Kurs języka Haskell

Lista zadań na pracownię nr 0

Na zajęcia 27 lutego i 3 marca 2020

Informacje dotyczące sposobu organizacji pracowni

- Niniejsza lista zadań nie jest punktowana i nie wymaga przygotowania przed zajęciami. Celem zajęć nr 0 jest zorganizowanie i przećwiczenie pracy na pracowni. Po jej zakończeniu każdy student powinien umieć:
  - edytować programy haskellowe, również w formie *Literate Haskell*,
  - kompilować je i uruchamiać w środowisku interaktywnym `ghci`,
  - wyszukiwać dokumentację języka, kompilatora i bibliotek standardowych,
  - sprawnie podłączać swój komputer do projektora w pracowni w celu zaprezentowania swojego rozwiązania.
- Na zajęciach (z wyjątkiem pierwszych) obowiązuje system deklaracji.
- Rozwiązania wszystkich zadeklarowanych zadań z listy należy zgłosić w systemie SKOS w postaci pojedynczego pliku o nazwie *`Imie`*`_`*`Nazwisko`*`_`*`nr`*`.lhs`, gdzie *`Imie`* i *`Nazwisko`* to, odpowiednio, imię i nazwisko studenta zgłaszającego rozwiązanie zapisane bez znaków diakrytycznych, a *`nr`* jest numerem listy zadań. Komentarze w pliku należy umieszczać w kodowaniu UTF-8. Na początku pliku powinno znajdować się imię i nazwisko studenta, nazwa przedmiotu, numer rozwiązywanej listy i data. Przed rozwiązaniem każdego zadania powinien znajdować się komentarz zawierający numer zadania, ewentualnie jego treść i komentarze do rozwiązania. Zgłoszony plik powinien się poprawnie kompilować za pomocą kompilatora GHC w wersji 8.4 lub nowszej.
- Ocena końcowa z pracowni jest niemalejącą funkcją liczby zdobytych w trakcie semestru punktów.
- W trakcie pracowni studenci korzystając z projektora prezentują swoje rozwiązania wszystkim uczestnikom zajęć.
- Za przedstawienie swojego rozwiązania podczas pracowni można uzyskać od 0 do n dodatkowych punktów, gdzie n jest liczbą punktów przyznaną za zadeklarowanie zadania.
- Zachęca się studentów do przynoszenia na pracownię własnych komputerów. Rozwiązania zadań można także przedstawiać korzystając z komputerów stacjonarnych znajdujących się w pracowni.

Zadania do samodzielnego wykonania

Zadanie 1. Sprawdź czy możesz korzystać z polecenia `ghci` i ewentualnie doinstaluj niezbędne pakiety:

- `ghc` — rozwiązanie minimalne (sam kompilator), zob.:

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

- `haskell-platform` — metapakiet instalujący wraz z `ghc` kompletną Platformę Haskella ("Haskell with batteries included"), zob.:

  `https://www.haskell.org/platform/`

Jeśli nie zawsze masz połączenie z Internetem, to warto też zainstalować pakiety z dokumentacją:

- `ghc-doc` — dokumentacja kompilatora `ghc`,
- `haskell-platform-doc` — metapakiet instalujący wraz z `ghc-doc` dokumentację kompletnej Platformy Haskella.

Zadanie 2. Skonfiguruj swój ulubiony edytor do edycji programów haskellowych. Jeśli używasz:

- `vim`-a, to standardowy plugin `vim`-a zapewnia koloryzowanie składni i automatyczne wcięcia, choć w Internecie są dostępne lepsze rozwiązania, zob. np.

  `https://github.com/neovimhaskell/haskell-vim`

  Zastanów się, jak najwygodniej przełączać pomiędzy edycją programu i jego kompilacją — osobne zakładki w konsoli, dodatkowy plugin pozwalający na uruchomienie kompilatora za pomocą kombinacji klawiszy?

- `emacs`-a, to pakiet `elpa-haskell-mode` oferujący tryb Emacsa dla Haskella może być niezłym rozwiązaniem.

Zadanie 3. Jeśli miałeś już kontakt z Haskellem (np. na Metodach Programowania do roku 2017 lub na Programowaniu funkcyjnym po roku 2017), to zaprogramuj samodzielnie funkcje opisane na końcu bieżącej listy. W przeciwnym razie skopiuj ich implementacje przedstawione przez kolegów i spróbuj je skompilować i uruchomić (nawet jeśli nie wszystko będzie jasne). Przećwicz przy tym komendy `:h`, `:l`, `:m` i `:t` interpretera `ghci`. Rozwiązania przygotuj w postaci pojedynczego pliku w stylu *Literate Haskell* (z rozszerzeniem `*.lhs`).

Zadania do zaprezentowania za pomocą projektora

Zadanie 4. Skonfiguruj i przećwicz sprawne podłączanie swojego komputera do projektora.

