STU FEI

Funkcionálne programovanie

1. Úvod do funkcionálneho programovania, vlastnosti funkcionálneho programovania, Glasgow Haskell kompilátor

Martin Drozda
martin.drozda@stuba.sk
C601

Prednáška: utorok 13:00-15:00 AB150
Cvičenie: streda 15:00-17:00 CD150

Evaluácia 2022

Páčilo sa mi nahliadnuť na programovanie z inej strany. Pri tomto predmete sa mnohí museli naučiť znova chodiť respektíve ešte horšie zabudnúť chodiť a naučiť sa to inak. Páčilo sa mi tiež zadanie, ktoré bolo veľmi praktické a unikátne.

Podmienky absolvovania

Zadanie: počas semestra (50 bodov, min. 20)

Riadny / opravný termín (50 bodov, min. 20)

Pozvaná prednáška: povinná a vopred oznámená

Cvičenia

Niekoľko “menších” zadaní počas semestra

Väčšie zadanie: téma bude oznámená

Literatúra

Real World Haskell. Bryan O'Sullivan, Don Stewart, and John Goerzen, 2008.

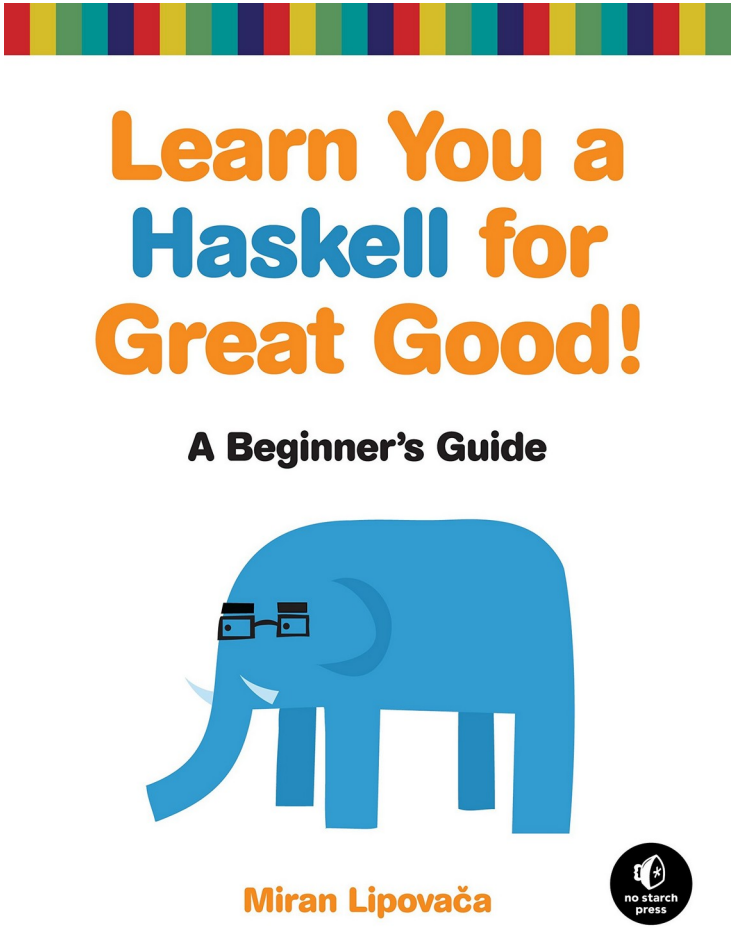
Learn You a Haskell for Great Good!: A Beginner's Guide. Miran Lipovaca, 2011.

Thinking Functionally with Haskell. Richard Bird, 2014.

What I Wish I Knew When Learning Haskell. Stephen Diehl, 2020.

Literatúra

http://learnyouahaskell.com/chapters



Haskell Curry (1900-1982)

Americký matematik a logik

PhD študent Davida Hilberta (1930)

Logika kombinátorov: prednáška λ-kalkul (neskôr)

Programovacie paradigmy

Imperatívne programovanie

- Postupnosť krokov

Deklaratívne programovanie

- Popisuje žiadaný výstup výpočtu bez špecifikácie potrebných krokov (utópia alebo realita?)

