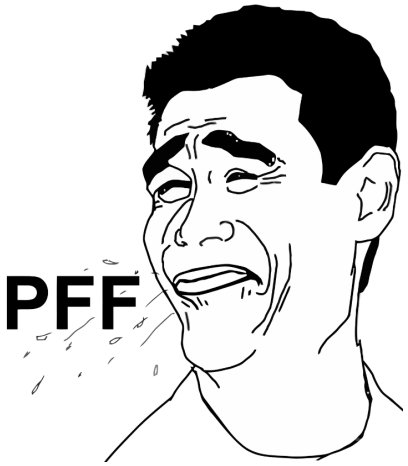
Массивы

Проблема

Задача 1: Вводится 3 числа. Вывести их в порядке возрастания.




```
int a, b, c;
std::cin >> a >> b >> c;
const int min = std::min(a, std::min(b, c));
const int max = std::max(a, std::max(b, c));
std::cout << min << ' ' << a + b + c - min - max << ' ' << max;
```

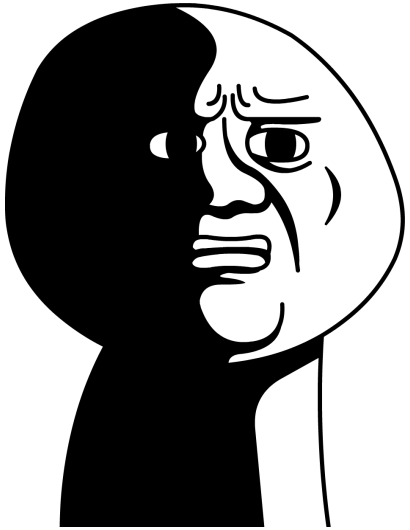
Проблема

Задача 2: Вводится 5 чисел. Вывести их в порядке возрастания.


```
int a, b, c, d, e;
std::cin >> a >> b >> c >> d >> e;
...
```

Проблема

Задача 3: Вводится 100 чисел. Вывести их в порядке возрастания.



```
???
```

Проблема

Задача 3: Вводится 100 чисел. Вывести их в порядке возрастания.

- Придумывание и записывание 100 различных переменных - нерациональное расходование времени.
- Циклы очень сильно упростили бы работу. Но перебирать переменные в цикле - невозможно.

Массив

Последовательность элементов **одного типа**, расположенных **непрерывно** в памяти, к которым имеется **доступ по индексу** через некоторый **уникальный идентификатор**.

```
int array[10];
```

Важно: выражение, стоящее в квадратных скобках должно быть положительным константным значением известным на этапе компиляции!

```
int n;
std::cin >> n;
int array[n];  // по Стандарту C++ так нельзя
```

Массив

Последовательность элементов **одного типа**, расположенных **непрерывно** в памяти, к которым имеется **доступ по индексу** через некоторый **уникальный идентификатор**.

```
int array[10];
```

- Данный массив хранит 10 `int`'ов
- Доступ к каждому элементу можно получить с помощью имени `array`, через операцию `[]`:

```
array[0], array[2], array[9]; // нумерация с 0
```

- **Гарантируется**, что все они идут в памяти подряд, без разрывов:

```
&a[i] - &a[j] == i - j;
```

Решение

Задача 3: Вводится 100 чисел. Вывести их в порядке возрастания.

```
int array[100];
for (int i = 0; i < 100; ++i) {
  std::cin >> array[i];
}
for (int i = 0; i < 100; ++i) {
  int min_idx = 0;  // индекс минимального
  for (int j = 1; j < 100; ++j) {
    if (array[j] < array[min_idx]) min_idx = j;
  }
  std::cout << array[min_idx] << ' ';
  array[min_idx] = 1'000'000;  // предполагаем, что значения меньше 1'000'000
}
```

Операции над массивами

```
int array[10];

array[1];  // операция доступа по индексу

sizeof(array);  // 40: возвращает суммарный размер в байтах

sizeof(array) / sizeof(int);  // 10: количество элементов

&array;  // адрес массива ( int(*)[10] )
```

Инициализация массивов

- Неинициализированный массив

```
int a[10];
```

- Инициализация нулями

```
int a[10]();
int b[20]{};
```

- Заполнение значениями

```
int a[5]{1, 2, 3};  // 1 2 3 0 0 (остальное заполняется нулями)
int b[]{1, 2, 3};   // 1 2 3     (размер вычисляется автоматически)
```

- Копирование запрещено :(

```
int b[5] = a;  // CE
```

Связь массивов и указателей

Связь массивов и указателей

```
int array[10];
```

В большинстве ситуаций массив автоматически преобразуется в указатель на свой нулевой элемент:

```
std::cout << array;       // выводится адрес нулевого элемента
std::cout << array + 5;   // адрес пятого элемента
```

Связь массивов и указателей

```
int array[10];
```

Верно и обратное - к указателям можно применять `[]`, то есть воспринимать их как массивы:

```
int* p = array;
std::cout << p[2];  // второй элемент массива

p += 5;
std::cout << p[2];  // седьмой элемент массива
```

Как это работает?

```
p[2];  // равносильно                *(p + 2)
2[p];  // да, настолько равносильно *(2 + p)
```

Связь массивов и указателей

Важно понимать, что массив $\neq$ указатель на первый элемент.

```
int array[10];
int* p = array;

std::cout << sizeof(array) << ' ' << sizeof(p);  // 40 8

&array;  // имеет тип int(*)[10]
&p;      // имеет тип int**
```

Но в большинстве ситуаций массив действительно ведет себя как указатель (неявно приводится к указателю).

