погрешности

23.08

(закладывать в сравнение double константу Δ)

почему во второй раз считыв. быстрее!
в первый раз данные отпр. в КЭШ,
а во второй раз отпр. уже не надо

если хотим указать в стр. только 2 переменные, остальные не указывать →
→ указать в {} 2 перем., ост. не надо

{} - обнулить всё в структуре

```
void    ALLTests ()
{
   UTest (1, 1, 2, 3, NaN, NaN, 0);
   UTest (2, 1, 0, -4, -2, +2, 2);
   UTest (3, 1, 0, 0, 0, NaN, 1);
   .....
}
```

постоянно ук. вызов

похоже на цикл

```
a [nTests]
```

```
void   ALLTests ()
{
      const int nTests = 10;
      double a [nTests] = { 1, 1, 1, ... };
      double b [nTests] = { 2, 0, 0, ... };
```

```
for (int i=0; i<nTests; i++)
{
    UTest(i, 1, 0, -4, -2, +2, 2)
```

(неудобно, т.к. один принцип. неск. ряд)

⇒ удобнее сделать так:

UTest(i, $a_i$, ...)

запрос к языку: создать ряд однородн. объектов, к кот. можно обращ. по номеру → **массив**

```
    UTest(i, a[i], b[i], ...);
```

**ненадежность** (много массивов и их можно перепутать)

```
}
}
```

**опр** **Массив** – ряд однородных объектов, к которым можно обращ. по номеру.

нумерация массивов с 0 ⇒ тесты нумеровать с 0

идеальная функция (где нечего путать):

```
void AllTests()
{
```

**сделать массив структ.** (*)

```
    const int nTests = 10;
```

```
const int nTests = 6;
struct SP data[nTests] = {{1, 2, 3, NaN, Na
    {1, 0, -4, -2, +2, 2}, {1, 0, 0, 0, NaN, ...}
```

```
for (int i=0; i<nTests; i++)
    UTest (i, data[i]);
```

(*)
или можно писать с назв. полей:
{ .a=1, .b=2, .c=3, .... }

лучше: добавить комментарий с обозн.:
// a b c x1 x2 nRoots

размер массива запис. в квадр. скобках

глобальные перемен. влияет на работу каждой функции (много логических взаимод.) – проект взрыв. от кол-ва взаимод.

кол-во взаимод. = кол-во глоб. переменных * кол-во функций