Programovacie paradigmy

Imperatívne programovanie

- Procedurálne programovanie
- OOP

Deklaratívne programovanie

- Funkcionálne programovanie
- Logické programovanie
- Databázové programovanie

```
SELECT * FROM customers WHERE age < 50;
```

Funkcionálne programovanie

Funkcie sú „first-class citizen“, je ich možné viazať na premenné, používať ako argumenty, je ich možné vrátiť ako ostatné dátové typy.

Funkcie môžu byť „pure“, teda pre dané hodnoty argumentov, vždy vrátia rovnaký výsledok.

Funkcie umožňujú kompozíciu.

Programovacie jazyky

Fortran: 1954, John Backus (tiež Backus-Naur form)

Lisp: 1958, John McCarthy (tiež AI)

Algol: 1958, kolektív autorov

Cobol: 1959, Short Range Committee

Basic: 1964, John G. Kemeny, Thomas E. Kurtz

C: 1972, Dennis Ritchie

C++: 1985, Bjarne Stroustrup

Java: 1995, James Gosling

PHP: 1995, Rasmus Lerdorf

Kotlin: 2011, JetBrains

Funkcionálne prog. jazyky

Lisp: 1958, John McCarthy

ML: 1973, Robin Milner

Scheme: 1975, Guy L. Steele, Gerald Jay Sussman

Erlang: 1986

Haskell: 1990

Clojure: 2007, Rich Hickey

A tiež: C++, Kotlin, Python, Scala, Java...(lambda funkcie)

