{- Zayadur Khan 01/29/17 CIS-252 M001 Assignment 2 -}

import Data.Char

-- compareChars returns a string indicating how many of the

-- characters a, b, c are equal to one another

compareChars :: Char -> Char -> Char -> String

compareChars a b c

| (a == b) && (b == c) = "All equal"

| ((a /= b) && (b /= c)) && (a /= c) = "All distinct"

| otherwise = "Two match"

-- combine returns an integer combining three integers

-- between 0 and 9

combine :: Int -> Int -> Int -> Int

combine x y z

| ((x >= 0) && (y >= 0)) && (z >= 0) = (x \* 100) + (y \* 10) + (z)

| otherwise = (-1)

-- splitFloat splits a float value into its whole number part

-- and fractional part, returning the respective integer and float

splitFloat :: Float -> (Integer, Float)

splitFloat num = (int, fl)

where

int

| num < 0 = fromInteger (ceiling num)

| otherwise = fromInteger (floor num)

fl

| num < 0 = fromInteger int - num

| otherwise = num - fromInteger int

{- Problem 4 -> smartphone company pricing plans

standard

| base price = 50

| extra devices after 1 device = 10 / device

| extra data after 4GB = 15 / GB

power-user

| base price for 1 device = 80

| extra devices = 8 / device

| extra data after 10GB + 2GB / device = 20 / GB

new customer discount = $25 -}

-- stdCost calculates monthly cost under standard pricing plan

-- for customer with 'dev' devices who uses 'gb' gigabytes of data

-- over a month, where 'new' is true if customer is new

-- return -1 if dev < 1 or gb < 0

stdCost :: Int -> Int -> Bool -> Int

stdCost dev gb new

| (dev < 1) || (gb < 0) = -1

| new == False = cost

| otherwise = cost - 25

where

cost

| (dev == 1) && (gb <= 4) = 50

| gb <= 4 = 50 + (10 \* (dev - 1))

| otherwise = 50 + (10 \* (dev - 1)) + (15 \* (gb - 4))

-- powerCost calculates the monthly fee under the power-user

-- plan for a customer with 'dev' devices who uses 'gb' gigabytes

-- of data over a month

-- return -1 if dev < 1 or gb < 0

powerCost :: Int -> Int -> Bool -> Int

powerCost dev gb new

| (dev < 1) || (gb < 0) = -1

| new == False = cost

| otherwise = cost - 25

where

cost

| (dev == 1) && (gb <= 4) = 80

| gb <= (10 + 2 \* dev) = 80 + (8 \* (dev - 1))

| otherwise = 80 + (8 \* (dev - 1)) + (20 \* (gb - (10 + 2 \* dev)))

-- bestPlan determines which pricing plan is the better choice

-- for an established customer with 'dev' devices who expects

-- to use 'gb' gigabytes of data in a given month

-- return string for respective pricing policy

-- return -1 if dev < 1 or gb < 0

bestPlan :: Int -> Int -> String

bestPlan dev gb

| (dev < 1) || (gb < 0) = "Try again"

| (stdCost dev gb False) < (powerCost dev gb False) = "Standard"

| (stdCost dev gb False) > (powerCost dev gb False) = "Power User"

| otherwise = "Same cost"