Связь массивов и указателей

Сравнивать массивы с помощью операций $>$, $<$, $==$, ... не имеет смысла, так как реально сравниваются указатели на первые элементы.

```
int a[]{1, 2, 3};
int b[]{1, 2, 3};

a == a;  // всегда true
a == b;  // всегда false
```

Многомерный массив

Многомерный массив

Массивы могут иметь несколько размерностей

```
int a[10];  // одномерный массив
a[1];       // обращение к элементу

int b[5][20];  // двумерный массив
b[3][11];      // обращение к элементу

int c[7][9][3];  // трехмерный массив
c[3][8][0];      // обращение к элементу

...
```

Инициализация многомерных массивов

```
int a[2][3]{1, 2, 3, 4};  // заполнение по строкам
// 1 2 3
// 4 0 0

int b[][2]{1, 2, 3, 4};  // первая размерность может быть выведена
// 1 2
// 3 4

int c[3][2]{{1, 2}, {3, 4}, {5, 6}};
// 1 2
// 3 4
// 5 6
```

Многомерные массивы и указатели

Как и одномерные массивы многомерные приводятся к указателю на первый элемент. Первый элемент многомерного массива - массив меньшей размерности.

```
int a[1][2][3];
a[0];  // тип int[2][3]

int b[2][3];
b[1][1] <=> *(b[1] + 1) <=> *(*(b + 1) + 1) <=> *(&b[0][0] + 3 + 1);
```

Правила работы с массивами

(подробности позже)

Правила работы с массивами

1. Нельзя создавать массивы ссылок и массивы функций, но можно создавать массивы указателей и массивы указателей на функции.

```
int& a[10];       // Compilation error
int b[20](int);  // Compilation error
```

```
error: declaration of 'a' as array of references
error: declaration of 'b' as array of functions
```

```
int* c[30];         // Ok (массив указателей)
int (*d[40])(int); // Ok (массив указателей на функции)
```

Правила работы с массивами

2. Нельзя создать массив с неизвестным числом элементов, но можно его объявить.

```
int a[];                // Compilation error (определение)

int b[] = {1, 2, 3};    // Ok: int[3]

extern int c[];         // Ok (объявление)
```

```
error: storage size of 'a' isn't known
```

Правила работы с массивами

3. При сравнении массивов сравниваются адреса нулевых элементов (не значения!). Это значит, что результат сравнения на равенство для разных массивов всегда `false`.

```
int a[3]{1, 2, 3};

int b[3]{1, 2, 3};

std::cout << (a == b) << ' ' << (a == a);
```

```
0 1
```

Правила работы с массивами

4. Массивы нельзя присваивать друг другу (исключение - строки при инициализации).

```
int a[3];
int b[3]{1, 2, 3};
a = b;  // Compilation error
```

5. Лайфхак: если массив - это поле структуры или класса, то чудесным образом присваивание начинает работать

```
struct S {
    int array[3];
};

S c{1, 2, 3};
S d{4, 5, 6};
c = d;  // Ok (c.array == {4, 5, 6})
```

Динамические массивы

Динамические массивы

Создаются с помощью оператора `new[]`, который возвращает указатель на нулевой элемент массива.

```
int* array = new int[10];
```

Так как результат - указатель на элемент (не на массив!), его нельзя использовать в предыдущих контекстах

```
sizeof(new int[10]) == sizeof(int*);  // 8
sizeof(*array) == sizeof(int);        // true
```

Динамические массивы

Размер стекового массива, в отличие от динамического обязан быть константой времени компиляции!

```
int n = 10;

// Работаем с n ...

int a[n]{1, 2, 3};  // стандартом запрещено, но компиляторы позволяют

int* b = new int[n]{1, 2, 3};  // Ok
```

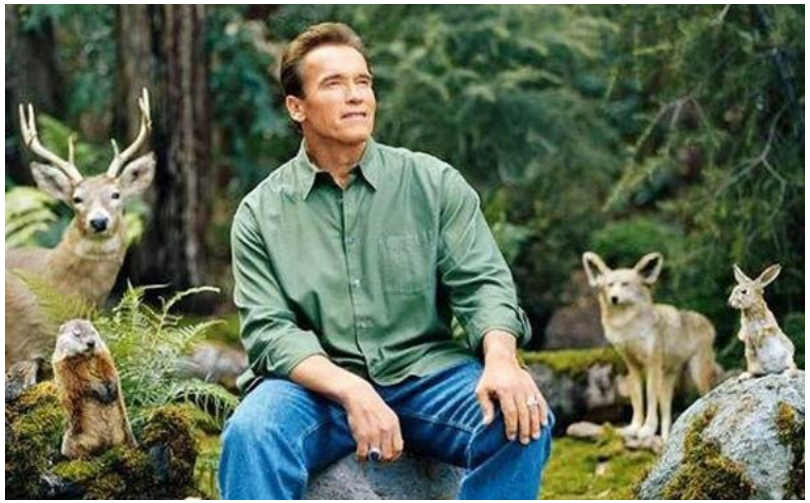
```
warning: ISO C++ forbids variable length array 'a'
```

Динамические массивы

И не забывайте убирать за собой!

Важно: память выделенную с помощью `new[]` необходимо очищать с помощью `delete[]`.

```
int* array = new int[10];
// ...
delete[] array;
```



Многомерные динамические массивы

Многомерный динамический массив можно моделировать массивом массивов:

```
// выделение памяти
int** array = new int*[n];
for (int i = 0; i < n; ++i) {
  array[i] = new int[m];
}

// заполнение
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < m; ++j) {
    std::cin >> array[i][j];
  }
}

// очищение
for (int i = 0; i < n; ++i) {
  delete[] array[i];
}
delete[] array;
```