Знайомство з функціональним програмуванням

2019

Основні питання

- Знайомство з функціональним програмуванням:
  - мова функціональним програмування *Haskell;*
  - списки;
  - *list comprehension.*
- Визначення функцій рівняннями. Зіставлення зі зразком. Операційна семантика.
- Лінивість обчислень.

https://www.haskell.org/

Downloads Community Documentation News

Haskell

An advanced, purely functional programming language

Declarative, statically typed code.

```
primes = filterPrime [2..]
  where filterPrime (p:xs) =
          p : filterPrime [x | x <- xs, x `mod` p /= 0]
```

Try it!

Type Haskell expressions in here.

```
λ take 9 (zipWith (+) [1..] [10,20..])
[11,22,33,44,55,66,77,88,99] :: (Enum a, Num a) => [a]
λ
```

Got 5 minutes?

Type `help` to start the tutorial.

Or try typing these out and see what happens (click to insert):

`23 * 36` or `reverse "hello"` or `foldr (:) [] [1,2,3]` or `do line <- getLine; putStrLn line` or `readFile "/welcome"`

These IO actions are supported in this sandbox.

Try it!

Type Haskell expressions in here.

```
λ take 9 (zipWith (+) [1..] [10,20..])
[11,22,33,44,55,66,77,88,99] :: (Enum a, Num a) => [a]
λ
```


Videos

Escape from the ivory tower: The Haskell journey, by Simon Peyton-Jones

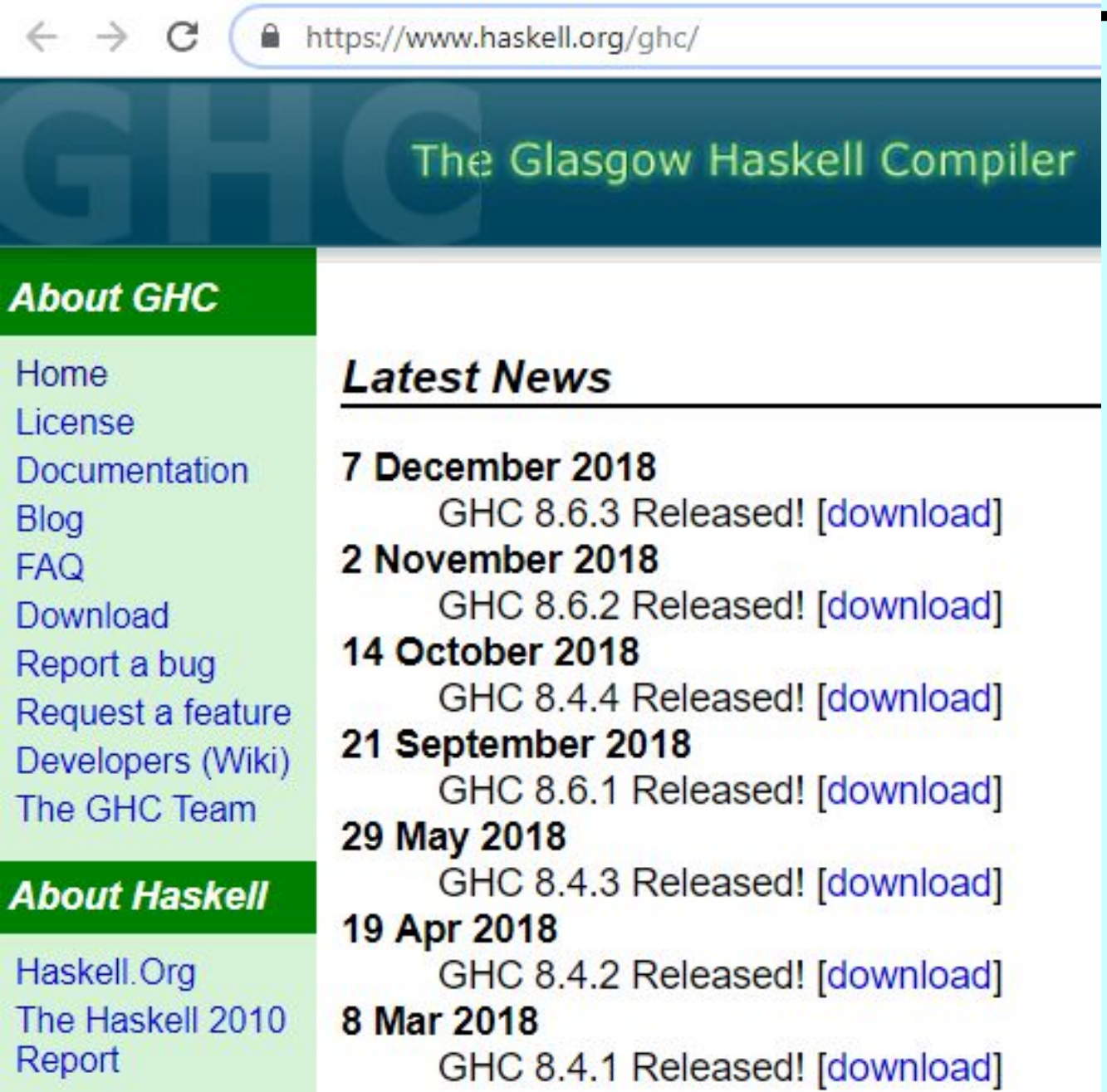
Haskell take limiting side for parallel programming, by Ryan Newton

