

# FLP 2022/2023 – funkcionální projekt: Haskell

Daniel Uhříček  
iuhricek@fit.vut.cz

## Úvod

Toto je zadání funkcionálního projektu do předmětu Funkcionální a logické programování 2022/2023. Za projekt zodpovídá Ing. Daniel Uhříček, konzultace jsou možné kdykoliv po předchozí domluvě.

## Obecné požadavky

Svá řešení odevzdávejte v předepsaném tvaru (zkomprimovaný archiv zip, viz Závazné pokyny pro projekty) prostřednictvím VUT IS. Odevzdaný projekt musí obsahovat zdrojové texty v Haskellu. V hlavní složce musí být soubor `Makefile`, který program přeloží a výsledný binární kód umístí také do hlavní složky. Dále se doporučuje přiložit stručný popis všeho, co nebylo dořešeno nebo naopak bylo implementováno nad rámec zadání.

Například soubor `flp-fun-xlogin00.zip` po rozbalení obsahuje:

```
Makefile
src
├── Main.hs
├── Minimize.hs
├── ParseInput.hs
└── Types.hs
doc
├── README.md
└── test-description.txt
test
├── test01.in
├── test01.out
├── test02.in
└── test02.out
```

Pro překlad použijte kompilátor `ghc` s volbou `-Wall`. Cílový program pojmenujte **flp22-fun**. Můžete využít standardní knihovny z balíku `base`, případně knihovny z balíků `containers`, `parsec`, `vector`, `split`, `directory`, a `random`. Za referenční verzi Haskellu je považována verze 7.6.3 (server merlin) nebo verze 9.2.5. Před odevzdáním si ověřte, zda lze zip rozbalit a program přeložit a spustit, například:

```
unzip flp-fun-xlogin00.zip
make
./flp22-fun -2 test.in
```

## Hodnocení

Za vypracovaný projekt lze získat až 12 bod. Hodnocena bude míra splnění zadání, kvalita řešení, čistota a kvalita kódu. Za obzvláště kvalitní řešení lze získat prémiové body navíc. Vyvarujte se nestandardních funkcí, které obcházejí bezpečné otypování nebo obcházejí zapouzdření vedlejších efektů – nepoužívejte nic, co má ve svém jméně slovo `unsafe`. Snažte se psát čistý kód respektující různé varování a připomínky, které zazněly na přednáškách a cvičeních. Prosím také o dodržení, ať: úvodní řádky zdrojových kódů obsahují název projektu, login, jméno autora a rok řešení; součástí definic na globální úrovni jsou typové anotace a stručný a výstižný komentář; je zřejmé, co a z jakých modulů importujete<sup>1</sup>. Projekt vypracovávejte samostatně.

Následuje popis jednotlivých zadání.

---

<sup>1</sup>[https://wiki.haskell.org/Import\\_modules\\_properly](https://wiki.haskell.org/Import_modules_properly)

# 1 ECDSA

Vytvořte program, který implementuje ECDSA a umožňuje pro zvolenou eliptickou křivku (EC) vygenerovat pár klíčů, podepsat hash zprávy a následný podpis zkontrolovat.

## 1.1 Rozhraní programu

Program bude možné spustit:

```
flp22-fun volby [vstup]
```

kde

- *vstup* je jméno vstupního souboru (pokud není specifikován – stdin)
- *volby* jsou parametry ovlivňující chování programu:
  - i ze vstupu načte informace o EC do vaší vnitřní reprezentace. Na stdout jí vypíše zpět (očekává se, že tento výpis bude řešen instancí typové třídy `Show` pro váš datový typ reprezentující EC).
  - k ze vstupu načte informace o EC. Na stdout vypíše pár nově vygenerovaných klíčů ( $d, Q$ ).
  - s ze vstupu načte informace o EC společně s private klíčem a hashem zprávy, která má být podepsána. Na stdout vypíše vypíše vygenerovaný podpis ( $r, s$ ).
  - v ze vstupu načte informace o EC společně s public klíčem a hashem zprávy, jejíž podpis má být ověřen. Na stdout vypíše `True` pokud podpis je v pořádku, `False` pokud v pořádku není.

