Map — конспект темы

Всё могут словари — контейнер map

`map` (словарь) — контейнер, хранящий пары «ключ — значение» и упрощающий поиск значения по ключу.

Чтобы использовать словарь, нужно подключить заголовочный файл `map`. При объявлении словаря требуется указать сразу два типа: тип ключей и тип значений:

```
#include <map>
#include <string>

using namespace std;

int main() {
    // Создаём пустой словарь:
    // ключи - строки, значения - числа.
    map<string, int> legs_count;
}
```

Есть два способа наполнить словарь:

1. При инициализации переменной.

```
#include <map>
#include <string>

using namespace std;

int main() {
    map<string, int> legs_count = {{"rabbit"s, 4}, {"dog"s, 4}, {"chicken"s, 2},
                                   {"spider"s, 8}, {"fly"s, 6}};
}
```

2. Вставками.

`map` хранит в каждом элементе сразу два значения, а значит, для итерирования нужно две переменных. Их можно объявить декомпозированием:

```
void PrintMap(const map<string, int>& m) {
    cout << "Size: "s << m.size() << endl;
```

```
    for (const auto& [key, value]: m) {
        cout << key << " - "s << value << endl;
    }
}
```

- `auto` — ключевое слово. Оно просит компилятор самостоятельно вывести тип переменной.
- `&` указывает, что мы хотим использовать ссылки вместо копирования элементов.
- `const` указывает, что мы не собираемся менять элементы.
- `[` и `]` нужны для перечисления переменных. Для итерирования по словарю используются две переменные. Одна — для ключей, другая — для значений. Они последовательно пройдут все пары ключ-значение, хранящиеся в словаре.

Элементы `map` отсортированы, как и в `set`. Но при сортировке учитывается только ключ элемента.

Чтобы обратиться к элементу словаря, используются квадратные скобки.

Есть два способа добавить в словарь новую запись.

1. Через метод `insert`:

```
// Так как добавляем пару, её нужно заключить в фигурные скобки:
legs_count.insert({"cat"s, 4});
```

2. Через квадратные скобки:

```
legs_count["elephant"s] = 4;
PrintMap(legs_count);
```

Метод `erase` позволяет удалить элементы. В него передаётся ключ удаляемого элемента. Значение указывать не нужно, так как элемент `map` однозначно идентифицируется ключом:

```
map<string, int> legs_count = {{"rabbit"s, 4}, {"dog"s, 4}, {"ostrich"s, 2},
                               {"car"s, 4}};
cout << "We know about "s << legs_count.size() << " animals"s << endl;

// Удаляем автомобиль:
legs_count.erase("car"s);
cout << "We know about "s << legs_count.size() << " animals"s << endl;
```

Подводные камни map

Метод `count` позволяет проверить наличие ключа:

```
if (legs_count.count(animal_name) == 1) {
    cout << "Yes, we know this animal!"s << endl;
} else {
    cout << "Sorry, no info about this species!"s << endl;
}
```

Узнать количество элементов можно методом `size`, а проверить на пустоту — методом `empty`.

Каждый раз, когда вы запрашиваете значение по несуществующему ключу, операция `[]` меняет словарь: добавляет ключ со стандартным значением.

Можно использовать цикл `range-based for`, чтобы менять словарь, по которому он проходит:

```
map<string, int> days_here = {
    {"Karl"s, 10}, {"Gustav"s, 3}, {"Richard"s, 42}, {"Wolfgang"s, 15} };

// если мы хотим поменять значение, то const писать не нужно
for (auto& [name, days] : days_here) {
    ++days;
}

cout << "Karl has been here for "s << days_here["Karl"s] << " days"s << endl;
```

Программа выведет текст:

```
Karl has been here for 11 days
```

Ключи словаря всегда константны. Их изменение влекло бы нарушение работы словаря.

Код выше можно сделать более эффективным, если заменить присваивание на обмен функцией `swap`. Присваивание уничтожает значение первой переменной, а затем копирует. Функция `swap` просто меняет два словаря местами — это быстрее:

```
swap(days_here, days_here_new);
```

Присваивание уничтожит содержимое `days_here`, а затем скопирует словарь. Функция `swap` меняет два словаря местами. Это быстрая операция.

Словари и константность

Операция `[]` позволяет читать из словаря и менять в нём значения. Но она может приводить к изменению — если ключа нет, он автоматически добавляется в словарь.

Чтобы словарь не обновлялся при запросе неизвестного ключа, применяют **метод** `at()`:

```
const map<string, int> legs_count = {{"rabbit"s, 4}, {"dog"s, 4}, {"chicken"s, 2}};
int value = legs_count.at("dog"s); // как и для других методов круглые скобки
```

Если обратиться по несуществующему ключу, возникнет специальная ошибка — **исключение**.

Код с `at()` скомпилируется:

```
  const map<string, int> birds_amount = {{"chicken"s, 2}, {"duck"s, 3}, {"goose"s, 2}};
  cout << birds_amount.at("bat"s) << endl;
```

Но во время исполнения во второй строке появится исключение:

```
terminate called after throwing an instance of 'std::out_of_range'
  what():  map::at
Aborted (core dumped)
```

Команда `terminate` вызывается, чтобы завершить работу программы, если исключение не было обработано.

```
| `[]`           | есть ключ или индекс | нет ключа или индекса                          |
|----------------|----------------------|------------------------------------------------|
| `vector`       | Ok                   | неопределённое поведение                       |
| `const vector` | Ok                   | неопределённое поведение                       |
| `map`          | Ok                   | добавляется элемент со значением по умолчанию |
| `const map`    | ошибка компиляции    | ошибка компиляции                              |
```

Контейнер pair

В реализации контейнера `map` есть контейнер `pair`, состоящий из пары значений разного типа.

Тип `entry` — пара из строки и целого числа `pair<string, int>`:

```
map<string, int> legs_count = {{"dog"s, 4}, {"ostrich"s, 2}};

// Аналогично for (const pair<string, int>& entry : legs_count)
for (const auto& entry : legs_count) {
    // здесь надо обратиться к содержимому
}
```

При создании словаря пара была записана внутри фигурных скобок. `.first` — это **ключ**, а `.second` — **значение**:

```
map<string, int> legs_count = {{"dog"s, 4}, {"ostrich"s, 2}};
for (const auto& entry : legs_count) {
    cout << entry.first << " : "s << entry.second << endl;
}
```

`auto` позволяет декомпозировать содержимое пары — связать имена в программе с полями `first` и `second`:

```
map<string, int> legs_count = {{"dog"s, 4}, {"ostrich"s, 2}};

// Имена animal и leg_count ссылаются на ключ и значение, соответственно:
for (const auto& [animal, leg_count] : legs_count) {
    cout << animal << " : "s << leg_count << endl;
}
```

На выходе:

```
dog : 4
ostrich : 2
```

Пару из двух значений можно создать на лету фигурными скобками, а затем вставить в словарь методом `insert()` как ключ и значение:

```
const auto [octopus_iterator, success] = legs_count.insert({"octopus"s, 8});
```

Метод `insert()` не только принимает, но и возвращает пару. Она состоит из:

- итератора `octopus_iterator`, который указывает на добавленный элемент,
- булева значения `success`. Оно равно `true`, если вставка удалась, то есть такой записи в словаре ещё не было. В противном случае значение равно `false`.

`pair` может работать как вместе с `map`, так и самостоятельно.