JPP 2018/19 - Zadanie z Haskella

A. Grafika wektorowa i przekształcenia

W tej części tworzymy bibliotekę umożliwiającą definiowanie i przekształcanie prostych rysunków wektorowych.

W pliku `Mon.hs` zdefiniowana jest klasa `Mon`

```
infixl 5 ><
class Mon m where
  m1 :: m              -- element neutralny operacji ><
  (><) :: m -> m -> m  -- >< musi być łączne
```

Zdefiniuj (co najmniej) poniższe typy i funkcje:

```haskell
-- typ R objaśniony w tekście poniżej
type R2 = (R,R)

data Vec    -- wektor 2D
data Point  -- punkt 2D

instance Eq Vec
instance Eq Point
instance Show Vec

point :: R2 -> Point
vec :: R2 -> Vec

instance Mon Vec

data Picture
-- odcinek pomiędzy punktami o podanych współrzędnych
line :: (R,R) -> (R,R) -> Picture
-- prostokąt o podanej szerokości i wysokości zaczepiony w (0,0)
rectangle :: R -> R -> Picture
-- suma (nałożenie) dwóch rysunków
(&) :: Picture -> Picture -> Picture
```

```
type IntLine = ((Int,Int), (Int,Int))
type IntRendering = [IntLine]

-- Obrazowanie przy danym współczynniku powiększenia
-- z zaokrągleniem do najbliższych wartości całkowitych
renderScaled :: Int -> Picture -> IntRendering
```

Uwaga: definicje typów wskazanych jako `data` mogą być alternatywnie wprowadzone przy użyciu `newtype`.

Sugerowane jest przyjęcie

```
type R = Rational
```

ale można użyć innego typu, byle spełnione były warunki zadania.

Wskazówka: dla uzyskania przybliżeń wymiernych funkcji trygonometrycznych można użyć na przykład formuły Bhaskara I:

https://en.wikipedia.org/wiki/Bhaskara_I%27s_sine_approximation_formula i odrobiny matematyki.

Transformacje

Zdefiniuj

```
data Transform
-- przesunięcie o wektor
translate :: Vec -> Transform
-- obrót wokół punktu (0,0) przeciwnie do ruchu wskazówek zegara
-- jednostki mozna sobie wybrać
rotate :: R -> Transform
fullCircle :: R -- wartość odpowiadająca 1 pełnemu obrotowi (360 stopni)
instance Mon Transform


trpoint :: Transform -> Point -> Point
trvec :: Transform -> Vec -> Vec
transform :: Transform -> Picture -> Picture
```

Transformacje powinny odpowiadać zwykłym przesunięciom i obrotom na płaszczyźnie, w szczególności spełniać własności (zdefiniowane w TestTransform):

```
-- składanie translacji
testCompTrans :: R2 -> R2 -> R2 -> Bool
testCompTrans v1 v2 p = trpoint (translate (vec v1) >< translate (vec v2)) (point p)
                        == trpoint (translate (vec v1 >< vec v2)) (point p)
```

```haskell
-- składanie rotacji
testCompRot :: R -> R -> R2 -> Bool
testCompRot r1 r2 p = trpoint (rotate r1 >< rotate r2) (point p)
                      == trpoint (rotate (r1 + r2)) (point p)

-- łączność składania rotacji
testAssRot :: R -> R -> R -> R2 -> Bool
testAssRot a b c p = trpoint (rotate a >< (rotate b >< rotate c)) (point p)
                      == trpoint ((rotate a >< rotate b) >< rotate c) (point p)

-- rotacja o pełny obrót jest identycznością
testFullCircleRot :: R2 -> Bool
testFullCircleRot p = trpoint (rotate fullCircle) (point p) == (point p)
```

B. Wyjście

Program powinien wypisywać linie wynikłe z obrazowania rysunku (dowolnego elementu typu Picture) w uproszczonym podzbiorze język Postscript (zwane dalej treścią), opatrzone prologiem i epilogiem (które są stałe).

