** = **std::less<K>**

Аллокатор **A = std::allocator<std::pair<const K,T>>** или **A = std::allocator<K> :**

выделение - **operator new()**

освобождение - **operator delete()**

# Ассоциативный массив (map) - типы

**Внутренние типы:**

```
template<class Key, class T, class Compare, class Alloc >
class map {
  public:
  using iterator =  …
  using const_iterator = …;
  using reverse_iterator = …;
  using const_reverse_iterator = ...;
  using key_type = Key;          // Тип ключа
  using mapped_type = T;         // Тип значения
  using value_type = std::pair<const Key, T>;       // Тип эл-та
  using key_compare = Compare;                      // Тип опер. сранения
  using reference = …;           // Тип ссылки на эл-т
  using pointer = …;             // Тип указателя на эл-т
  using const_reference = …;     // Тип константной ссылки на эл-т
  using const_pointer = …;       // Тип константного ук-ля на эл-т
  using difference_type = …;     // Тип результата вычитания итераторов
  using size_type = …;           // Тип размера
  using allocator_type = Allocator;       // Тип аллокатора
  // ...
```

Типы итераторов

# Ассоциативный массив (map) - создание

**Создание отображения:**

```
map();                                                  // создание пустого отображения

template< class InputIt >
map( InputIt first, InputIt last,                       // … с содержимым в диапазоне
    const Compare& comp = Compare(),
    const Allocator& alloc = Allocator() );

map( const map& other );                                // к. копирования

map( map&& other );                                     // к. перемещения

map( std::initializer_list<value_type> init,            // … с использованием
    const Compare& comp = Compare(),                    // содержимого списка
    const Allocator& alloc = Allocator() );             // инициализатора
```

# Ассоциативный массив (отображение, map)

**Вставка элементов:**

```
std::pair<iterator, bool> insert(                  // вставка элемента (пары)
   const value_type& value );
…
iterator insert( iterator hint,                    // вставка элемента с подсказкой
                 const value_type& value );        // (где искать или перед чем вставлять)
…
template< class InputIt >                          // вставка элементов с содержимым в
void insert( InputIt first, InputIt last );        // диапазоне

void insert(                                       // вставка элементов списка
   std::initializer_list<value_type> ilist );      // инициализатора

template< class... Args >
std::pair<iterator,bool> emplace(                  // вставка элемента с конструиро-
   Args&&... args );                               //    ванием его на месте по args
template <class... Args>
iterator emplace_hint( const_iterator hint,        // … с подсказкой
Args&&... args );
```

# Ассоциативный массив (отображение, map)

**Доступ и поиск :**

```
T& at( const Key& key );                    // ссылка на значение по ключу
const T& at( const Key& key ) const;        // если нет, то искл. std::out_of_range

T& operator[]( const Key& key );            // ссылка на значение по ключу
…                                           // если нет, то ВСТАВКА value_type(key, T())

iterator find( const Key& key );            // поиск элемента по ключу
…                                           // если нет, то возвращается end()
size_type count( const Key& key ) const;    // количество элементов с ключом
…                                           // (0 или 1)

std::pair<iterator,iterator> equal_range(   // диапазон элементов с заданным
   const Key& key );                        // ключом (если нет, то end()) )
…
iterator lower_bound( const Key& key );     // итератор на первый элемент
…                                           // больший key
iterator upper_bound( const Key& key );     // итератор на первый элемент
…                                           // не меньший key
```

# Ассоциативный массив (отображение, map)

**Очистка и удаление:**

```
void clear();                                   // удаление всех элементов
…

void erase( iterator pos );                     // удаление элемента на позиции pos
iterator erase( iterator pos );                 // …возвр. итер. за последним удаленным
…
void erase( iterator first, iterator last );    // удаление элементов в диапазоне
…
size_type erase( const key_type& key );         // удаление элемента по ключу
                                                // (возвр. количество)
```

**Другое:**

```
void swap( map& other );                        //обмен содержимого с другим map

bool empty() const;                             // проверка пустоты контейнера

size_type size() const;                         // количество элементов
size_type max_size() const;                     // … максимально допустимое
```

# Ассоциативный массив (отображение, map)

**Пример:**

```
std::map<std::string, int> m1;          // создание пустого отображения
m1[“Ivanov"] = 4;
m1[“Petrov"] = 5;                       // вставка элементов
m1[“Sidorov"] = 3;

// создание  с заполнением
std::map<std::string, int> m2 { {"Ivanov", 4}, {"Petrov",5}, {"Sidorov", 3}, };

m2[“Sidorov"] = 5;                      // обновление по существующему ключу
m2.insert(std::pair<std::string, int>(“Vasechkin", 3));  // вставка элемента – пары

auto s = m2.find("Sidorov");            // поиск
if (s != m2.end())                      // проверка нахождения
      m2.erase(s);                      // удаление
else  std::cout << “Sidorov not found!";

for (auto p : m2)                       // вывод содержимого
      std::cout << p.first << " - " << p.second << “\n ";
```

# Упорядоченные ассоциативные контейнеры

**map<K,V,C,A>** ассоциативный массив (отображение) – последовательность пар <ключ-значение> (**K**,**V**)

**multimap<K,V,C,A>** ассоциативный массив с дублирующимися ключами (кратность 2 и выше, мультиотображение)

Multimap похож на map с той лишь разницей, что несколько элементов могут иметь одинаковые ключи. Не требуется также, чтобы пара ключ-значение была уникальной. multimap всегда хранит ключи в отсортированном порядке.

