

Пункт1. Разминка

В случае с gcc Debug приложение упадёт на ассёрте при попытке обратиться за границы string.

```
| λ build/srv_1          0 (0.988s) < 15:46:05
| 4
| 3
| 2
| 1
| 0
| /usr/include/c++/15.1.1/bits/basic_string.h:1369: constexpr std::__cxx11::basic_string<_CharT, _Tr
| aits, _Alloc>::reference std::__cxx11::basic_string<_CharT, _Traits, _Alloc>::operator[](size_type
| ) [with _CharT = char; _Traits = std::char_traits<char>; _Alloc = std::allocator<char>; reference
| = char&; size_type = long unsigned int]: Assertion '_pos <= size()' failed.
| fish: Job 1, 'build/srv_1' terminated by signal SIGABRT (Abort)
```

В случае с gcc Release приложение упадёт с sigsegv из-за выхода за границы string.

```
| λ build/srv_1          0 (1.0
| 4
| 3
| 2
| 1
| 0
| fish: Job 1, 'build/srv_1' terminated by signal SIGSEGV (Address boundary error)
```

Upd1: выход за границы строки происходит из-за переполнения, условие цикла $i \leq 0$ выполняется и еще раз отнимается единица (когда $i=0$), для unsigned типов в стандарте задано поведение, что при таких случаях число перейдёт к другой своей границе, в данном случае к $\text{std::numeric_limits<unsigned int>::max}()$ (UINT_MAX). И по сути в программе будет следующий вызовы $s[\text{UINT_MAX}]$, а это явно больше чем размер строки.

Пункт 1.2. Упражнение 1.

В случае Debug мы получим следующее значение:

```
| λ build/srv_1
Duration: 25390123
```

В случае Release мы получим следующее значение:

```
| λ build/srv_1
Duration: 44
```

Так происходит потому что в релизе используются различные оптимизации компилятора, а такие оптимизации могут удалять неиспользуемый код. Так как значение факториала мы больше нигде не используем (никуда не пишем, нигде не выводим), то компилятор просто убирает этот код и логика не попадает в бинарник.

Пункт 1.3. Упражнение 2.

В случае выполнения с двумя потоками получаем следующее время:

```
| λ build/srv_1
Duration: 23004816
```

В случае выполнения последовательно получаем следующее время:

```
| λ build/srv_1  
Duration: 43192570
```

Это происходит из-за того что в первом случае у нас потоки действуют независимо друг от друга и только главный поток ждёт пока завершится работа. А во втором случае ждём пока завершится первый вызов функции вычисления факториала и второй.