

**Занятие 16.12.17**

# Основные пункты

- Вычислительная сложность
- Базовые структуры данных и их использование в C++

# Вычислительная сложность

- Нужно как-то сравнивать ресурсы, которые будут потрачены тем или иным алгоритмом
- Они включают:
  - Время процессора
  - Занимаемая память

# Вычислительная сложность

- Вариант #1 – точный подсчет, например, отдельных команд процессора
- Проблемы:
  - Команды занимают разное время
  - У разных процессоров разные наборы команд
  - Громоздкость выражений и сложность подсчета

# Вычислительная сложность

- Как правило, достаточно сравнивать поведение алгоритмов в целом
- Вариант #2 – сравнивать общий вид зависимости использованного времени/памяти от входных данных

# «O» большое

- Для таких оценок используется нотация «O» большое.
- Формально:

$$f(n) = O(g(n))$$

тогда и только тогда, когда

$$\exists M: |f(n)| \leq M|g(n)| \quad \forall n$$

# «O» большое

- Объяснение на примерах:
- мы говорим, что алгоритм имеет сложность  $O(n)$  операций, если с ростом размера входных данных затрачиваемое время/ресурсы растут линейно

# «O» большое

- $O(n)$ :
  - $n = 10$ , операций 10
  - $n = 20$ , операций 20
- Но так же под  $O(n)$  подходит:
  - $n = 1$ , операций 103
  - $n = 2$ , операций 206
  - $n = 1000$ , операций 112 910

**главное – что в целом растет линейно**



# «O» большое

- Мы говорим, что алгоритм имеет сложность  $O(n^2)$  операций, если с ростом размера входных данных затрачиваемое время/ресурсы растут квадратично.

$n = 10$ , операций около 100

$n = 20$ , операций около 400

# «O» большое

- Часто можно увидеть:
  - $O(1)$
  - $O(\log n)$
  - $O(\sqrt{n})$
  - $O(n)$
  - $O(n * \log n)$
  - $O(n^2)$
  - $O(2^n)$

# «O» большое

Что такое  $n$ ? Примеры:

- Определение простоты числа  $n$ :  $n$  – само число
- Сортировка массива:  $n$  – размер массива
- Работа со строкой:  $n$  – размер строки

# Базовые структуры данных

- Массив
- Вектор
- Множество
- Стек
- Очередь
- Словарь

# Массив

- Последовательность элементов фиксированного размера.
- Операции:
  - Получить элемент по индексу:  $O(1)$  времени
  - Записать элемент по индексу:  $O(1)$  времени

# Массив в C++

```
int arr[100];  
arr[0] = 123; // записали элемент  
cout << arr[0]; // получили элемент
```

---

```
int *arr = new int[100];  
arr[0] = 123;  
  
delete[] arr;
```

# STL

- STL (Standard Template Library) – набор алгоритмов, контейнеров и вспомогательных функций в языке C++.
- Контейнеры – объекты для хранения данных. В виде них в C++ уже реализованы все базовые структуры.
- Далее – общий обзор основных из этих структур.

# Вектор

- Массив имеет фиксированный размер, что не всегда удобно в практических ситуациях.
- Операции:
  - Получить элемент по индексу:  $O(1)$  времени
  - Записать элемент по индексу:  $O(1)$
  - Получить текущий размер вектора:  $O(1)$
  - Добавить элемент в конец вектора:  $O(1)^*$



# Вектор в C++

```
#include <vector>
```

```
vector<int> tmp;
```

```
tmp.push_back(1);
```

```
tmp.push_back(2);
```

```
tmp.push_back(3);
```

```
cout << tmp [0] << tmp[1] << tmp[2]; // 1 2 3
```

В угловых скобках <>  
указан

тип данных, которые  
будут

храниться в векторе

Индексация с 0, обращение как и к  
массиву

# Вектор в C++

```
vector<int> numbers;
```

```
cin >> n;
```

```
for (int i = 0; i < n; i++)
```

```
{
```

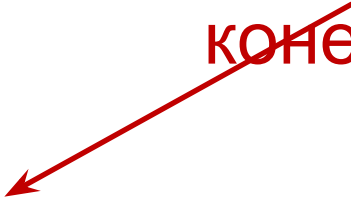
```
    int tmp;
```

```
    cin >> tmp;
```

```
    numbers.push_back(tmp);
```

```
}
```


Добавляем элемент в  
конец



# Вектор в C++

```
vector<int> numbers;  
// ...
```

Количество  
элементов  
на данный момент



```
for (int i = 0; i < number.size(); i++)  
{  
    cout << numbers[i] << endl;  
}
```

# Множество

- Структура данных, в которой каждый элемент хранится в единственном числе.
- Операции:
  - Добавить элемент в множество
  - Удалить элемент из множества
  - Проверить наличие элемента во множестве
  - Получить размер множества