Prolog:

```
300 400 translate
```

Epilog:

```
stroke showpage
```

Przykład:

```
300 400 translate

0 0 moveto 100 0 lineto
100 0 moveto 100 100 lineto
200 200 moveto 250 250 lineto

stroke showpage
```

W przypadku napotkania błędu, treść powinna wyglądać następująco

```
/Courier findfont 24 scalefont setfont 0 0 moveto (Error) show
```

(prolog i epilog bez zmian)

Poza pierwszymi dwoma i ostatnimi dwoma komendami, wolno używać tylko komend `moveto` i `lineto` (ewentualnie `rlineto`). Ponadto dozwolone jest jeszcze `show` do wypisania błędu.

Semantyka komend `moveto/lineto` jest analogiczna do opisanej w punkcie C poniżej.

Prolog i epilog są tak skonstruowane, aby dla typowych rysunków efekt renderowania dało się obejrzeć w przeglądarce plików Postscript bądź wydrukować na kartce A4.

C. Wejście - PostKrypt

PostKrypt jest językiem opisu grafiki wektorowej (zbliżonym do podzbioru popularnego języka o podobnej nazwie). Jest to język stosowy, tzn. argumenty i wyniki operacji są przechowywane na stosie, a co za tym idzie, operacje zapisywane są postfiksowo, np. `2 2 add` czy `10 10 moveto`. Próba zdjęcia z pustego stosu jest błędem.

Struktura leksykalna

Program jest ciągiem leksemów rozdzielonych odstępami. Poprawnymi leksemami są literały całkowite (ze znakiem) oraz słowa stanowiące nazwy operacji.

Wejście złożone wyłącznie z ciągu białych znaków jest poprawnym programem, reprezentującym pusty rysunek.

Arytmetyka

Literały w kodzie są całkowite, ich wystąpienia oznaczają włożenie odpowiedniej liczby na stos. Operacje `add sub div mul` są wykonywane na liczbach wymiernych; pobierają argumenty ze stosu i tam odkładają wynik. Kolejność argumentów funkcji odpowiada kolejności ich umieszczania na stosie (np. `30 10 div` daje w wyniku 3).

Próba dzielenia przez 0 jest błędem, który powinien zostać obsłużony przez program (jak opisano powyżej).

Bieżący punkt i bieżąca ścieżka

Przez punkt rozumiemy jego współrzędne `(x,y)`, przez ścieżkę ciąg linii (u nas odcinków, w ogólności mogą to być krzywe). W każdym momencie wykonania programu istotne są pojęcia bieżącego punktu (początkowo nieustalone) i bieżącej ścieżki (początkowo pusta). Jeśli nie wskazano inaczej, operacje graficzne zaczynają się w bieżącym punkcie, przesuwają bieżący punkt we wsþółrzędne swojego zakończenia i dołączają swój wynik na koniec bieżącej ścieżki (nie odkładają go na stosie).

Próba wykonania operacji wymagającej bieżącego punktu, gdy jest on nieustalony, jest błędem (za wyjątkiem `closepath`).

Operacje graficzne

- `moveto` zdejmuje dwie współrzędne ze stosu i ustawia bieżący punkt na te współrzędne; rozpoczyna nową bieżącą ścieżkę składającą się z jednego punktu;
- `lineto` zdejmuje dwie współrzędne ze stosu i dodaje do bieżącej ścieżki (i do rysunku) odcinek od bieżącego punktu; do punktu o tych współrzędnych, odpowiednio przesuwając bieżący punkt.
- `closepath` dodaje do bieżącej ścieżki (i do ryzunku) odcinek od bieżącego punktu do pierwszego punktu ścieżki, odpowiednio przesuwając bieżący punkt. Jeżeli bieżąca ścieżka jest pusta lub złożona z jednego punktu, `closepath` nie ma żadnych efektów.

Próba użycia nieznanej operacji jest błędem.

C.2 Rozszerzenie - Transformacje

Operacja specyfikująca transformację oznacza, że część rysunku wyspecyfikowana po niej powinna podlegać takiej transformacji (oprócz transformacji wyspecyfikowanych wcześniej). Jedynie `closepath` po prostu zamyka bieżącą ścieżkę.

Do takich operacji zaliczamy:

- `translate` zdejmuje dwie współrzędne ze stosu i specyfikuje przesunięcie o taki wektor
- `rotate` zdejmuje liczbę ze stosu i specyfikuje obrót o taki kąt wyrażony w stopniach (dopuszczamy dowolne liczby wymierne, w tym ujemne).

