IB015 – Domácí úkol 8: Databáze dopravy

Termín: do 6. 11. 23.59; 5 pokusů odevzdání; způsob odevzdání je podrobně popsán níže.

V prvním velkém domácím úkolu budeme pracovat s databází dopravy, pro kterou si naprogramujeme několik užitečných funkcí.

Databáze bude obsahovat zjednodušený model přepravy dopravními prostředky jako autobus či vlak z místa A do místa B. Zároveň v databázi budeme ukládat zakoupené jízdenky na jednotlivé spoje.

Kostru zadání si můžete stáhnout z ISu.

Reprezentace databáze

Pro snazší orientaci v kódu si na začátku zavedeme několik typových aliasů. V této úloze budeme používat odpovídající alias vždy, když bude typ významově odpovídat názvu aliasu.

```
type TripId = Integer
type Place = String
type Price = Integer
type TicketId = Integer
type PersonId = Integer
```

Dále si definujeme reprezentaci jízdenek a spojů.

```
type Ticket = (TicketId, PersonId, TripId, Tarif)
data Tarif = Adult | Child | Student | Senior
```

Jízdenka je typový alias pro čtveřici, kde jednotlivé složky jsou v tomto pořadí: jednoznačné ID zakoupené jízdenky, jednoznačné ID osoby, která si jízdenku zakoupila, jednoznačné ID spoje a jako poslední složka jízdenky je `Tarif`, který je definovaný vlastním datovým typem.

```
type Trip = (TripId, Transport, Place, Place, Price)
data Transport = Plane | Train | Bus | Ship
```

Spoj je definován pěticí, která je po složkách: jednoznačné ID spoje, vlastní datový typ `Transport` (který reprezentuje typ dopravního prostředku), místo odkud spoj jede, místo kam spoj jede a nakonec cena daného spoje.

ID jsou vždy jednoznačná pro danou entitu v rámci celé databáze. Nemůže tedy například existovat více jízdenek se stejným ID. Může se ale klidně stát, že existuje jak jízdenka s ID 42, tak spoj s ID 42 a tyto věci spolu nemusí nijak souviset. U osob máme pouze číselný identifikátor, další informace o nich nemáme.

Konečně databáze je jednoduchým aliasem pro dvojici. První složkou této dvojice je seznam zakoupených jízdenek a druhou složkou je seznam spojů.

```
type Database = ([Ticket], [Trip])
```

Funkce k implementaci

Vaším úkolem je naprogramovat následující sadu funkcí.

Nejprve budeme chtít implementovat několik tzv. *getterů*, což jsou funkce typicky nad nějakou datovou strukturou / datovým typem, které zpřístupňují jednotlivé (pod)složky dané struktury. V Haskellu getter zpravidla používáme, pokud k nějaké složce přistupujeme často, proto, abychom zbytečně neduplikovali kód a zároveň jej učinili čitelnějším.

Všimněte si, že z názvu funkcí a jejich typové signatury je obzvláště zřejmé, co funkce dělá. Popíšeme tak pouze funkcionalitu prvního getteru, zbytek naprogramujte podle typové signatury

Poznámka: Tyto gettery mohou přijít vhod při implementaci ostatních funkcí.

- `getTripId :: Trip -> TripId`

  Tato funkce dostane jako argument spoj a vrátí jeho ID.

- `getTransport :: Trip -> Transport`
- `getPlaceFrom :: Trip -> Place`
- `getPlaceTo :: Trip -> Place`
- `getPrice :: Trip -> Price`
- `getTicketId :: Ticket -> TicketId`
- `getPersonId :: Ticket -> PersonId`
- `getTicketTripId :: Ticket -> TripId`
- `getTarif :: Ticket -> Tarif`

Dále chceme naprogramovat funkce s lehce sofistikovanější funkcionalitou.

- `lookupTarif :: TicketId -> [Ticket] -> Tarif`

  Funkce vyhledá jízdenku podle ID v zadaném seznamu jízdenek a vrátí její tarif.

  Můžete předpokládat, že jízdenka se zadaným ID v databázi existuje (právě jedna[1]).

- `tripsFrom :: Place -> [Trip] -> [Trip]`
- `tripsTo :: Place -> [Trip] -> [Trip]`

  Dvojice funkcí, která na vstupu dostane název místa a seznam spojů. Funkce vrací seznam spojů, které ze zadaného místa vyráží, respektive do zadaného místa přijíždí.

  Pořadí spojů ve výsledném seznamu musí odpovídat pořadí spojů ve vstupním seznamu.

  Poznámka: zamyslete se nad tím, jak funkce napsat bez duplikace kódu.

- `transports :: Place -> Place -> [Trip] -> [Transport]`

  Výstupem funkce je seznam všech dopravních prostředků, které jezdí přímo (bez přestupů) z místa zadaného v prvním argumentu do místa zadaného ve druhém argumentu.

  Na výsledném pořadí dopravních prostředků nezáleží, ale prvky seznamu se nesmí opakovat.