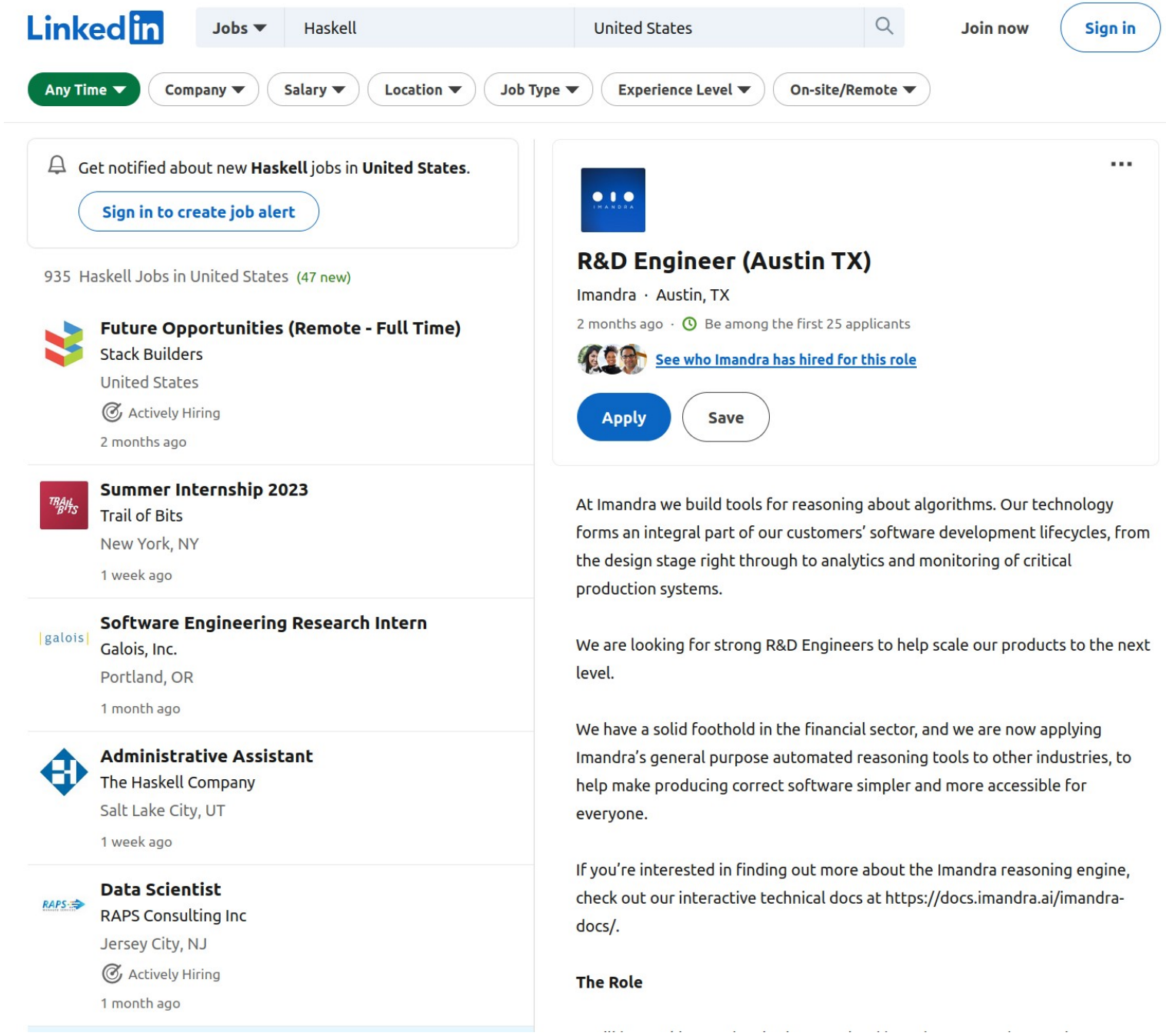
Haskell: Facebook

https://engineering.fb.com/2015/06/26/security/fighting-spam-with-haskell/

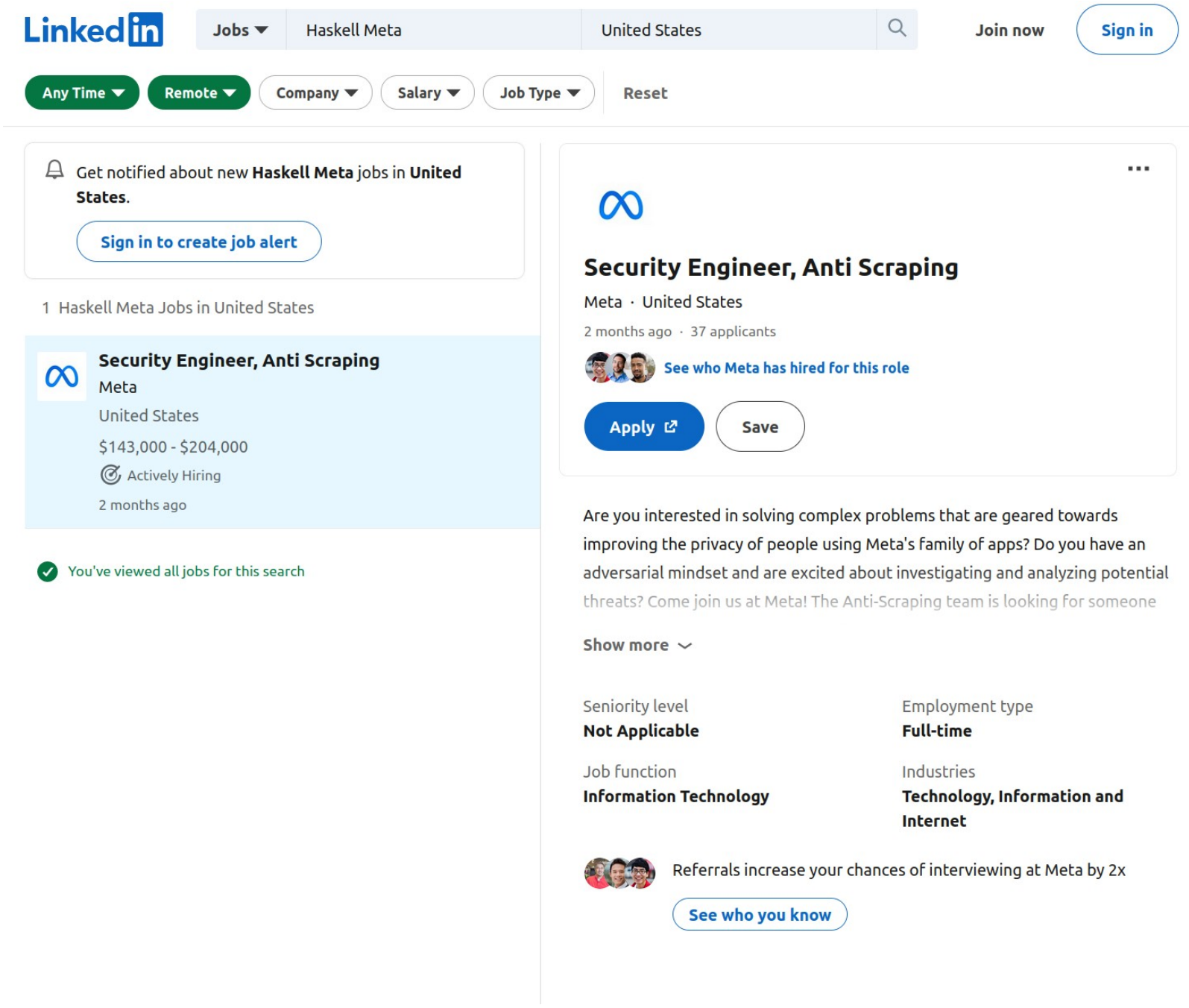
"One of our weapons in the fight against spam, malware, and other abuse on Facebook is a system called Sigma. Its job is to proactively identify malicious actions on Facebook, such as spam, phishing attacks, posting links to malware, etc. Bad content detected by Sigma is removed automatically so that it doesn't show up in your News Feed.