Dokładniej: analogicznie do bieżącej ścieżki, istnieje pojęcie bieżącej transformacji (początkowo identyczność). W efekcie operacji specyfikującej transformację, bieżąca transformacja staje się złożeniem transformacji dotychczasowej i wyspecyfikowanej.

W momencie wykonywania dowolnej operacji używającej współrzędnych, podlegają one przekształceniu przez bieżącą transformację.

W razie wątpliwości można sprawdzić jak transformacje działają w języku PostScript - program w PostKrypcie, po dodaniu prologu i epilogu opisanych w części B, powinien być poprawnym opisem strony w języku PostScript

Program główny

Wywołanie programu głównego z argumentem będącym liczbą naturalną N powinno wczytać ze standardowego wejścia opis rysunku w PostKrypt zgodnie z opisem w części C i wypisywać na standardowe wyjście, zgodnie z opisem w części B, efekt obrazowania tego rysunku powiększonego N razy (przy pomocy funkcji `renderScaled`), z zaokrągleniem współrzędnych początków i końców linii do najbliższej liczby całkowitej.

Jeżeli argument nie jest liczbą całkowitą, program powinien wypisać informację o poprawnym sposobie wywołania.

Wywołanie programu bez argumentu powinno przyjąć powiększenie N=1

Przykłady

W plikach `goodNN.in` są przykłady poprawnych programów, odpowiadajęce im wyniki są w plikach `goodNN.ps`

W plikach `badNN.in` są przykłady błędnych programów.

Wymagania techniczne

Należy oddać dwa pliki: `Lib.hs` zawierający moduł realizujący część A (po zaimportowaniu do programu TestTransform powinny działać wszystkie testy), oraz Main.hs zawierający program główny i ewentualnie funkcje pomocnicze.

Na początku każdego z oddawanych plików **MUSI** znajdować się informacja o ich autorze (co najmniej inicjały i nr indeksu)

Testy i program będą uruchamiane z użyciem plików `Mon.hs` i `TestTransform.hs` stanowiących część zadania.

Program powinien kompilować się przy użyciu GHC 7.6 lub 8.2. W razie wątpliwości punktem odniesienia jest maszyna `students`.

GHC 7.6 jest tam w /usr/bin, a 8.2 w `/home/students/inf/PUBLIC/MRJP/ghc-8.2.2/bin`.

Dopuszczalne importy

Poza modułami składającymi się na zadanie i rozwiązanie, dopuszczalne są następujące importy (z pakietów base, mtl, QuickCheck, HUnit, hspec)

```
Prelude
Control.*
```

```
Data.*
System.*
Test.*
Text.*
```

Użycie innych importów (zwłaszcza spoza wymienionych pakietów) wymaga uzyskania uprzedniej zgody.

Ocenianie

Zadanie będzie oceniane nie tylko pod kątem poprawności, ale również czytelności kodu i wykorzystania poznanych mechanizmów języka.

Zadanie MUSI być rozwiązane samodzielnie. Wszelkie zapożyczenia muszą być wyraźnie zaznaczone z podaniem źródła.

Zabronione jest oglądanie rozwiązań, jak również wszelkie formy udostępniania własnego rozwiązania innym osobom.

Rozwiązania niesamodzielne będą oceniane na 0p. W wypadku stwierdzenia istotnego podobieństwa dwóch (lub większej liczby) rozwiązań, wszystkie będą oceniane na 0p.

Rozwiązania częściowe

Warunkiem uzyskania punktów za rozwiązanie częściowe (nie obejmujące całego zakresu zadania, bądź nie przechodzące wszystkich wymaganych testów) jest wyraźne opisanie zrealizowanego zakresu w komentarzu na początku oddawanego pliku.

Rozwiązaniom częściowym przyznana zostanie znacznie mniejsza liczba punktów niż rozwiązaniom pełnym (zasadniczo najwyżej 50%). Wyjątkiem jest pominięcie części C.2: Rozszerzenie - Transformacje, której pominięcie będzie się wiązało z utratą ok 20% punktów.