IB015 – Domácí úkol 12: Lodě

Termín: 30. prosince 2019 23.59; způsob odevzdání je popsán níže.

V posledním domácím úkolu si konečně v některých funkcích vyzkoušíte práci se vstupem a výstupem. Odměnou vám za to mohou být opět až dva body do hodnocení.

Tentokrát budete střílet a potápět. Vaším úkolem bude naprogramovat zjednodušenou verzi hry lodě hratelnou v terminálu. Jak bylo řečeno, procvičíte si `IO`, nicméně vhod přijde i spousta dalších věcí, které jste viděli během semestru. Určitě se budou hodit vzory, práce s vlastními datovými typy a naše staré známé funkce `map` a `filter`.

Kostru domácí úlohy si můžete stáhnout ze studijních materiálů.

Použité datové typy

Souřadnice

Pro souřadnice používáme typový alias `type Coord = (Int, Int)`. Použití aliasu nám jednak zkrášluje typ funkce, zadruhé nám také lépe vyjadřuje to, co dané hodnoty reprezentují. Jelikož jde ale pouze o typový alias, můžete oba typy zaměňovat tak, jak to již znáte u dvojice `[Char]` a `String`.

Lodě

Pro reprezentaci lodě máme vlastní datový typ `data Ship = Ship [(Coord, Status)]`. Seznam v argumentu hodnotového konstruktoru obsahuje všechny souřadnice, přes které se loď rozprostírá. Ke každé souřadnici (první složka dvojice) se váže její aktuální stav (druhá složka dvojice). Stav reprezentujeme vlastním datovým typem `Status`, kde `AsNew` značí, že je loď na dané pozici v perfektní kondici, naopak `Damaged` vyjadřuje, že na této souřadnici už je práce u konce.

Nepoškozená loď je tedy taková, která má všechny druhé složky v seznamu dvojic rovny `AsNew`. Naproti tomu u potopené lodi jsou všechny tyto hodnoty `Damaged`. Poškozenou lodí rozumíme takovou loď, která ještě není potopená a zároveň už není nepoškozená.

```
Ship [((1, 1), AsNew), ((1, 2), AsNew)]       -- nepoškozená loď
Ship [((1, 1), Damaged), ((1, 2), AsNew)]     -- poškozená loď
Ship [((1, 1), Damaged), ((1, 2), Damaged)]   -- potopená loď
```

Herní plán

Herní plán je čtvercový tak, jak jsme u moří a oceánů zvyklí. K jeho reprezentaci slouží vlastní datový typ `data ShipsPlan = ShipPlan Int [Ship]`, kde první parametr odpovídá jeho velikosti a druhý seznamu lodí, které se nachází v herním plánu. Pozice v plánu indexujeme od 1 po n včetně, kde n je jeho velikost. Horní levý roh odpovídá pozici `(1, 1)`, levý dolní roh pozici `(1, n)` a pravý dolní roh pak pozici `(n, n)`.

Prázdný herní plán je takový plán, který neobsahuje žádné lodě.

Validní herní plán je takový, který má kladnou velikost a zároveň v něm neexistuje loď, která by z něj zasahovala ven, nebo dvě lodě, které by spolu kolidovaly (tj. lodě, jež sdílí alespoň jedno políčko – pozici). Ve validním herním plánu také nenajdete potopenou loď, ani loď bez souřadnic – tj. `Ship []`

U všech funkcí můžete předpokládat, že vstupem bude validní herní plán. Naopak žádná vaše funkce vracející herní plán jej nesmí vrátit nevalidní.

```
ShipPlan 10 []                          -- validní a zároveň prázdný herní plán
ShipPlan 10 [Ship [((2, 4), AsNew)]]    -- validní neprázdný herní plán
ShipPlan (-42) []                       -- nevalidní herní plán (velikost je menší než 1)
ShipPlan 10 [Ship [((15, 40), AsNew)]] -- nevalidní herní plán (loď je vně)
ShipPlan 10 [Ship [((1, 2), AsNew), ((1, 3), Damaged)]]      -- validní herní plán
ShipPlan 10 [Ship [((1, 2), AsNew)], Ship [((2, 2), AsNew)]] -- validní herní plán
ShipPlan 10 [Ship [((1, 2), AsNew)], Ship [((1, 2), AsNew)]] -- nevalidní (konflikt lodí)
ShipPlan 10 [Ship [((1, 2), Damaged), ((1, 3), Damaged)]]    -- nevalidní (potopená loď)
```

Orientace a střelba

V kostře pak ještě naleznete dva datové typy; jeden pro reprezentaci orientace lodě, druhý pro výsledky střelby na lodě. K obojímu se ještě vrátíme, prozatím nám bude stačit, že u orientace `data ShipOrientation` rozlišujeme orientaci horizontální (vodorovnou) – `Horizontal` – a vertikální (svislou) – `Vertical`.