**set<K,C,A>** множество (набор) ключей K.

тип ассоциативных контейнеров, в которых каждый элемент должен быть уникальным.

Значение элемента не может быть изменено после его добавления в набор, хотя можно удалить и добавить измененное значение этого элемента.

**multiset<K,C,A>** множество (набор) с дублирующимися ключами (кратность 2 и выше)

# Упорядоченные ассоциативные контейнеры: примеры

***set***

```
std::set<int> s;
s.insert({ 4,7,3,2,7,8,7,9 });

std::cout << "\n set s:  ";
for (auto n : s)
   std::cout << n << " - ";

s.erase(s.find(3), s.find(8));


std::cout << "\n set s:  ";
for (auto n : s)
   std::cout << n << " - ";
```

**вывод:**

```
set s:  2 - 3 - 4 - 7 - 8 - 9 –
set s:  2 - 8 - 9 –
```

***multiset***

```
std::multiset<int> s;
s.insert({ 4,7,3,2,7,8,7,9 });

std::cout << "\n set s:  ";
for (auto n : s)
   std::cout << n << " - ";

s.erase( s.lower_bound(7),
         s.upper_bound(7) );

std::cout << "\n set s:  ";
for (auto n : s)
   std::cout << n << " - ";
```

**вывод:**

```
multiset s:  2 - 3 - 4 - 7 - 7 - 7 - 8 - 9 -
multiset s:  2 - 3 - 4 - 8 - 9 -
```

# Неупорядоченные ассоциативные контейнеры

# Неупорядоченные ассоциативные контейнеры

**unordered_map<K,V,H,E,A>** неупорядоченный ассоциативный массив пар <ключ-значение> (**K**,**V**)
**unordered_multimap<K,V,H,E,A>** неупорядоченный ассоциативный массив с дублирующимися ключами
**unordered_set<K,H,E,A>** неупорядоченное множество элементов типа **K**
**unordered_multiset<K,H,E,A>** неупорядоченное множество с дубликатами

Обычно реализуются как хеш-таблицы (с разрешением коллизий по методу цепочек)

Параметры:
- **K, V** – типы ключей и значений
- **H** – хеш-функция
- **E** – функция сравнения на равенство
- **A** – аллокатор, определяющий способ размещения и освобождения памяти

По-умолчанию
аллокатор **A = std::allocator<std::pair<const K,T>>** или **A = std::allocator<K> :**
- выделение - **operator new(),** освобождение - **operator delete()**

хеш-функция по умолчанию: **H = K** is **std::hash<K>**
функция сравнения по-умолчанию: **H =** type **K** is **std::equal_to<K>**

# Н/у ассоциативный массив (**unordered_map**) - типы

**Внутренние типы:**

```
template<class Key, class T, class Hash, class KeyEqual, class Alloc >
class map {
  public:
  using iterator =  …
  using const_iterator = …;
  using local_iterator = …;
  using const_local_iterator = ...;
  using key_type = Key;          // Тип ключа
  using mapped_type = T;         // Тип значения
  using value_type = std::pair<const Key, T>;       // Тип эл-та
  using hasher = Hash;                    // Тип опер. сранения
  using key_equal = KeyEqual;             // Тип опер. сранения
  using reference = …;           // Тип ссылки на эл-т
  using pointer = …;             // Тип указателя на эл-т
  using const_reference = …;     // Тип константной ссылки на эл-т
  using const_pointer = …;       // Тип константного ук-ля на эл-т
  using difference_type = …;     // Тип результата вычитания итераторов
  using size_type = …;           // Тип размера
  using allocator_type = Allocator;       // Тип аллокатора
  // ...
```

Итераторы по корзине

Типы итераторов

# Н/у ассоциативный массив (**unordered_map**) - создание

**Создание отображения unordered_map**

```
unordered_map();                                    // создание пустого контейнера
                                                    // с числом корзин по-умолчанию

unordered_map(                                      // с заданным числом корзин
        size_type bucket_count,                     // и др. параметрами
        const Hash& hash = Hash(),
        const key_equal& equal = key_equal(),
        const Allocator& alloc = Allocator() );

template< class InputIt >
unordered_map( InputIt first, InputIt last,         // … с содержимым в диапазоне
        size_type bucket_count  /*…*/ )

unordered_map( const unordered_map& other );  // к. копирования
unordered_map( unordered_map&& other );       // к. перемещения

unordered_map(                                      // … с использованием
        std::initializer_list<value_type> init,     // инициализатора
      size_type bucket_count  /*…*/ );              // содержимого
```