Pamiętaj, że rozmiar kątowy piksela obrazu rzuconego na ścianę pracowni jest 30–50% mniejszy, niż rozmiar piksela przeciętnego ekranu laptopa (15.6" Full HD oglądanego z odległości 60 cm), dlatego czcionka w edytorze i terminalu powinna być odpowiednio powiększona. Naucz się powiększać te czcionki. Sprawdź jak uruchomić edytor i terminal w trybie pełnoekranowym tak, aby nie marnować miejsca na zbyteczne paski narzędzi i menu. Pamiętaj, że wyświetlanie na projektorze pulpitu pełnego prywatnych plików, zdjęć, otwartej poczty itp. jest niegrzeczne i krępujące dla słuchaczy — przygotuj pusty pulpit z odpowiednim, neutralnym tłem. Zauważ, że ciemne czcionki na jasnym tle wyglądają w projektorze lepiej, niż — przeważnie używane podczas programowania — jasne czcionki na ciemnym tle.

Pokaż kolegom, jak skonfigurowałeś swoje środowisko do edycji i kompilacji programów haskellowych oraz — jeśli rozwiązałeś poprzednie zadanie — także swoje implementacje prostych funkcji.

Funkcje do zaprogramowania

1. `explode :: Integer -> [Integer]` — funkcja, która zamienia liczbę dodatnią na ciąg cyfr jej rozwinięcia dziesiętnego. Przyda się przy tym funkcja `unfoldr` z modułu `Data.List` i standardowa funkcja `reverse`. Pokaż, jak skorzystać z modułu `Data.List` w `ghci` i jak sprawdzić typ tych funkcji.

2. `implode :: [Integer] -> Integer` — funkcja odwrotna do powyższej. Spróbuj zaprogramować ją z użyciem standardowej funkcji `foldl` (przyda się przy tym standardowa funkcja `fst`).

3. `rot13 :: String -> String` — funkcja kodująca i dekodująca napisy zaszyfrowane szyfrem Cezara z kluczem 13. Tu przyda się pewnie standardowa funkcja `map` i różne funkcje z modułu `Data.Char`. Pokaż gdzie szukać dokumentacji tego modułu.

4. `subsequences :: [a] -> [[a]]` — funkcja tworząca listę podciągów podanej listy. Podciągiem nazywamy tu dowolną listę powstałą przez pominięcie wybranych elementów oryginalnej listy. Lista długości n ma 2^n podciągów.

5. `inits :: [a] -> [[a]]` — funkcja tworząca listę prefiksów podanej listy. Lista długości n ma $n + 1$ prefiksów.

6. `tails :: [a] -> [[a]]` — funkcja tworząca listę sufiksów podanej listy. Lista długości n ma $n + 1$ sufiksów.

7. `segments :: [a] -> [[a]]` — funkcja tworząca listę segmentów podanej listy. Segmentem nazywamy tu dowolną listę powstałą przez odrzucenie dowolnej liczby początkowych i końcowych elementów listy. Lista długości n ma $\frac{n(n+1)}{2} + 1$ segmentów.

8. `permutations :: [a] -> [[a]]` — funkcja tworząca listę permutacji podanej listy. Skorzystaj z serwisu `hoogle.haskell.org` i pokaż jak sprawdzić, jakie jeszcze inne funkcje dostępne w bibliotekach standardowych Haskella mają typ `[a] -> [[a]]`.

9. `merge :: Ord a => [a] -> [a] -> [a]` — funkcja scalająca uporządkowane listy (z powtórzeniami).

10. `msortPrefix :: Ord a => Int -> [a] -> [a]` — funkcja sortująca podaną liczbę elementów podanej listy zgodnie z algorytmem Mergesort. Przyda się tu funkcja `merge` z poprzedniego punktu i standardowa funkcja `drop`.

11. `msort :: Ord a => [a] -> [a]` — funkcja sortująca podaną listę. Przyda się tu funkcja z poprzedniego punktu i standardowa funkcja `length`.

12. `qsort :: Ord a => [a] -> [a]` — funkcja sortująca podaną listę zgodnie z algorytmem Quicksort.

13. `isort :: Ord a => [a] -> [a]` — funkcja sortująca podaną listę zgodnie z algorytmem Insertion Sort.

14. `ssort :: Ord a => [a] -> [a]` — funkcja sortująca podaną listę zgodnie z algorytmem Selection Sort.

15. `elem :: Eq a => a -> [a] -> Bool` — funkcja sprawdzająca, czy podany element występuje na liście.

16. `intersperse :: a -> [a] -> [a]` — funkcja wstawiająca podany element pomiędzy każdą parę sąsiednich elementów podanej listy. Np. wartością `intersperse ',' "abcde"` jest `"a,b,c,d,e"`.