U střelby reprezentované datovým typem `data ShotResult` máme pro změnu stavy tři: `Ocean` pro střelbu bez zásahu cíle, `Hit` pro střelbu, která nějakou loď zasáhla, a nakonec `Sunk` pro střelbu, která loď zasáhla a zároveň potopila (tj. stav všech souřadnic je `Damaged`).

Funkce k implementaci

Na vás bude implementovat následujících šest funkcí.

- **`isEmpty :: ShipsPlan -> Bool`**

  Funkce, která dělá přesně to, co říká. Na vstupu dostane herní plán a vrátí `True` právě tehdy, když je plán prázdný.

```
> isEmpty (ShipPlan 10 [Ship [((2, 3), AsNew), ((2, 4), AsNew)]])
False
> isEmpty (ShipPlan 10 [])
True
```

- **`toShip :: Coord -> ShipOrientation -> ShipSize -> Ship`**

  Tato funkce dostane na vstup vše, co potřebuje k tvorbě lodi. Vaším úkolem je z tohoto vstupu poskládat loď a tu vrátit.

  Loď vytvoříte tak, že pro zadané souřadnice (x, y), které reprezentují políčko lodě nejblíže k levému hornímu rohu, vytvoříte vzestupně seřazený seznam sousedních souřadnic délky `ShipSize` odpovídající souřadnicím, na kterých se má loď nacházet. Lodě vytvořené touto funkcí musí být nepoškozené. Můžete předpokládat, že v testech bude `ShipSize` vždy kladná (toto ovšem platí pouze pro testy funkce `toShip`, při testování celé hry si toto musíte zajistit vhodným zpracováním vstupu).

```
> toShip (1, 2) Horizontal 1
Ship [((1, 2), AsNew)]
> toShip (3, 4) Vertical 4
Ship [((3, 4), AsNew), ((3, 5), AsNew), ((3, 6), AsNew), ((3, 7), AsNew)]
> toShip (3, 4) Horizontal 4
Ship [((3, 4), AsNew), ((4, 4), AsNew), ((5, 4), AsNew), ((6, 4), AsNew)]
> toShip (-2, 0) Vertical 2
Ship [((-2, 0), AsNew), ((-2, 1), AsNew)]
```

  Povšimněte si, že funkce produkuje i lodě, které se nemohou ve validním herním plánu objevit.

- **`placeShip :: Coord -> ShipOrientation -> ShipSize -> ShipsPlan -> Maybe ShipsPlan`**

  Dostanete na vstup všechna data, která potřebujete k tvorbě lodi, a k tomu herní plán, do kterého tuto nově utvořenou loď vložíte, pokud je to možné (tj. pokud by nevznikl nevalidní herní plán) – vložte ji na začátek seznamu lodí. Tento herní plán vrátíte zabalený v `Maybe`. Pokud loď odpovídající vstupním argumentům do plánu vložit nelze, vraťte z funkce hodnotu `Nothing`.

```
> placeShip (2, 3) Horizontal 2 (ShipPlan 10 [])
Just (ShipPlan 10 [Ship [((2, 3), AsNew), ((3, 3), AsNew)]])
> placeShip (2, 3) Vertical 1 (ShipPlan 10 [])
Just (ShipPlan 10 [Ship [((2, 3), AsNew)]])
> placeShip (2, 3) Vertical 10 (ShipPlan 10 [])
Nothing
> placeShip (2, 3) Vertical 0 (ShipPlan 10 [])
Nothing
> placeShip (20, 30) Vertical 5 (ShipPlan 10 [])
```

```
Nothing
> placeShip (2, 3) Horizontal 2 (ShipPlan 10 [Ship [((3, 2), AsNew), ((3, 3), AsNew)]])
Nothing
```