# Н/у ассоциативный массив (**unordered_map**)

**Вставка элементов:**

```
std::pair<iterator, bool> insert(                  // вставка элемента (пары)
   const value_type& value );
…
iterator insert(const_iterator  hint,              // вставка элемента с подсказкой
                const value_type& value );         // (где искать или перед чем вставлять)
…
template< class InputIt >                          // вставка элементов с содержимым в
void insert( InputIt first, InputIt last );        // диапазоне

void insert(                                       // вставка элементов списка
   std::initializer_list<value_type> ilist );      // инициализатора

template< class... Args >
std::pair<iterator,bool> emplace(                  // вставка элемента с конструиро-
   Args&&... args );                               //    ванием его на месте по args
template <class... Args>
iterator emplace_hint( const_iterator hint,        // … с подсказкой
Args&&... args );
```

# Н/у ассоциативный массив (**unordered_map**)

**Доступ и поиск :**

```
T& at( const Key& key );                          // ссылка на значение по ключу
const T& at( const Key& key ) const;              // если нет, то искл. std::out_of_range

T& operator[]( const Key& key );                  // ссылка на значение по ключу
…                                                 // если нет, то ВСТАВКА value_type(key, T())

iterator find( const Key& key );                  // поиск элемента по ключу
…                                                 // если нет, то возвращается end()
size_type count( const Key& key ) const;          // количество элементов с ключом
…                                                 // (0 или 1)

std::pair<iterator,iterator> equal_range(         // диапазон элементов с заданным
   const Key& key );                              // ключом (если нет, то end()) )
…
iterator lower_bound( const Key& key );           // итератор на первый элемент
…                                                 // больший key
iterator upper_bound( const Key& key );           // итератор на первый элемент
…                                                 // не меньший key
```

# Н/у ассоциативный массив (**unordered_map**)

**Очистка и удаление:**

```
void clear();                                   // удаление всех элементов
…

iterator erase(const_iterator pos );            // удаление элемента на позиции pos
…                                               // возвр. итер. за последним удаленным

void erase(const_iterator first,                // удаление элементов в диапазоне
           const_iterator last );
…
size_type erase( const key_type& key );         // удаление элемента по ключу
                                                // (возвр. количество  0/1)
```

**Другое:**

```
void swap(unordered_map& other );               //обмен содержимого с другим map

bool empty() const;                             // проверка пустоты контейнера

size_type size() const;                         // количество элементов
size_type max_size() const;                     // … максимально допустимое
```

# Н/у ассоциативный массив (**unordered_map**)

**Хэширование:**

```
float load_factor() const;                  // коэф. заполнения (эл-тов на корзину)

float max_load_factor() const;              // возвращает или устанавливает
void max_load_factor( float ml );           // максимальный коэф. заполнения

void rehash( size_type count );             // меняет кол-во корзин и
                                            // перераспределяет элементы

void reserve( size_type count );            // rehash(ceil(count / max_load_factor()))
```

**Работа с корзинами:**

```
size_type bucket_count() const;             // кол-во корзин
size_type max_bucket_count() const;         // …максимально возможное (при орган.)

size_type bucket( const Key& key ) const;   // индекс корзины по ключу

size_type bucket_size( size_type n ) const; // размер корзины с индексом n

local_iterator begin( size_type n );        // итератор на первый / последний
… /* cbegin, end, cend */                   // элемент корзины с индексом n
```

# Н/у ассоциативный массив (**unordered_map**)

**Пример:**

```
unordered_map<int, string> m(8);        // создание пустого контейнера
                                        // с 8 корзинами

// вставка 5 элементов (пар)
m.insert({ {1, "ivanov"},{2, "petrov"},{3, "sidorov"}, });
m.insert(pair<int, string>(4, "vasechkin"));
m.emplace(9, "maleev");

// по всем корзинам
for (size_t b = 0; b < m.bucket_count(); b++) {
        // вывод номера корзины
        cout << endl << b << ":  ";
        // по всем элементам корзины
        for (auto i = m.begin(b); i != m.end(b); i++)
                // вывод пары
                cout << "(" << i->first << "; " << i->second << ")  ";
}
```

```
0:
1:  (4; vasechkin)
2:
3:
4:  (9; maleev)  (1; ivanov)
5:
6:  (3; sidorov)
7:  (2; petrov)
```

# Неупорядоченные ассоциативные контейнеры

**unordered_map<K,V,H,E,A>** неупорядоченный ассоциативный массив (отображение) – последовательность пар <ключ-значение> (**K**,**V**)

**unordered_multimap<K,V, H,E,A>** неупорядоченный ассоциативный массив с дублирующимися ключами (кратность 2 и выше, мультиотображение)
Multimap похож на map с той лишь разницей, что несколько элементов могут иметь одинаковые ключи. Не требуется также, чтобы пара ключ-значение была уникальной.

**unordered_set<K, H,E,A>** неупорядоченное множество (набор) ключей K.
тип ассоциативных контейнеров, в которых каждый элемент должен быть уникальным.
Значение элемента не может быть изменено после его добавления в набор, хотя можно удалить и добавить измененное значение этого элемента.

**unordered_multiset<K, H,E,A>** неупорядоченное множество (набор) с дублирующимися ключами (кратность 2 и выше)