We recently completed a two-year-long major redesign of Sigma, which involved replacing the in-house FXL language previously used to program Sigma with Haskell. The Haskell-powered Sigma now runs in production, serving more than one million requests per second."

Haskell: jobs



Meta


LinkedIn
Jobs
Haskell Meta
United States
Join now
Sign in
Any Time
Remote
Company
Salary
Job Type
Reset
Get notified about new Haskell Meta jobs in United States.
Sign in to create job alert
1 Haskell Meta Jobs in United States
Security Engineer, Anti Scraping
Meta
United States
$143,000 - $204,000
Actively Hiring
2 months ago
You've viewed all jobs for this search
Security Engineer, Anti Scraping
Meta · United States
2 months ago · 37 applicants
See who Meta has hired for this role
Apply
Save
Are you interested in solving complex problems that are geared towards improving the privacy of people using Meta's family of apps? Do you have an adversarial mindset and are excited about investigating and analyzing potential threats? Come join us at Meta! The Anti-Scraping team is looking for someone
Show more
Seniority level
Not Applicable
Employment type
Full-time
Job function
Information Technology
Industries
Technology, Information and Internet
Referrals increase your chances of interviewing at Meta by 2x
See who you know

C++: lambda

```
auto f = [](int init) -> int { //bez capture
   return ++init;
 };

int (*f)(int) = [](int init) {
   return ++init;
 };
```

Notácia

Prefixová, tiež poľská alebo Łukasiewicz notácia

- + 2 3
- (+) 2 3 (Haskell)
- (*) ((-) 5 6) 7, teda (5 – 6) * 7 v infix notácii

Infixová, pomocou operátora

- 2 + 3 (aj Haskell)

Postfixová, tiež reverzná poľská notácia

- 2 3 + (hodnoty do registrov, potom načítanie operácie)

Prefix vs infix (Haskell)

```
mod 5 3
5 `mod` 3


myFunction x y = x * y
myFunction 2 3
2 `myFunction` 3
```

Výraz

```
myScore x =
    if x > 90 then "You got a A"
    else if 80 < x && x < 90 then "you got a B"
    else if 70 < x && x < 80 then "You got a C"
    else if 60 < x && x < 70 then "you got a D"
    else "You got a F"
```

Výraz musíme vyhodnotiť, teda if musí mať then časť, ako aj else časť.

if-else v C/C++/Java nie je výraz, ale statement/príkaz

thunk

Thunk je nevyhodnotený výraz, najmä v kontexte “lenivej” evaluácie

Lazy object v iných jazykoch, ako napr. Kotlin

Evaluácia

Okamžitá (eager)

- int a = factorial(10); v C/C++ nastane okamžité zavolanie funkcie a jej vyhodnotenie = 3628800

Lenivá (lazy)

- let a = product [1..10], výpočet nastane, keď evaluácia bude potrebná, možno nikdy
- [1..10], Haskell zápis pre list 1, 2, …, 10
- [1..], nekonečný list je OK, pretože bude vyhodnotený len ak bude treba
- let a = product [1..], vyhodnotenie nastane neskôr

Polymorfizmus

Ad hoc

- preťažovanie funkcií, napr. +

Inclusion

- dynamické (neskoré) viazanie (virtuálne metódy)

Parametric

- skoré viazanie (počas kompilácie, template v C++)

Coercion (implicitná typová konverzia)