Lentczner

Other Tales, by Katie Miller

Spock, by Oskar Wickström

https://www.haskell.org/ghc/



List comprehension.
Функція швидкого сортування

Одна із спискових родзинок (синтаксичний цукор)

Генератор

Охорона

```
qsort []          =  []
qsort (x:xs)      =  qsort [y | y <- xs, y<x ]
                      ++ [x]
                      ++ qsort [y | y <- xs, y>=x]
```

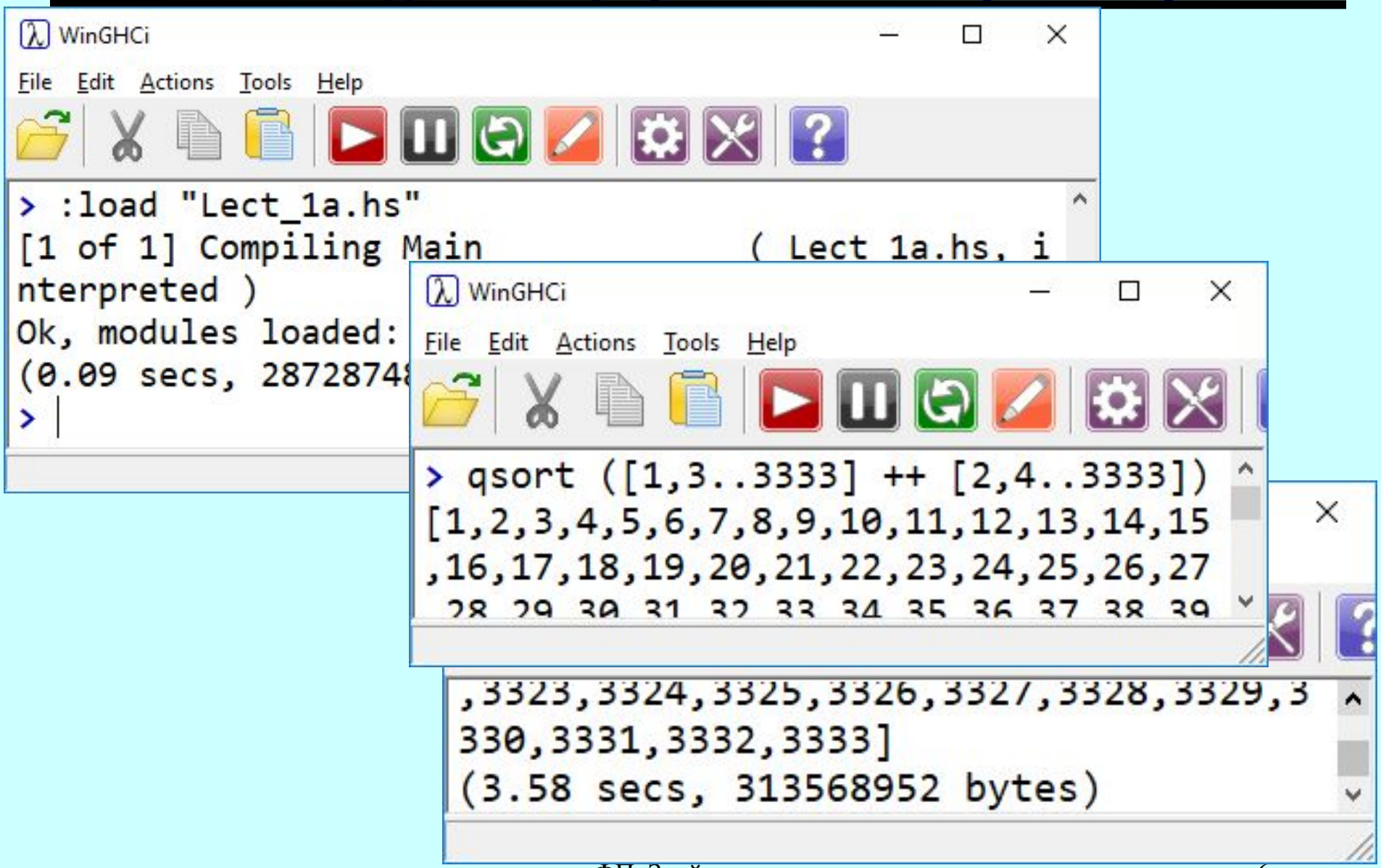
Функція швидкого сортування

Варто відзначити

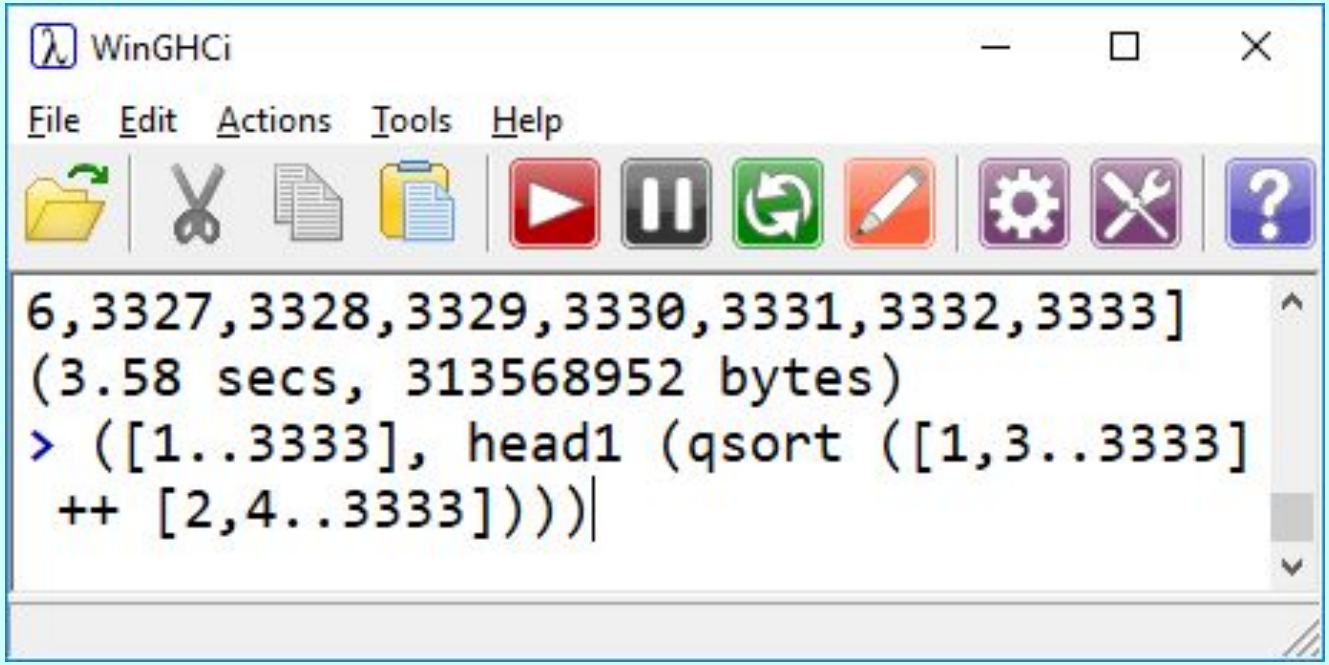
Функції визначаються рівняннями

Ключові терміни операційної семантики:
редукція та зіставлення зі зразком (*pattern matching*)