## 1.2 Očekávaný vstup a výstup

V následujících ukázkách je <informace o EC> zaměněno za:

```
Curve {  
  p: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F  
  a: 0  
  b: 7  
  g: Point {  
    x: 0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798  
    y: 0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8  
  }  
  n: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141  
  h: 1  
}
```

### 1.2.1 -i

Vstup:

<informace o EC>

Výstup:

<informace o EC>

### 1.2.2 -k

Vstup:

<informace o EC>

Výstup<sup>2</sup>:

```
Key {  
  d: 0xc9dcda39c4d7ab9d854484dbed2963da9c0cf3c6e9333528b4422ef00dd0b28e  
  Q: 0x040e411e56210f20bf5172cbefab02135421b1eb58f6918d28c1b848be5eee42...  
}
```

---

<sup>2</sup>Veřejný klíč Q začínající 04 hexa je v SEC uncompressed formátu, který začíná 04, následuje n bytů souřadnice X a n bytů souřadnice Y. Viz <https://secg.org/sec1-v2.pdf#subsection.2.3.3> (body 3.1, 3.2, 3.3).

### 1.2.3 s

Vstup:

<informace o EC>

Key {

d: 0xc9dcda39c4d7ab9d854484dbed2963da9c0cf3c6e9333528b4422ef00dd0b28e

Q: 0x040e411e56210f20bf5172cbefab02135421b1eb58f6918d28c1b848be5eee42...

}

Hash: 0x94996fead5b722c3bd07360c459927976e804f869626f4897def03fa56b009e3

Výstup:

Signature {

r: 0xb21ff64650be40610ba9324bc6bd273eafa87ac1bbc075be425c0f422f53196f

s: 0x3b4d090468eddbea8a53c565d19a24c56377786c49a7f114459c43bc7d59615a

}

### 1.2.4 -v

Vstup:

<informace o EC>

Signature {

r: 0xb21ff64650be40610ba9324bc6bd273eafa87ac1bbc075be425c0f422f53196f

s: 0x3b4d090468eddbea8a53c565d19a24c56377786c49a7f114459c43bc7d59615a

}

PublicKey {

Q: 0x040e411e56210f20bf5172cbefab02135421b1eb58f6918d28c1b848be5eee42...

}

Hash: 0x94996fead5b722c3bd07360c459927976e804f869626f4897def03fa56b009e3

Výstup:

True

## 2 Knapsack problem

Vytvořte program, který řeší optimalizační verzi 0-1 problému batohu (knapsack problem).

### 2.1 Rozhraní programu

Program bude možné spustit:

```
flp22-fun volby [vstup]
```

kde

- *vstup* je jméno vstupního souboru (pokud není specifikován – stdin)
- *volby* jsou parametry ovlivňující chování programu:
  - i ze vstupu načte informace o instanci knapsack do vaší vnitřní reprezentace. Na stdout jí vypíše zpět (očekává se, že tento výpis bude řešen instancí typové třídy **Show** pro váš datový typ reprezentující knapsack).
  - b ze vstupu načte informace o knapsack instanci. Na stdout vypíše řešení nalezené prohledáváním stavového prostoru hrubou silou. V případě, že řešení nebylo nalezeno, vypíše **False**.
  - o ze vstupu načte informace o knapsack instanci. Na stdout vypíše řešení nalezené některou z následujících optimalizačních metod: (1) genetic algorithm, (2) simulated annealing, (3) ant colony optimization, nebo (4) particle swarm optimization. Algoritmy by mimo jiné měly být známé z předmětů IZU, SFC, EVO, a SUI. V případě, že řešení nebylo nalezeno, vypíše **False**.

### 2.2 Očekávaný vstup a výstup

V následujících ukázkách je <informace o knapsack instanci> zaměněno za:

```
Knapsack {  
  maxWeight: 46  
  minCost: 324  
  items: [  
    Item {  
      weight: 36
```

```

    cost: 3
  }
  Item {
    weight: 43
    cost: 1129
  }
  Item {
    weight: 202
    cost: 94
  }
  Item {
    weight: 149
    cost: 2084
  }
]
}
```

### 2.2.1 -i

Vstup:

<informace o knapsack instanci>

Výstup:

<informace o knapsack instanci>

### 2.2.2 -b

Vstup:

<informace o knapsack instanci>

Výstup:

Solution [0 1 0 0]

### 2.2.3 -o

Stejný formát jako v případě -b.

### 3 Vlastní zadání

V případě zájmu o samostatné zadání dle vašich preferencí a specializace prosím kontaktujte cvičícího. Na základě dohody lze vypsát prakticky libovolné zadání na míru. Iniciativa s vlastním zadáním je velmi vítána. Projekt může být součástí vaší diplomové práce, projektu do jiného předmětu, zaměstnání nebo zájmu. Jedinou podmínkou je, že musí být vypracován v jazyce Haskell.

Jelikož však nebude možné takové zadání automaticky testovat spolu s ostatními, očekávejte své zapojení při prokazování funkčnosti řešení. Konkrétní forma bude záviset na dohodnutém tématu, ale může jít o osobní předvedení nebo vypracování a odevzdání sady automatizovaných testů.