- Implicitné pretypovanie napr. Int na Float (Haskell to neumožňuje, ale...niekedy je odvodený iný typ napr. Num)
- konverzný konštruktor v C++

Haskell

Purely functional

- Referenčná transparentnosť: výraz je možné nahradiť jeho vyhodnotením, všetky „pure“ funkcie sú referenčne transparentné. Referenčná transparentnosť je širší pojem ako čistá funkcia.
- Každá „pure“ funkcia jazyka Haskell je funkcia v matematickom zmysle

Statically typed

- Typ každého výrazu je odvodený počas kompilácie
- Typ nie je potrebné uviesť, bude odvodený, ale je lepšie ho uviesť :)
- let a = 1 Aký typ bude odvodený? Nie Int, ale Num.

RT vs lenivá evaluácia

Referenčná transparentnosť (RT) je potrebná pre lenivé vyhodnocovanie.

Ak odložíme evaluáciu výrazu na neskoršie, je potrebné, aby jeho vyhodnotenie teraz a neskôr bolo identické. A RT garantuje práve túto vlastnosť.

Side effect

Side effect nastane, keď funkcia číta a/alebo modifikuje externé parametre (vonkajší stav).

Príklad:

- Funkcia obsahuje I/O
- Funkcia používa globálne premenné: v striktnom zmysle čítanie globálnej premennej nie je side effect, ale musíme garantovať, že čítaním nenastane žiadna zmena externého stavu, čo nemusí byť garantovateľné
- Funkcia obsahuje rand(), pretože vyžiadanie (pseudo-)náhodného čísla zmení stav generátora

Pure funkcia

Pure funkcia neobsahuje side effect.

```
int add(const int a, const int b) const { //C++
   return a + b;
 };
```

Metóda add() pre tie isté vstupné parametre vráti tú istú hodnotu, zároveň nemodifikuje stav objektu. Pure funkcie sa dajú optimalizovať, inlinovať, v niektorých prípadoch aj evaluovať počas kompilácie.

Haskell

Lazy (lenivá evaluácia)

Pure funkcie

Immutable expressions: hodnota výrazu sa po evaluácii nemôže zmeniť.

Garbage collection (aj vďaka immutable expressions)

Haskell

Haskell98

Haskell2010

“Haskell 2010 is an incremental update to the language, mostly incorporating several well-used and uncontroversial features previously enabled via compiler-specific flags.”

Zdroj: https://en.wikipedia.org/wiki/Haskell_%28programming_language%29#Haskell_2010

Hackage, Stackage

Hackage: The Haskell Package Repository

https://hackage.haskell.org/

„Hackage is the Haskell community's central package archive of open source software.”

Stackage: Stable Haskell package sets

https://www.stackage.org/

“A distribution of compatible Haskell packages from Hackage that build together”

cabal, stack

cabal install PACKAGE

cabal install parallel

stack install parallel

cabal používa Hackage

stack používa Stackage

Základné typy

() - The unit type („void“)

Char - A single unicode character

String - Strings

Text - Unicode strings

Bool - Boolean values

Int - Machine integers

Integer - GMP arbitrary precision integers

Float - Machine floating point values

Double - Machine double floating point values

Parametrizované typy

[a] – List s elementmi typu a

(a,b) – Dvojica s elementmi typu a, b

(a,b,c) – Trojica s elementmi typu a, b, c

[1, 2] : OK

[1, "hello"] : chyba, 2 rozdielne typy

Hello, factorial

```
int n = 10;
unsigned long long fact = 1;
for(int i=1; i<n; ++i) {
  fact *= i;
}
```

```
fact :: Integer -> Integer
fact 0 = 1
fact n = n * fact (n-1)  -- lenivá evaluácia??
```

funkcia

```
add :: Int -> Int -> Int
add x y = x + y
```

add má dva argumenty typu Int a vráti hodnotu typu Int.

```
add 1 2 //vráti 3
```