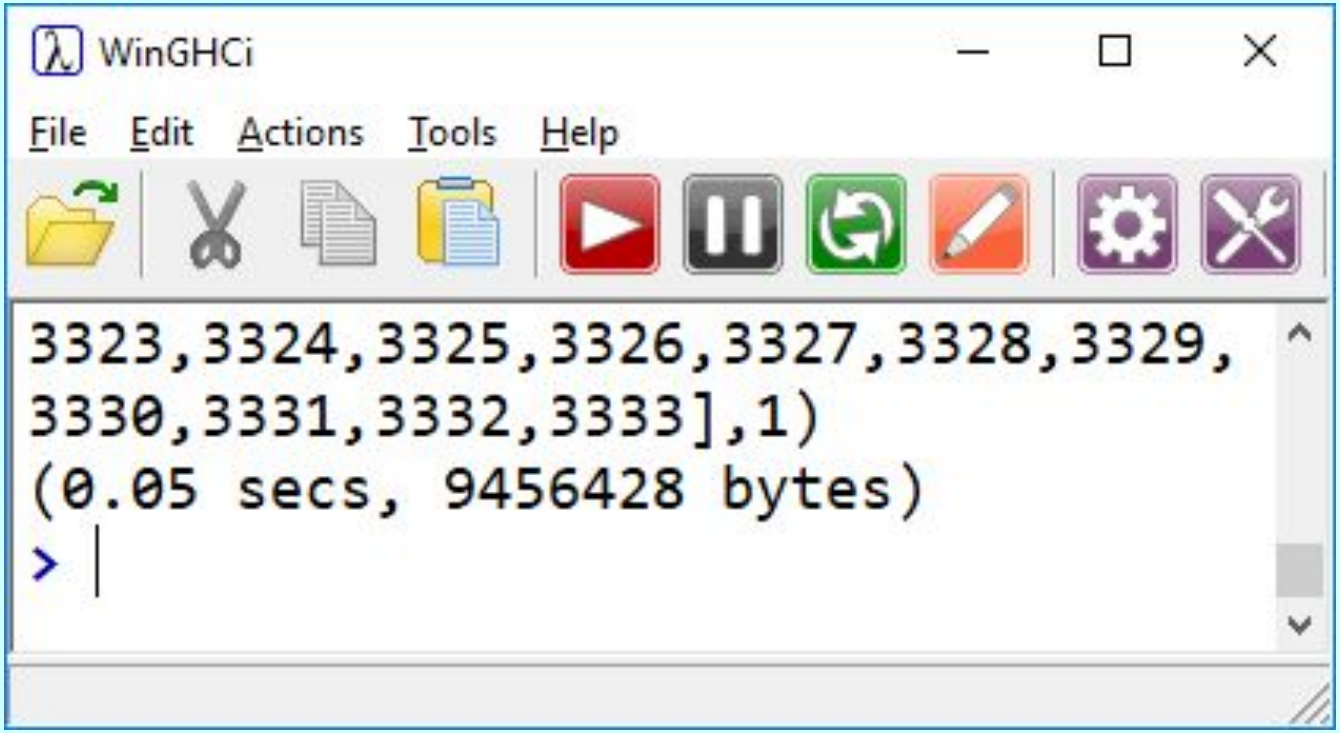
The Glasgow Haskell Compiler.
Функція швидкого сортування. Експериментуємо...