  Poznámka: Funkce na odstranění duplikátů existuje a to ***`nub`*** *z modulu* *`Data.List`*

- `updatePrice :: TripId -> Price -> [Trip] -> [Trip]`

  Tato funkce aktualizuje cenu (na hodnotu zadanou druhým argumentem) spoje, jehož ID je shodné s prvním argumentem funkce.

  Ostatní spoje se nesmí nijak změnit, žádné spoje nepřidávejte ani nemažte a pořadí spojů musí zůstat zachováno.

- `tripCountByTransport :: Transport -> [Trip] -> Int`

  Výsledkem funkce je počet spojů, které jsou realizovány zadaným dopravním prostředkem.

- `travelsBy :: PersonId -> TripId -> [Ticket] -> Bool`

  Tato funkce vrátí `True`, pokud osoba s ID zadaným v prvním argumentu cestovala alespoň jednou spojem s ID zadaným v druhém argumentu.

- `peopleByTarif :: Tarif -> [Ticket] -> [PersonId]`

  Výsledkem je seznam ID osob, které alespoň jednou cestovaly zadaným tarifem (mohly cestovat i jiným).

  Výstupní seznam nesmí obsahovat duplikáty a na pořadí prvků nezáleží.

- `travellersFrom :: Place -> Database -> [PersonId]`

  Stejně jako u předchozí funkce je výsledkem seznam ID osob, tentokrát takových, které alespoň jednou cestovaly ze zadaného místa.

[1] Jedinečnost plyne z jednoznačnosti ID, dále již na ni nebudeme upozorňovat.

Opět platí, že výstupní seznam nesmí obsahovat duplikáty a na pořadí nezáleží.

- `ticketPrice :: (Tarif -> Price -> Price) -> TripId -> Tarif -> [Trip] -> Price`

  Tato funkce slouží k výpočtu ceny spojení. Jejím prvním argumentem je funkce, která na základě tarifu a základní ceny spočítá výslednou cenu jízdenky. Druhý argument je ID spoje, který nás zajímá, a třetím argumentem je tarif, který chceme využít. Poslední argument je seznam spojů. Výsledkem funkce `ticketPrice` je cena, kterou člověk zaplatí za zvolený spoj při daném tarifu.

  Můžete předpokládat, že zadané ID spoje v seznamu existuje.

```
ticketPrice (\_ p -> p) 42 Student
  [(0, Plane, "Brno", "Praha", 1000), (42, Bus, "Brno", "Praha", 50)] ~>* 50

freeChild :: Tarif -> Price -> Price
freeChild Child _ = 0
freeChild _     p = p

ticketPrice freeChild 42 Student
  [(0, Plane, "Brno", "Praha", 1000), (42, Bus, "Brno", "Praha", 50)] ~>* 50
ticketPrice freeChild 42 Child
  [(0, Plane, "Brno", "Praha", 1000), (42, Bus, "Brno", "Praha", 50)] ~>* 0
```

- `spent :: (Tarif -> Price -> Price) -> PersonId -> Database -> Price`

  Výstupem této funkce je opět cena, ale tentokrát taková, kterou osoba, jejíž ID je zadané v druhém argumentu, zaplatila za všechny své jízdenky uložené v databázi. Význam prvního argumentu je totožný jako u `ticketPrice`.

  V případě, že si zadaná osoba nezakoupila ani jednu jízdenku, je výsledkem 0.

```
spent freeChild 42
  ([(0, 42, 0, Student), (1, 42, 42, Child), (42, 5, 0, Adult)],
  [(0, Plane, "Brno", "Praha", 1000), (42, Bus, "Brno", "Praha", 50)])
    ~>* 1000
spent freeChild 5
  ([(0, 42, 0, Student), (1, 42, 42, Child), (42, 5, 0, Adult)],
  [(0, Plane, "Brno", "Praha", 1000), (42, Bus, "Brno", "Praha", 50)])
    ~>* 1000
spent freeChild 21
  ([(0, 42, 0, Student), (1, 42, 42, Child), (42, 5, 0, Adult)],
  [(0, Plane, "Brno", "Praha", 1000), (42, Bus, "Brno", "Praha", 50)])
    ~>* 0
```

Poznámky a tipy

- Importovat smíte moduly z balíku `base`.[2]
- Neduplikujte kód! Snažte se vždy využít funkce, které jste již naprogramovali. Pokud to nejde přímo, ale přesto vidíte v řešení podobu, vytkněte podobnou část do pomocné funkce.
- Nevynalézejte znovu kolo! Snažte se využívat knihovních funkcí, především nemá smysl je zbytečně reimplementovat. Důrazně doporučujeme použít vyhledávač Hoogle ve fakultní verzi, která automaticky vyhledává pouze v `base`. Pozornost věnujte především modulu `Data.List`, najdete v něm například užitečnou funkci `nub`.
- Pomocné funkce definujte lokálně, pokud jsou využívány jedinou funkcí.
- Funkce jsou v kostře zadefinovány jako `undefined`, takže projdou překladem, ale jejich zavolání způsobí chybu.
- Nejste-li si jisti nějakou částí zadání, zeptejte se v diskusním fóru.
- Nezapomeňte, že **opisování je zakázáno** a bude postihováno podle disciplinárního řádu.
- Přebíráte-li kód odjinud, uveďte zdroj, jinak bude na vaši práci pohlíženo jako na plagiát.
- Než řešení odevzdáte, **pečlivě si přečtěte následující sekci** a ujistěte se, že váš kód splňuje všechny náležitosti. Neztrácejte body jen kvůli nepozornému čtení pokynů.