# Множество в C++

- В C++ существует две реализации множества – `set` и `unordered_set`. Пока остановимся только на первой.
- Занимаемая память –  $O(n \log n)$
- Добавление элемента –  $O(\log n)$
- Удаление элемента –  $O(\log n)$
- Проверка элемента –  $O(\log n)$

# Множество в C++

```
#include <set>
```

```
set<int> my_set;
```


```
my_set.insert(3);
```

```
my_set.insert(4);
```

```
my_set.insert(3);
```

```
cout << my_set.size(); // 2
```

В множестве два  
элемента:  
числа 3 и 4



# Множество в C++

```
set<int> my_set;  
my_set.insert(1);  
my_set.insert(2);  
my_set.erase(1);  
my_set.erase(2);  
  
cout << my_set.size(); // 0
```

Удалили все  
элементы

# Множество в C++

```
set<int> my_set;
```

```
// Определим, есть ли там элемент 2
```

```
if (my_set.find(2) != my_set.end())
```

```
{
```

```
    cout << "Element 2 exists!";
```

```
}
```



# Множество в C++

```
set<int> my_set;  
// Вывести все элементы множества
```

```
for (auto it = my_set.begin();  
     it != my_set.end();  
     it++)  
{  
    cout << *it << endl;  
}
```



Это  
итераторы,  
их разберем  
не сегодня

# Стек

- Структура данных «LIFO» (Last-In First-Out).
- Операции:
  - Добавить элемент на вершину стека:  $O(1)$
  - Получить элемент с вершины стека:  $O(1)$
  - Удалить элемент с вершины стека:  $O(1)$
  - Определить, не пустой ли стек:  $O(1)$

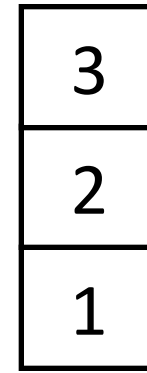
# Стек

- Названия операций:
  - Добавить элемент: push
  - Получить элемент на вершине: top
  - Удалить элемент с вершины: pop
  - Проверить на пустоту: empty

# Стек в C++

```
stack<int> s;  
s.push(1);  
s.push(2);  
s.push(3);
```

Кладём  
на  
вершину



Тут  
вершина

```
while (!s.empty())  
{  
    cout << s.top() << " "; // 3 2 1  
    s.pop();  
}
```

Берем с  
вершины

# Очередь

- Структура данных «FIFO» (First-In First-Out).
- Операции:
  - Добавить элемент в конец очереди:  $O(1)$
  - Получить элемент с начала очереди:  $O(1)$
  - Удалить элемент с начала очереди:  $O(1)$
  - Определить, не пуста ли очередь:  $O(1)$

# Очередь

- Названия операций:
  - Добавить элемент в конец: `push`
  - Получить элемент в начале: `front`
  - Удалить элемент в начале: `pop`
  - Проверить на пустоту: `empty`

# Стек | Очередь

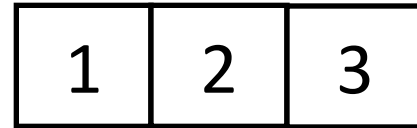
Операция	Стек	Очередь
Добавить	push	push
Удалить	pop	pop
Получить первый элемент / элемент на вершине	top	front
Проверить на пустоту	empty	empty

# Очередь в C++

```
queue<int> q;  
q.push(1);  
q.push(2);  
q.push(3);
```

```
while (!q.empty())  
{  
    cout << q.front() << " "; // 1 2 3  
    q.pop();  
}
```

Берем из  
начала



Начал  
о

Коне  
ц



# Словарь

- Структура данных, в которой можно сохранять и получать значения по произвольным ключам
- Операции:
  - Записать значение по ключу
  - Получить значение по ключу
  - Проверить наличие ключа в словаре
  - Получить размер словаря

# Словарь в C++ (map)

- Занимаемая память:  $O(n \log n)$
- Сложность операций:
  - Получить значение по ключу:  $O(\log n)$
  - Записать значение по ключу:  $O(\log n)$
  - Проверить наличие ключа:  $O(\log n)$
  - Получить размер словаря:  $O(1)$

# Словарь в C++ (map)

```
#include <map>
```

<ТИП КЛЮЧА, ТИП  
значения>



```
map<char, int> my_map;
```

```
my_map['A'] = 5;
```

```
my_map['B'] = 6;
```

```
cout << "A is " << my_map['A'];
```

Обращаемся как к  
массиву

# Словарь в C++ (map)

```
#include <map>
```

<ТИП КЛЮЧА, ТИП  
значения>

```
map<char, int> my_map;
```

```
my_map['A'] = 5;
```

```
if (my_map.find('A') != my_map.end())
```

```
{
```

```
    cout << "Key 'A' exists!";
```

```
}
```

# Итого

- Вектор (vector)
- Множество (set)
- Стек (stack)
- Очередь (queue)
- Словарь (map)