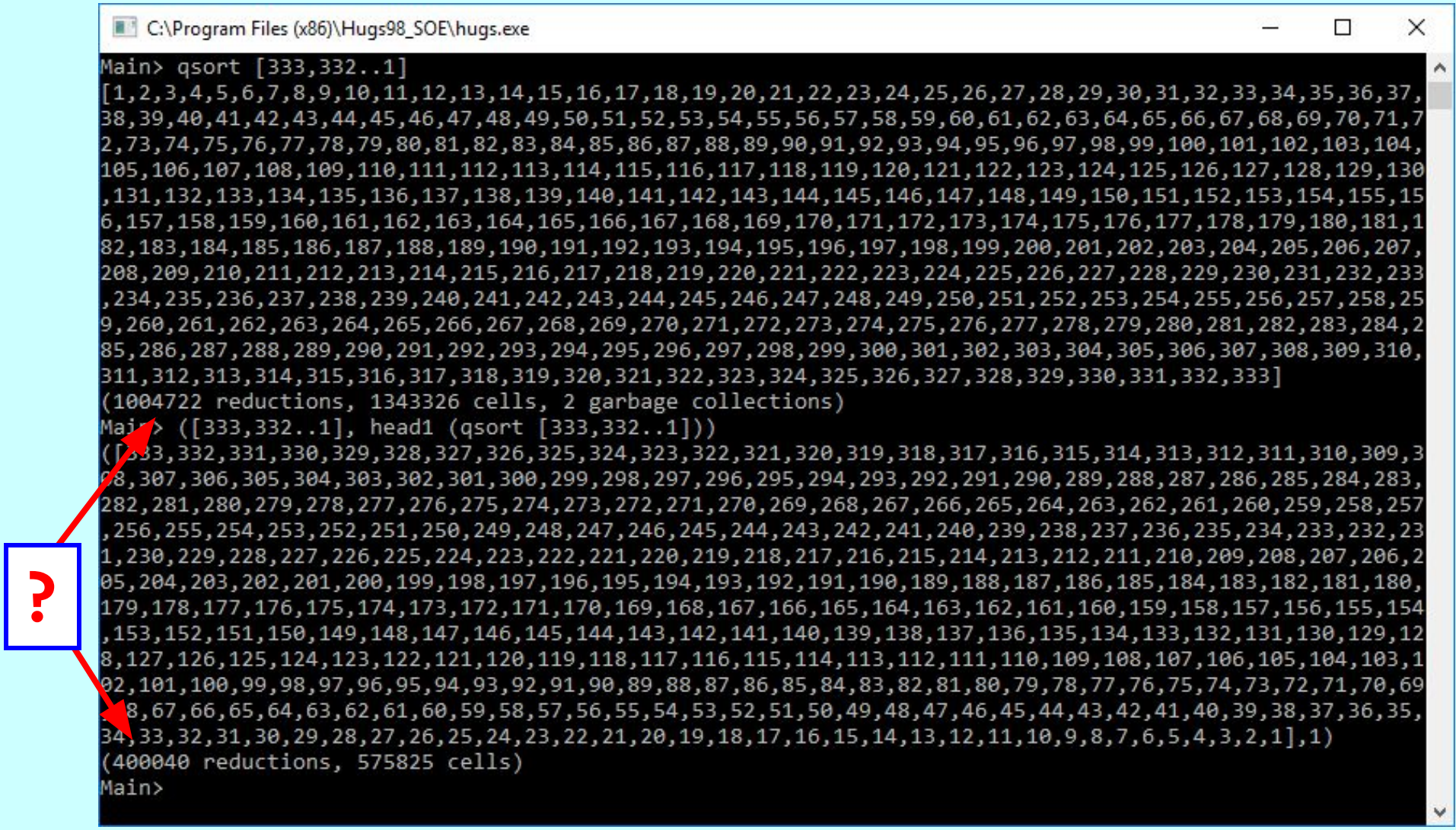
Лінивість обчислень (1/2)



Лінивість обчислень (2/2)



Функція швидкого сортування. Експериментуємо...



List comprehension.
Декартовий добуток

```
dekart xs ys = [(x,y) | x <- xs, y <- ys]
```

Два генератори

```
> dekart [1,2,3] ['a', 'b', 'c']
[(1,'a'),(1,'b'),(1,'c'),(2,'a'),(2,'b'),(2,'c'),(3,'a'),(3,'b'),(3,'c')]
```

List comprehension:

- генератори;
- охорони;
- локальні визначення (конструкції ***"let ... in..."*** та ***"where ..."***).

Список простих чисел. (1/3)

Функція, що генерує (нескінченний!) список простих чисел

```
allPrimeNumbers_v0 = [n | n <- [2..], isPrimeLoc n == True]
    where isPrimeLoc n | n <= 1     = False
                       | otherwise  = getDivisorsLoc n == [1,n]
          getDivisorsLoc n
                     | n < 1 = []
                     | otherwise = [x | x <- [1..n], mod n x == 0 ]
```

```
> otherwise
True
```

List comprehension:

- генератори;
- охорони;
- локальні визначення (конструкції ***"let ... in..."*** та ***"where ..."***).