[2] Výjimku tvoří moduly, které jsou „Unsafe“, ty však určitě nebudete potřebovat.

Odevzdání

Tento domácí úkol se neodevzdává přes odpovědník, nýbrž přes příslušnou odevzdávárnu v Informačním systému.

- Do odevzdávárny vkládejte **jediný** soubor s příponou `.hs` obsahující vaši implementaci **všech** požadovaných funkcí.
  - Pokud chcete odevzdat znovu, můžete soubor přepsat či přidat nový, nezáleží na tom – vyhodnocuje se vždy nejnovější odevzdaný soubor.
  - Žádné jiné soubory nevkládejte.
  - Již vložené soubory nepřejmenovávejte, mohou se pak vyhodnotit dvakrát.
- Vaše řešení **musí zachovat všechny definice datových typů tak, jak jsou v zadání**, jediná povolená modifikace je přidání instance typové třídy.
- Řešení musí jít přeložit překladačem GHC 9.2.1. Je možné, že na svých počítačích máte starší verzi, což by nemělo vadit, jde-li o verzi 8.4 a vyšší. I přesto vám doporučujeme, abyste si před odevzdáním svůj kód zkusili zkompilovat a spustit na Aise (nezapomeňte přidat modul s novým GHC), kde máte k dispozici GHC ve verzi 9.2.1. V případě selhání testů už při kompilaci nepřijdete o žádný pokus.
- **Všechny globální funkce musí mít typovou signaturu**.
- V odevzdaném souboru neuvádějte hlavičku `module` (pokud nevíte, o co se jedná, vůbec to nevadí).

Vyhodnocování

- Vyhodnocení po nahrání souboru **není** okamžité.
  - Automatický testovací nástroj kontroluje soubory v odevzdávárně v pravidelných intervalech několika minut. Podle času odevzdání a vytížení vyhodnocovacího serveru může vyhodnocení trvat několik desítek minut.
  - Pokud se vám výsledky neobjevily cca do půl hodiny a neblíží se zatím čas deadline, dejte nám vědět ve fóru.
- Po vyhodnocení se získané body a případný výpis testů, které na vašem řešení selhaly, **objeví v poznámkovém bloku**.
  - U nesprávně implementovaných funkcí se dozvíte příklad vstupu, na němž se váš výsledek neshoduje s očekávaným.
- Máte **pět možností odevzdání**, započítává se nejlepší z nich.
  - Pokud při odevzdání neprojde kontrola syntaxe, tak se do tohoto limitu nepočítá. Pokud však kontrola projde, je automaticky započítáno a nelze to zvrátit.
- Další odevzdání provedete tak, že do odevzdávárny nahrajete novou verzi.
- Vzhledem k prodlevám při vyhodnocování neodkládejte práci na poslední chvíli, ať možnost vícenásobného odevzdání v případě potřeby vůbec stihnete využít.
  - S blížícím se termínem uzavření odevzdáváren očekávejte větší (i několikahodinové) prodlevy.
  - Není žádná garance rychlosti vyhodnocování (může se tedy stát, že výsledky řešení odevzdaného v neděli ve 21.00 neuvidíte do konce deadline).
  - Na jakákoli odevzdání po termínu nebude brán zřetel.

S odevzdávárnou zacházejte s rozvahou, abyste nepřišli o možnosti odevzdání. I když nahrajete nové řešení ještě před zveřejněním výsledku v poznámkovém bloku, vyhodnocovací nástroj už může mít (a pravděpodobně má) vaše dřívější odevzdání ve frontě. Z jeho pohledu tak došlo ke dvěma odevzdáním a vy si vyplýtváte jeden pokus. Podobně se vám mohou započítat odevzdání navíc, pokud do odevzdávárny omylem vložíte více než jeden soubor nebo pokud soubor přejmenujete (každý soubor se vezme jako samostatné odevzdání).

Hodnocení

Za funkčnost můžete od automatických testů obdržet **až 2,5 bodu** podle toho, které funkce (nebo jejich části) se vám podařilo správně implementovat (viz tabulka níže).

Funkce	Body
`gettery`	0.1
`lookupTarif`	0.1
`tripsFrom`	0.1

Funkce	Body
`tripsTo`	0.1
`transports`	0.1
`updatePrice`	0.2
`tripCountByTransport`	0.2
`travelsBy`	0.2
`peopleByTarif`	0.2
`travellersFrom`	0.2
`ticketPrice`	0.5
`spent`	0.5

Jelikož jde pravděpodobně o vaše první setkání s tvorbou delšího kódu v Haskellu, doporučujeme vám věnovat pozornost tipům k psaní hezkého kódu, které naleznete ve sbírce. Výhodou hezkého kódu je, že většinou obsahuje méně chyb a pokud už nějaké obsahuje, snáze se hledají.