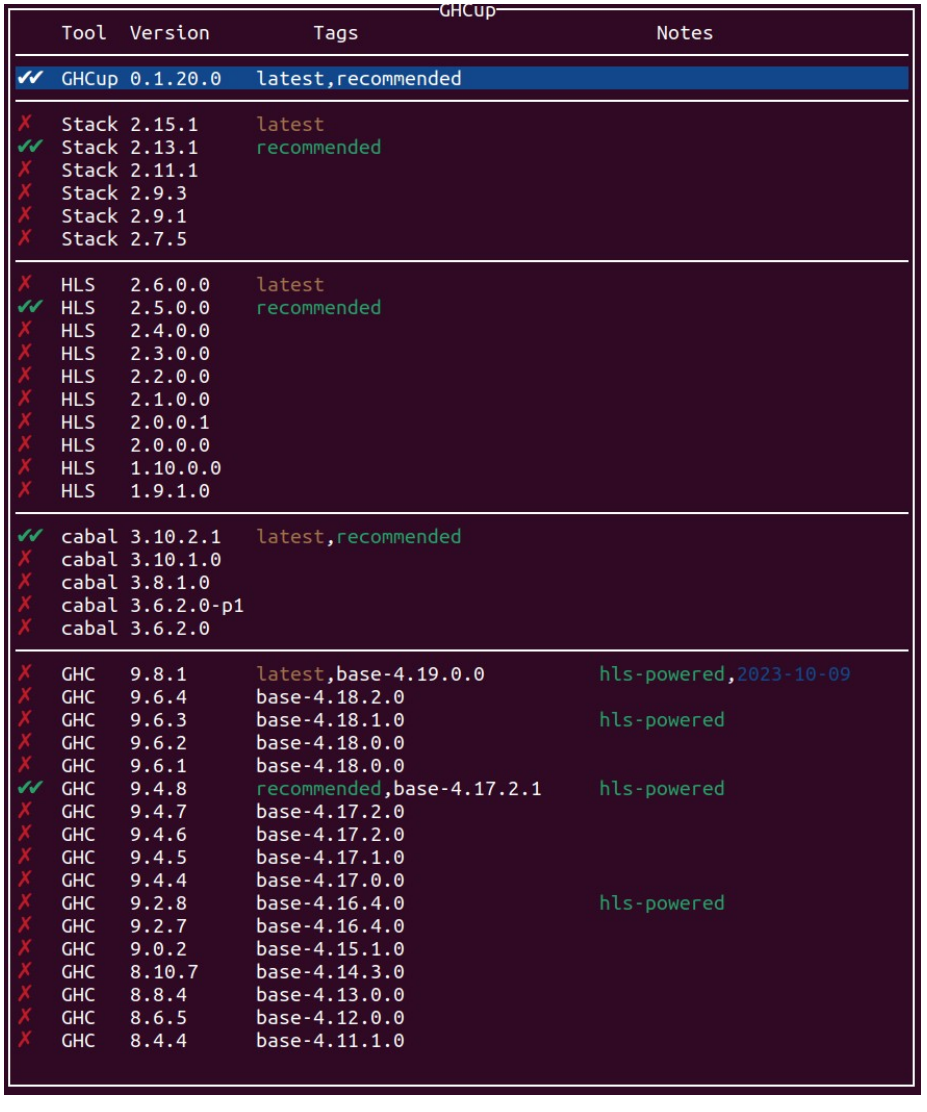
add (1 2) alebo add (1, 2) je nesprávne volanie