Список простих чисел. (2/3)
Операторна форма бінарних функцій

Функція обчислення (списку) дільників

```
getDivisors num
  | num < 1              = []
  | otherwise            = [x | x <- [1..num], num `mod` x == 0 ]
```

Бінарна функція ***mod*** подана в ***операторній*** формі (із використанням зворотних апострофів)

Операторна форма бінарної функції (***mod***)

Функція перевірки числа на простоту

```
isPrime num
  | num <= 1             = False
  | otherwise            = getDivisors num == [1,num]
```

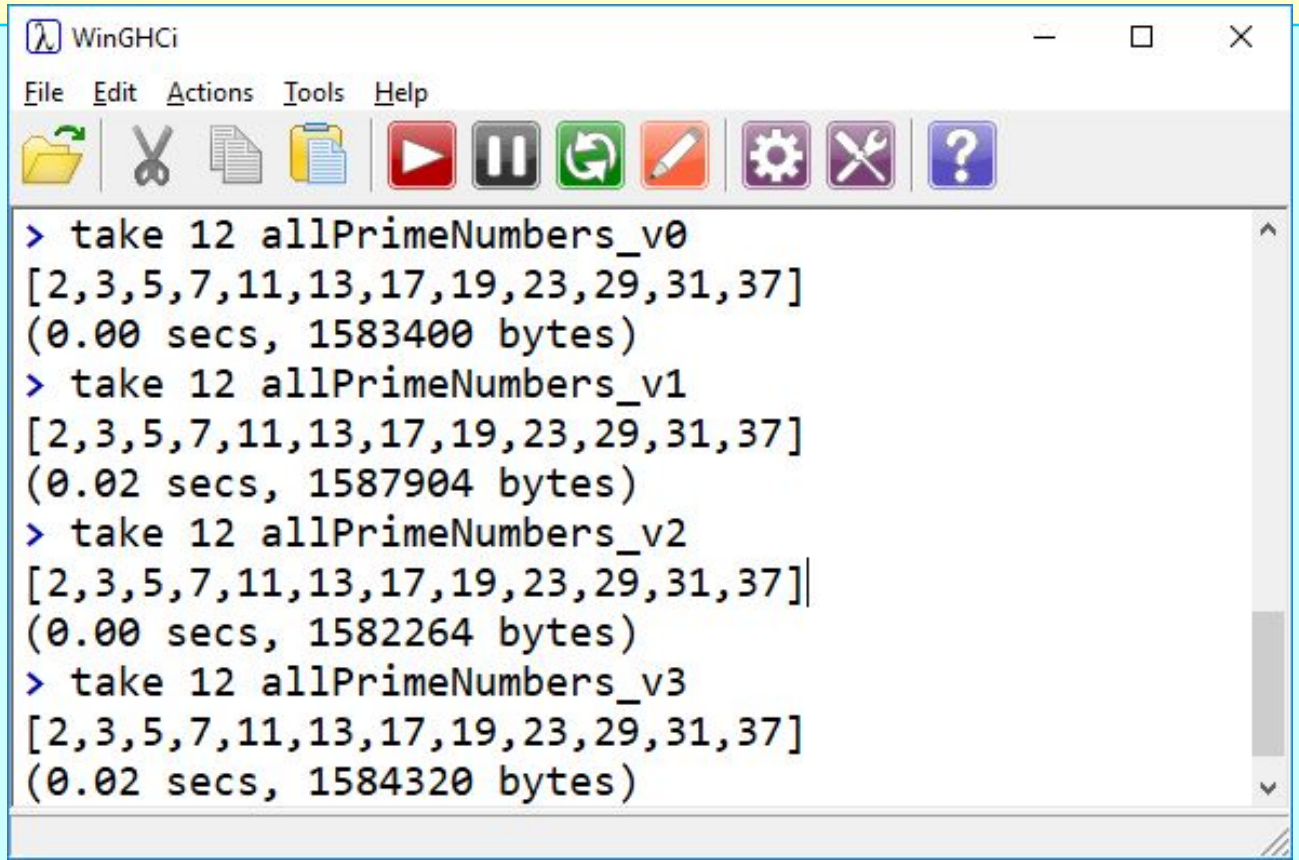
Функція, що генерує (нескінченний!) список простих чисел

```
allPrimeNumbers_v1 = [2] ++ [n | n <- [3,5..], isPrime n == True]
```