- `shoot :: Coord -> ShipsPlan -> (ShipsPlan, ShotResult)`

  Uskuteční výstřel na zadané souřadnice. V případě, že střelba zasáhne nějakou loď, je třeba tuto skutečnost v herním plánu reflektovat. Pokud je loď zasažena, musíte změnit stav dané souřadnice odpovídající lodě, případně pokud zásah loď potopí, tak tuto zničenou loď odeberte z herního plánu. Návratovou hodnotou pak bude nový plán a výsledek střelby `Hit` v prvním případě, respektive `Sunk` v případě druhém. V případě, že střelba zasáhne již poškozené políčko, berte to jako regulérní zásah s tím, že stav políčka už měnit nemusíte – z předchozího zásahu je již nastaven na `Damaged`, takže stačí vrátit vstupní plán a `Hit`.

  Pokud střelba mine všechny lodě nebo bude směřovat mimo herní plán, zasáhne oceán. V takové situaci stačí vrátit plán tak, jak jste jej dostali, s hodnotou `Ocean` jako výsledkem střelby.

  ```
  > shoot (2, 2) (ShipPlan 3 [Ship [((2, 2), AsNew), ((2, 3), AsNew)]])
  (ShipPlan 3 [Ship [((2, 2), Damaged), ((2, 3), AsNew)]], Hit)
  > shoot (2,3) (ShipPlan 3 [Ship [((2, 2), Damaged), ((2, 3), AsNew)]])
  (ShipPlan 3 [], Sunk)
  > shoot (8, 9) (ShipPlan 10 [])
  (ShipPlan 10 [], Ocean)
  > shoot (10, 15) (ShipPlan 5 [])
  (ShipPlan 5 [], Ocean)
  > shoot (2, 5) (ShipPlan 10 [Ship [((3, 3), AsNew), ((3, 4), Damaged)]])
  (ShipPlan 10 [Ship [((3, 3), AsNew), ((3, 4), Damaged)]], Ocean)
  ```

- `printPlan :: ShipsPlan -> IO ()`

  Vypíše daný herní plán. Nevypisujte nic více než daný plán (tedy ani žádnou nápovědu, popis souřadnic apod.). Pro políčka používáme 3 různé znaky dle toho, co se na nich nachází; `~` pro oceán, `#` pro nezasažené políčko lodě, `X` pro zasažené. Plán se vypisuje po řádcích od nejnižšího indexu po nejvyší, tj. pro plán velikosti dva vypíšeme pozice (1, 1), (2, 1) na jeden řádek a pozice (1, 2), (2, 2) na řádek druhý. Výpis bude opět ukončen novým řádkem.

  ```
  > printPlan (ShipPlan 3 [])
  ~~~
  ~~~
  ~~~
  > printPlan (ShipPlan 3 [(Ship [((2, 2), AsNew), ((2, 3), AsNew)])])
  ~~~
  ~#~
  ~#~
  > printPlan (ShipPlan 3 [(Ship [((2, 2), Damaged), ((2, 3), AsNew)])])
  ~~~
  ~X~
  ~#~
  ```

- `game :: IO ()`

  Funkce, jejímž zavoláním začíná hra. Hra poté běží, dokud ji uživatel neukončí, nebo dokud nepotopí všechny lodě.

Průběh hry

Průběh hry můžeme rozdělit do několika fází.

Fáze 1 – *tvorba plánu*

Fáze jedna nastává bezprostředně po spuštění hry, tj. voláním funkce `game`. Následně je uživatel vyzván k zadání velikosti herního plánu a toto číslo je od něj načteno s celým řádkem. Zde byste také měli vstupu odpovídající herní plán vytvořit. Následuje fáze 2.

V případě, že by vstup vedl k vytvoření nevalidního plánu, uživatele informujte jednořádkovou zprávou a proceduru opakujte.

Fáze 2 – *stavba lodí*

Teď je na čase do plánu přidat nějaké terče – lodě. Hráč bude vyzván k přidání nové lodě. Tu může přidat zadáním souřadnic (dvě čísla oddělena alespoň jednou mezerou) následovaných další alespoň jednou mezerou a znakem `H` či `V` značícím orientaci (`H` pro horizontální, `V` pro vertikální) a nakonec číslem značícím velikost (délku) lodě. Po každém úspěšném zadání lodě do herního plánu herní plán vypište na obrazovku. V opačném případě informujte uživatele jednořádkovou zprávou.