Glasgow Haskell compiler

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

Inštalácia: https://www.haskell.org/ghcup/

ghcup tui



GMP

GNU Multiple Precision Arithmetic Library

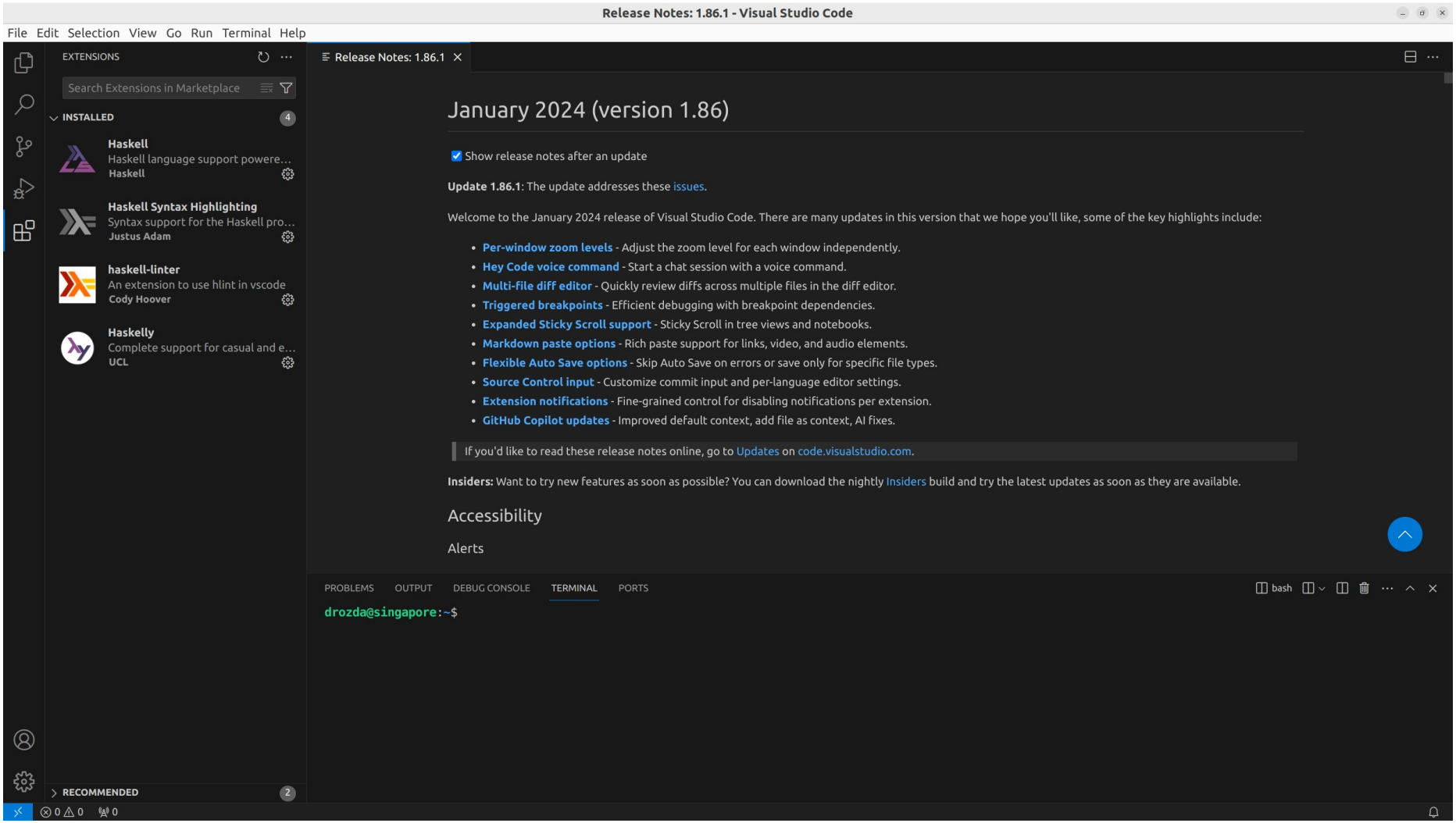
sudo apt install libgmp3-dev

```
drozda@singapore:~$ ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help

<no location info>: warning: [-Wmissed-extra-shared-lib]
    libgmp.so: cannot open shared object file: No such file or directory
    It's OK if you don't want to use symbols from it directly.
    (the package DLL is loaded by the system linker
     which manages dependencies by itself).
ghci>
```

IDE

emacs

Visual Studio Code



ghci

ghci : Prelude interpreter

Command	Shortcut	Action
:reload	:r	Code reload
:type	:t	Type inspection
:info	:i	Information
:load	:l	Set the active Main module in the REPL
:quit	:q	Leave ghci

ghc

ghc : kompilátor

ghc -O2 -Wall 1.hs

-O0 bez optimalizácie

-O1 -O2 s optimalizáciou

-Wall warnings, ale nie všetky

-dynamic dynamické linkovanie modulov (kratší spustiteľný kód)