Список простих чисел. (3/3)
Ще дві версії

```
sieve1 (x : xs) = x : (sieve1 ys)
                  where ys = [n | n <- xs, n `mod` x /= 0]
allPrimeNumbers_v2 = sieve1 [2..]
```

```
sieve2 (x : xs) = let ys = [n | n <- xs, n `mod` x /= 0] in
                          (x : (sieve2 ys))
allPrimeNumbers_v3 = sieve2 [2..]
```



Парадигма функціонального програмування

- Опис (визначення) функцій
- Обчислення функцій

Не імперативний стиль із незмінюваними даними (не використовується поняття послідовного виконання, немає змінних, немає присвоювань)

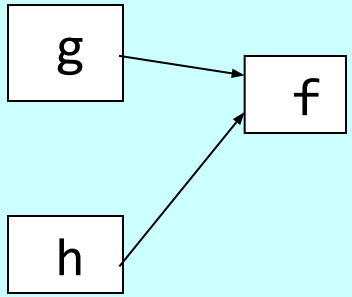
Чистота функцій (відсутність побічних ефектів): два обчислення функції з однаковими аргументами завжди дають один і той же самий результат!

Повернення до ***чистих*** функцій.

- Машинні бібліотеки функцій.
- *Algol*-60: функції із ***побічними ефектами*** (червоність).

Що дає функціональна парадигма?

- Можливість заміни виразу на результат обчислення цього виразу (не потрібні переобчислення).
- Спрощене тестування.
- Природні підходи до розпаралелювання.
- Спрощена інтеграція (інтеграція – один з найбільших ризиків в інженерії програмних систем).



Функціональне програмування. Теоретичні засади

- Формалізація семантики функціональних програм фактично спряжена з ***уточненням власне функцій*** та ***уточненням аплікації*** (уточненням застосування функції до аргументів).

  Теоретичні засади такої формалізації давно відомі. Це – ***лямбда-числення***.

  Окремі штрихи теоретичного підгрунтя :

  – чисте лямбда-числення – це числення ***анонімних*** функцій;
  – ***бета-редукція*** – основа трактування аплікації (тобто є засадою ***операційної семантики***);
  – можливі різні ***стратегії*** використання бета-редукції, зокрема, є стратегія, спряжена із так званими ***лінивими*** обчисленнями, є стратегія, спряжена із так званими ***енергійними*** обчисленнями;
  – теорема про нерухому точку (для визначення функцій, що задаються рекурсивними рівняннями).

- Можна обмежитись ***одноаргументними*** функціями, спираючись на ***каррінг*** функцій.

Haskell

Де-факто, *Haskell* є стандартом мов функціонального програмування

На *Haskell* реалізовано багато складних проектів. Ось деякий їх перелік за джерелом

`http://www.ibm.com/developerworks/ru/library/l-haskell/?S_TACT=105AGX99&S_CMP=GR01`

- Компілятори й інші засоби розробки.
- Розподілена система керування версіями *Darcs*.
- Віконний менеджер *xmonad*.
- Сервер *Web*-додатків *HAppS*.
- Інтерпретатор/компілятор *Pugs* для мови *Perl* 6.
- Операційна система *House*.
- Мова опису апаратних засобів *Lava*.
- Система обробки природної мови *LOLITA*.
- Системи доведення теорем *Equinox* / *Paradox* і *Agda*.

Література

1) Душкин Р. В. Функциональное программирование на языке Haskell. М.: ДМК Пресс, 2007.

2) Душкин Р. В. Справочник по языку Haskell. М.: ДМК Пресс, 2008.

3) Душкин Р. В. Практика работы на языке Haskell. М.: ДМК Пресс, 2009.

4) Липовача М. Изучай Haskell во имя добра! М.: ДМК Пресс, 2012.

5) Lipovača M. Learn You a Haskell for Great Good! Miran Lipovača. : No Starch Press», 2011.

6) Роганова Н. А. Функциональное программирование, 2002.

7) Филд А., Харрисон П. Функциональное программирование. М.: Мир, 1993.

8) Хендерсон П. Функциональное программирование. Применение и реализация. М.: Мир, 1983.

9) Хьюдак П., Петерсон Дж., Джозеф Фасел Дж. Мягкое введение в haskell. Учебник.

https://repl.it