```
4 5 H 10        -- validní vstup
42   15    V 5  -- validní vstup
4,3 V 10        -- nevalidní vstup
```

Ke zpracování tohoto vstupu můžete využít v kostře již implementovanou funkci `parseShipInput` (vizte níže). Hráč přidává lodě, dokud druhou fázi neukončí zadáním řetězce "end", což hru přesune do fáze 3. Ještě předtím ale vypište herní plán na obrazovku.

Pokud zadaným řetězcem není zmíněný ***`end`*** *a funkce* ***`parseShipInput`*** *(případně vaše vlastní funkce pro zpracování tohoto vstupu) selže, informujte o tom uživatele libovolnou jednořádkovou zprávou a vyzvěte jej znovu k přidání nové lodě.*

Fáze 3 – *střelba*

Nyní dojde na slibované potápění. Hráč je vyzván k zadání dvojice čísel oddělených alespoň jednou mezerou. Tyto souřadnice určují políčko, na které si hráč přeje vystřelit. K načtení tohoto vstupu můžete využít v kostře implementovanou funkci `parseShootInput`. Poté musíte tento výstřel vyhodnotit a v případě, že došlo k zasažení lodě (případně jejímu potopení), musíte tuto skutečnost zohlednit v herním plánu. Nakonec hráče informujte o výsledku jeho střelby – na řádek vypište pouze `Hit`, `Ocean` nebo `Sunk`.

Třetí fáze končí ve chvíli, kdy jsou všechny lodě potopeny, nebo kdykoliv hráč namísto souřadnic zadá `end` pro ukončení hry. V obou případech o ukončení hry informujte hráče jednořádkovou zprávou.

V případě, že hráč zadá nevalidní vstup (nepůjde o dvě čísla oddělena mezerou ani o ukončovací příkaz end nebo zadané souřadnice nebudou validní), opět jej jednořádkovou zprávou o této skutečnosti informujte a znovu jej vyzvěte k zadání souřadnic pro střelbu.

Pomocné funkce

V kostře kromě datových typů a funkcí čekajících na to, až je implementujete, najdete také dvě pomocné funkce pro zpracování některých vstupů od uživatele.

- `parseShipInput :: String -> Maybe (Coord, ShipOrientation, ShipSize)`

  Tato funkce od vás očekává řetězec odpovídající vstupu pro zadání nové lodě tak, jak byl popsán ve fázi 2. V případě, že je vstup validní, funkce vám vrátí trojici načtených hodnot zabalenou v `Maybe`. V případě, že vstup je nějakým způsobem závadný, funkce vrátí hodnotu `Nothing` jako signál, že se zpracování vstupu nezdařilo.

- `parseShootInput :: String -> Maybe Coord`

  Funkce analogická k `parseShipInput`, tentokrát však pro zpracování vstupu uživatele pro střelbu, jak je popsán ve fázi 3.

Referenční implementace a kompilace

Na Aise naleznete spustitenou referenční implementaci, abyste si mohli vyzkoušet kýžené chování programu. Spustíte ji jednoduše tak, že na Aise do terminálu napíšete `/home/xchlup2/ib015/ships`. Tato binárka vychází z kostry úlohy, takže funkce `main` pouze zavolá funkci `game` – to znamená, že takto můžete zkoušet pouze hru jako celek, ne jednotlivé funkce.

Zároveň si můžete zkompilovat i své řešení do podoby spustitelné binárky; stačí v příkazové řádce provést příkaz `ghc reseni12.hs -o ships`, který vám vytvoří spustitelný binární soubor s názvem `ships`. Ten poté spustíte příkazem `./ships`. V kostře máte již definovánu funkci `main` jako `main = game`, tudíž spuštěním vaší binárky se automaticky spustí funkce `game`.

Poznámky a tipy

- Směle využívejte funkcí z `Prelude`, `Data.List` a `Data.Maybe`.
- Importovat smíte i jiné moduly z balíku `base`, pohodlně se bez nich ale obejdete.
- 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.
- Nevymýšlejte znovu kolo. Pokud řešíte nějakou základní funkcionalitu, která by již měla být někde implementována, nebojte se použít hoogle.
- U všech funkcí uvádějte jejich typové signatury.