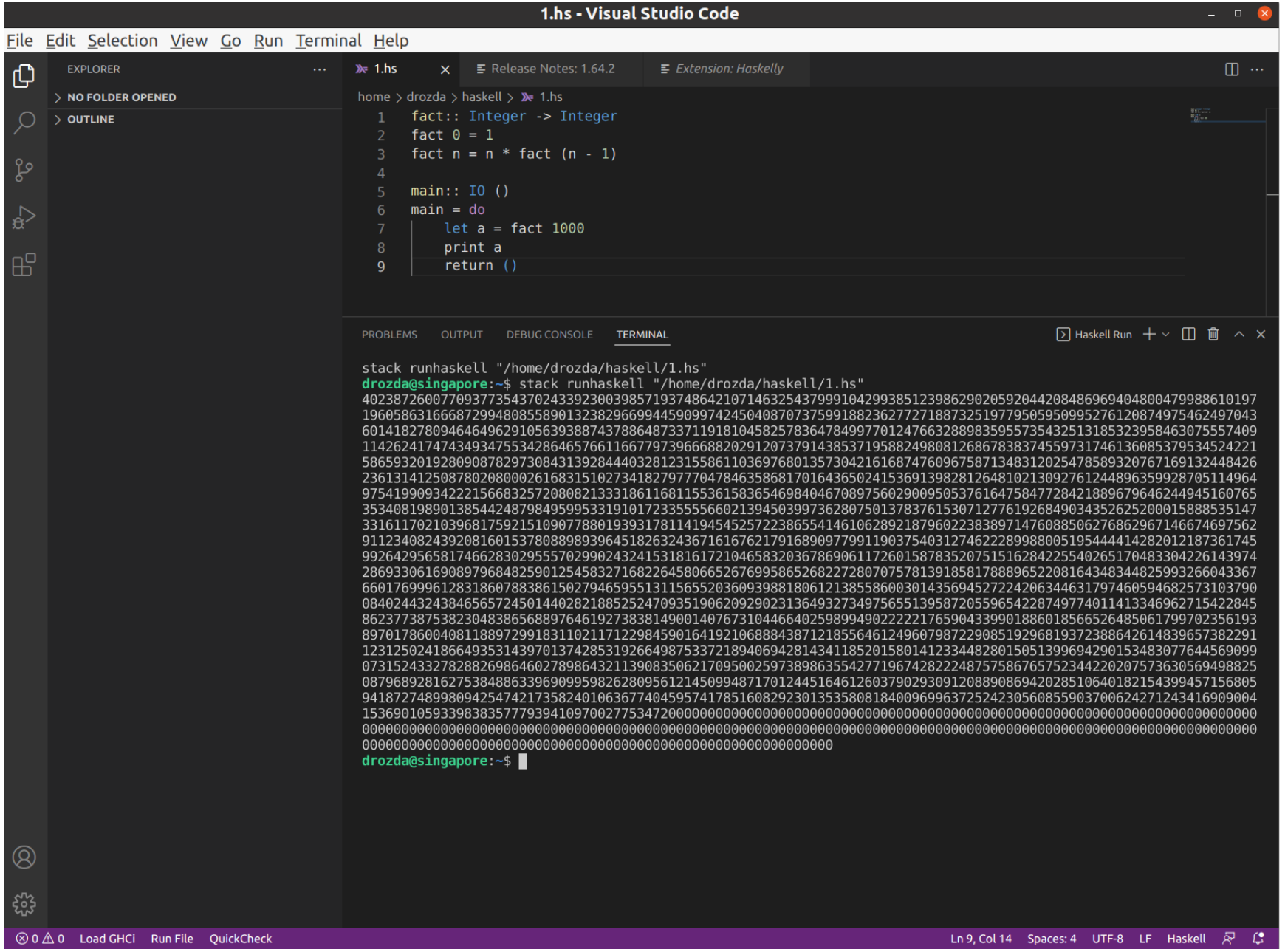
Hello world!

```
main:: IO ()
main = print "Hello world"


main:: IO ()
main = do
    print "Hello world"
```

main

```
fact:: Integer -> Integer
fact 0 = 1
fact n = n * fact (n - 1)


main:: IO ()
main = do
    let a = fact 1000
    print a
    return ()
```

main


1.hs - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
NO FOLDER OPENED
OUTLINE
1.hs
Release Notes: 1.64.2
Extension: Haskelly
home > drozda > haskell > 1.hs
1 fact:: Integer -> Integer
2 fact 0 = 1
3 fact n = n * fact (n - 1)
4
5 main:: IO ()
6 main = do
7 let a = fact 1000
8 print a
9 return ()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Haskell Run
stack runhaskell "/home/drozda/haskell/1.hs"
drozda@singapore:~$ stack runhaskell "/home/drozda/haskell/1.hs"
drozda@singapore:~$
Load GHCi Run File QuickCheck
Ln 9, Col 14 Spaces: 4 UTF-8 LF Haskell