- Funkce jsou v kostře zadefinovány jako `undefined`, takže projdou překladem, ale jejich zavolání způsobí chybu. Pokud nějakou funkci neimplementujete, ponechte ji jako `undefined` nebo ji zakomentujte či smažte.
- Nejste-li si jisti nějakou částí zadání, zeptejte se v diskusním fóru.
- Přebíráte-li kód odjinud, uveďte zdroj, jinak bude na vaši práci pohlíženo jako na plagiát.
- Nezapomeňte, že **opisování je zakázáno** a bude postihováno podle disciplinárního řádu.
- Než řešení odevzdáte, **pečlivě si přečtěte následující sekci** a ujistěte se, že váš kód splňuje všechny náležitosti. Neztrácejte body jen kvůli nepozornému čtení pokynů.

Odevzdání

Tento domácí úkol se neodevzdává přes odpovědník, nýbrž přes odevzdávárny v Informačním systému. O tom, kterou odevzdávárnu máte použít, rozhoduje číslo vaší seminární skupiny. Do odevzdávárny vkládejte **jediný** soubor s příponou `.hs` obsahující vaši implementaci požadovaných funkcí. Pokud odevzdaný soubor nepůjde přeložit překladačem GHC 8.6., testy selžou a vy nedostanete žádné body. Doporučujeme vám si jej proto před odevzdáním zkusit spustit na Aise (nezapomeňte přidat modul s novým GHC). **Všechny globálně definované funkce musí mít typovou signaturu**. V odevzdaném souboru neuvádějte hlavičku `module` (pokud nevíte, o co se jedná, vůbec to nevadí).

Vyhodnocení po nahrání souboru **není** okamžité; automatický testovací nástroj kontroluje soubory v odevzdávárně několikrát denně. Podle času odevzdání a vytížení vyhodnocovacího serveru může vyhodnocení trvat několik desítek minut. Po vyhodnocení se získané body a případný výpis neprošedších testů objeví v poznámkovém bloku. U nesprávně implementovaných funkcí se dozvíte příklad vstupu, na němž se váš výsledek neshoduje s očekávaným.

Od minulých úloh se testování dvanácté úlohy liší. Testy pro funkci `game` se tentokrát vůbec nespustí, pokud selžou testy některé z předchozích funkcí.

Máte **pět možností odevzdání**, započítává se nejlepší z nich. Další odevzdání provedete tak, že do odevzdávárny nahrajete novou verzi, jíž přepíšete odevzdaný soubor. 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.

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 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.

Pozor vzhledem k technickým limitacím odevzdáváren se váš soubor nevyhodnotí, pokud jej butete editovat přímo v ISu, prototo musíte nové řešení vždy vytvořit nahráním souboru, nikoliv editací.

Zároveň aby bylo možné vaše řešení otestovat, můžete používat jen omezenou množinu `IO` funkcí. Fungovat budou všechny funkce, které jsou zmíněny v části IO Replacements dokumentace naší vyhodnocovací služby, a všechny funkce, které v typu neobsahují konkrétně `IO`, ale něco z typových tříd `Monad` či `Applicative` (například `(>>=)`, `(>>)`, `return`, `mapM`, `sequence`).

Hodnocení

Za funkčnost můžete od automatických testů obdržet **až 1,8 bodu** podle toho, které funkce se vám podařilo správně implementovat. Za částečně implementované funkce (např. nefunguje některý okrajový případ) žádné body nezískáte. Nezapoměňte, že testy tentokrát **přeskočí** `game`, pokud selže některý z testů nějaké z předchozích funkcí. V takovém případě žádné body za funkci `game` nedostanete, proto odevzdávejte svá řešení s rozvahou.

Řešení budou po termínu odevzdávání hodnotit cvičící. Ti vám poskytnou zejména zpětnou vazbu na kód, ale také vám za úhlednost a pochopitelnost řešení mohou udělit další dvě desetiny bodu. Neočividné či zajímavé části řešení proto stručně komentujte v kódu. Tipy k psaní hezkého kódu naleznete v diskusním fóru.

V součtu tedy můžete za úlohu získat **až 2 body**.